



Nemzeti Közszolgálati Egyetem
Hadtudományi és Honvédtisztképző Kar
Intézményi Tudományos Diákköri Konferencia

Tanító minták generálása és menedzselése szimulációs környezetből

Készítette: Szántó Norbert, Bajka Adorján, Bárdi Olivér

Intézmény: Sapientia EMTE

Kar: Marosvásárhelyi Kar

Szakirány: Automatika és alkalmazott informatika

Konzulens neve, beosztása: Dr. Brassai Sándor Tihamér, Dr. Németh András,

Neptun kód: NZM3PX, I6NOGE, T6U5BZ

2023.

Tartalom

Tartalom	2
Bevezető	4
1. A projekt műszaki háttere.....	7
1.1. Hasonló kutatások.....	7
1.2. Technológiai megvalósítás.....	8
1.3. Szimuláció menete.....	10
1.4. CVAT – Címkéző program	11
1.5. Címkék feltöltése	12
1.6. Koordináta-rendszerek.....	13
2. Objektumok és utasítások	15
2.1. Szintmagasság meghatározása.....	15
2.2. Elhelyezési hibák.....	16
2.3. Objektum lerakási módszerek	17
2.4. Parancsértelmező	19
2.4.1. Utasítások.....	21
3. Terepek	24
3.1. Terepek kialakítása.....	24
3.1.1. Terepek anyaga	25
3.2. Terepek kialakítására használt szoftverek	26
3.2.1. Unreal Engine színtszerkesztő	26
3.2.2. Cesium.....	28
3.2.3. Instant Terra	29
3.2.4. Open Drone Map (ODM)	30
3.2.5. Gaea.....	31
3.3. Terepek menedzselése.....	32

4. Objektumok menedzselése, bejárások	34
4.1. Objektumok projektbe való integrálása	34
4.1.1. Objektumok konvertálása	35
4.1.2. Objektumok importálása és menedzselése	35
4.2. Bejárások	37
4.2.1. Drón mozgatása.....	38
4.2.2. Területbejárás	38
4.2.3. Objektum bejárás	40
4.2.4. Kamera mozgatása.....	41
5. Automatikus címkézés	43
5.1. Címkézési algoritmusok	44
5.1.1. Minimum pont szerinti oldalválasztás	44
5.1.2. Objektum élei szerinti oldalválasztás.....	46
5.1.3. Objektum tengelye szerinti oldalválasztás.....	47
5.2. Címke oldalainak meghatározása	48
5.3. Címkék sarokpontjainak meghatározása.	49
6. Mérések.....	51
Konklúzió.....	54
Felhasznált irodalom	56

Bevezető

Sajnálatos módon a 2023-as év sem hozta meg a békét a világnak, hiszen nemcsak hogy a tavaly februárban kirobbant orosz-ukrán háború sem ért véget, de 2023. október 7-én a közel-keleti térségben egy újabb véres fegyveres konfliktus vette kezdetét, amikor a Hamász terrrorszervezet Izrael történetének eddigi legbrutálisabb terrortámadását hajtotta végre, válogatás nélkül legyilkolva több mint ezer izraeli, palesztin és külföldi polgárt, megdöntve ezzel a legyőzhetetlen zsidó állam mítoszát. Természetesen, ahogy az várható volt, a válasz nem váratott sokáig magára és a kegyetlen harcok ma már Gáza területén zajlanak, naponta több száz újabb áldozatot követelve.

Az egyre instabilabbnak tűnő globális biztonsági környezet hatására az egyes államok és katonai szövetségek egyre nagyobb erőforrásokat fordítanak haderejük fejlesztésére, melynek során a súlypont egyértelműen eltolódni látszik a pilóta nélküli légi járművekre (UAV¹) alapozott képességek irányába. A drónok alkalmazásának egyik legfontosabb területe az ellenséges csapatok elhelyezkedésének, harcérétkének, létszámának, technikai felszerelésének, harcrendjének és tevékenységének felderítése, ami alapján meg tudjuk tervezni saját csapataink műveleteit. Az ilyen légi robotok jelentősége megkérdőjelezhetetlen, hiszen egyrészt alkalmazásuk során nincs szükség személyzetre, azaz nem veszélyeztetjük saját katonáink életét, másrészt a hagyományos repülőeszközöknél kisebb méretűeknek, alacsonyabb költségűeknek és magas technikai felszereltségűeknek köszönhetően hatékony alternatívát kínálnak a légi műveletek és szárazföldi csapatok támogatása során, támadó, védekező és elterelő feladatokban egyaránt. [1] [2]. A mesterséges intelligencia (AI²) az elmúlt években kiváló lehetőséget kínált az UAV a rendszerek autonómia szintjének növelésére, valamint a gyűjtött információk feldolgozására, ezzel is megkönnyítve az operátorok, illetve az értékelő-elemző munkát végzők feladatát.

A mesterséges intelligencia kutatások egyik leggyorsabban fejlődő területe a neurális hálózatok, amelyek az emberi agyhoz hasonló működésre képesek. A neurális hálózatok összekapcsolt mesterséges neuronokból állnak, amelyek valamilyen hierarchikus struktúrában

¹ Unmanned Aerial Vehicle

² AI – Artificial Intelligence

dolgozzák fel az információkat, és rétegenként egyre összetettebb jellemzőket vonnak ki a bemeneti információkból. Ez lehetővé teszi, hogy az emberek tanulásához hasonló módon fejlesszék képességeiket a rendelkezésre álló adatok felhasználásával. A neurális hálózati algoritmusok egyik fontos típusa a mélytanulási modellek csoportja, amely többretegű mesterséges idegsejteket használ az adatok összetett kapcsolatainak modellezésére. Ez forradalmasította az elmúlt években az AI-kutatást, a gépi látást, a természetes nyelvi feldolgozást és a beszédfelismerést egyaránt. [3] A mélytanulási modellek kínálják napjainkban a leghatékonyabb megoldást többek között a képosztályozásra és az objektumészlelésre. A tanítás elengedhetetlen feltétele ugyanakkor, hogy megfelelő mennyiségben álljon rendelkezésre adat, amelyek alapján a mélytanulási modellek fontos információkat szintetizálhatnak, és képessé válnak felismerni akár az emberek számára nehezen észlelhető mintákat is.

A mesterséges intelligencia azonban a harcászati felderítés mellett a katonai feladatok egyre szélesebb spektrumát támogathatja a jövőben, mint például a katonai robotikai rendszerek, szárazföldi, vízi, víz alatti, légi, vagy akár földalatti harci robotok, exoskeletonok, harcszimulátorok alkalmazása, továbbá a hírszerzés, célpontkutatás, összadatforrású felderítés és parancsnoki döntéshozatal támogatása. [4] [5]

Tevékenységünket a TKP2021-NVA-16 projekt keretében működő „Mesterséges intelligencia, alternatív valóságok és rádiófrekvenciás eljárások védelmi célú alkalmazása” kiemelt kutatási területen belül a „Mesterséges intelligencia alapú, drónplatformra telepített felderítőrendszer” fejlesztését végző kutatócsoportban végezzük. Feladatunk a virtuális szimulációs környezetből származó mérések eredményeinek felhasználásával tanító minták generálása, és a kapcsolódó feladatok menedzselése.

Neuronhálókkal történő képfelismerő rendszerek nagyon elterjedtek napjainkban, de fontos azt látnunk, nagyon sok adatra van szükség, hogy ezek a rendszerek helyesen tudjanak működni. A neuronhálók tanításához szükségünk van előre megcímkézett adatokra. A képfelismerés esetében tehát képekre, továbbá azokra az információkra, hogy azokon milyen

objektumok helyezkednek el és pontosan hol. Ezekre különböző leírási struktúrák léteznek, mint például a YOLO³, PASCAL VOC⁴ vagy a CVAT⁵.

Ilyen adathalmazok létrehozása a gyakorlatban elég körülményes, a képek elkészítése sok időt vesz igényben és gyakran nem is állnak rendelkezésünkre olyan objektumok nagy számban és változatos formában, amelyekről tanító halmazt szeretnénk generálni. A másik nagy probléma, hogy a rendelkezésünkre álló képeket címkéznünk is kell, amely során azokon egyenként be kell jelölni az objektumokat és megmondani az objektum típusát is.

A rendelkezésünkre álló korlátozott számú kép és a címkézés hosszas folyamata miatt gondoltunk arra, hogy egy 3D drón szimulációt készítsünk a tanítóhalmazok generálására és a neuronhálók működésének ellenőrzésére. Első lépésben az volt a célunk, hogy egy terepen különböző objektumokat helyezzünk el, majd a drón bejárja a terepet az általunk megadott módon és képeket készít pontosan úgy, mint azt egy tényleges repülés során tennénk. A drón kamerája által készített képek címkézése automatikusan történik a szimuláció során. Ilyen megoldással kellő mennyiségű, változatos adatot tudunk generálni a neuronháló tanítására. Másik fontos szempont a fejlesztés során az volt, hogy a lehető legkisebb emberi közreműködéssel lehessen a szimulációkat futtatni.

³ You Only Look Once – Csak egyszer nézd meg

⁴ PASCAL Visual Object Classes – Pascal vizuális objektum osztályozás

⁵ Computer Vision Annotation Tool – Gépi látás címkéző eszköz

1. A projekt műszaki háttere

1.1. Hasonló kutatások

Több kutatás foglalkozik drón alapú képfelismerő rendszerekkel, valamint a neuronháló tanításához szükséges tanítóhalmazok generálásával. Az eredményekből látszik, hogy az algoritmusok hatékonysága nagyban függ különböző tényezőktől, mint az időjárás, vagy a drón repülési magassága, amelyből a képet rögzíti. Ezen kutatásokban is kiemelték azt a problémát, hogy a rendszer tanításához nagy mennyiségű címkézett adatra van szükség és ilyenből kevés szabadon elérhető van az interneten [6] [7].

Több publikációban is kiemelték, hogy az objektumok felismerése nagyban függ attól, mennyire van az objektum más objektumok által. Ezt egy olyan környezetben különösen fontos figyelembe venni, ahol több mozgó objektum van, amelyek helyzete folyamatosan változik és ki is takarhatják egymást. Ilyen eset például, amikor járműveket próbálunk felismerni mozgás közben [8] [9]. Az objektumok előfordulási gyakorisága is fontos paraméter, mert ha egyféle objektum többször is szerepel a tanítóhalmaz mintái között, akkor azt a rendszer majd könnyebben is fogja felismerni, ami torzítja az eredményeket. Amennyiben az objektum kilóg a képből, jelentősen csökken a felismerés valószínűsége.

A drónok elterjedésével egyre többen kezdtek el foglalkozni drónok elleni védelmi rendszerekkel [10], ahol az algoritmusnak a képeken magukat a drónokat kell felismernie. A tanítás során vegyesen használtak szimulációból kinyert, valamint valós adatokat. A valós adatok hátránya az volt, hogy jelentős humán erőforrásra volt szükség a címkézésükhöz, és így időben is lassabb volt a folyamat, ugyanakkor a valós adatok a kutatás szerint jobbnak bizonyultak, mivel a rendszernek a végén valós képek alapján kell majd működnie, és a szimuláció által generált képek nem minden esetben voltak teljes mértékben valóságűek. Kisebb távolságra lévő drónok felismerése során jobb eredmények jelentkeztek, mivel ezek a képek nagyobb felbontásban álltak rendelkezésre, tehát több részlet alapján tudta a rendszer felismerni az adott objektumot [8]. A vizsgálatokat különböző mélytanulási algoritmusokkal is

elvégezték, mint a YoloV2, R-CNN⁶, CornerNet, RRPN⁷, SSD⁸ és a végén összehasonlították az eredményeket.

1.2. Technológiai megvalósítás

A szimuláció megvalósítására az Epic Games által készített Unreal Engine szoftvert használtuk a Microsoft AirSim nevű Plugin-jával.

Az Unreal Engine egy főként számítógépes játékok megalkotására létrehozott játékmotor, de filmek készítésénél is gyakran használt megoldás ez, főként napjainkban, mikor egyre több vizuális effektus jelenik meg a filmvásznon [11]. Másik elterjedt felhasználása a virtuális valóság és különböző más alternatív valóság szimulációk készítése. Azért esett a választásunk az Unreal Engine környezetre, mivel a hozzáférhető drón szimulációs szoftverek, mint például a Gazebo, inkább a drónok működésének fizikai szimulációjára fókuszálnak, miközben a környezet kevésbé hasonlít egy valós környezetre. Az Unreal Engine alkalmas arra, hogy nagyon valóság-hű terepeket lehessen benne létrehozni, ami fontos annak érdekében, hogy valóság-hű képeket tudjunk generálni az adathalmazunk számára.



1. ábra Unreal Engine felület

⁶ Region based Convolutional Neural Networks – Régió alapú konvolúciós neurális hálózatok

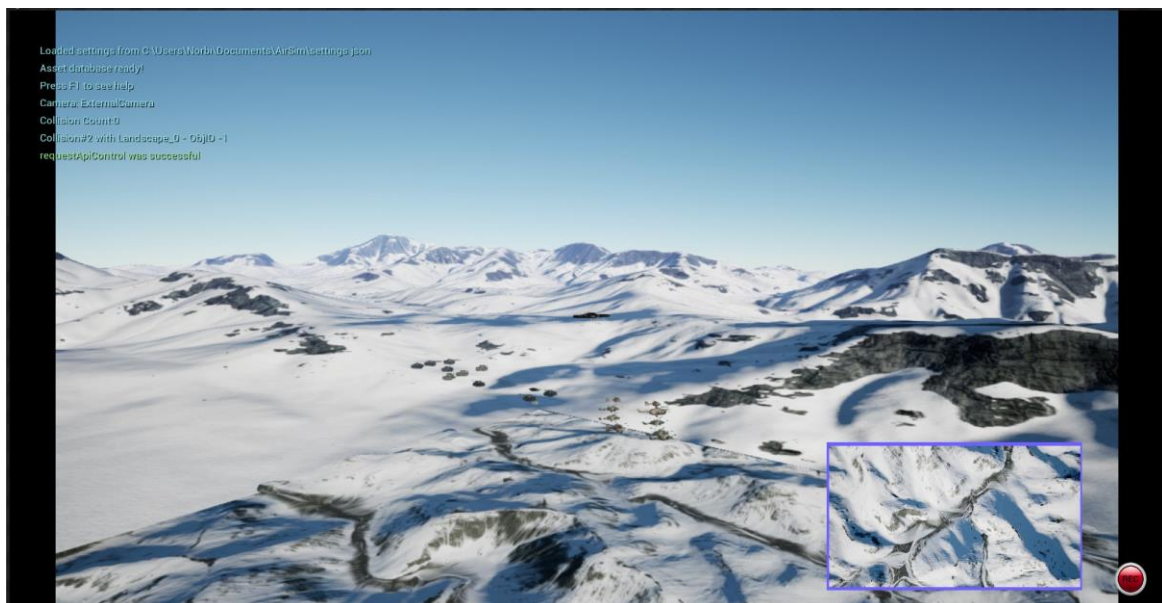
⁷ Radar Region Proposal Network – Kereső régió ajánló hálózat

⁸ Single Shot Detector – Egyutas érzékelő

Az 1. ábrán az Unreal Engine grafikus felülete látszik, ahol éppen az egyik általunk használt térkép van megnyitva.

Az AirSim egy open source projekt, amit az Unreal Engine-be egy Plugin-ként lehet integrálni [12]. Az AirSim tulajdonképpen egy előre leprogramozott drón szimulációs szoftver, ami nagyban megkönnyíti egy hasonló alkalmazás elkészítését. Tartalmazza a drónt és az irányításához szükséges interfészeket, mint PX4 autópilóta, vagy egy API⁹, amit Python vagy C++ programozási nyelvekben lehet integrálni. Az AirSim Plugin nagyban megkönnyítette a szimuláció fejlesztésének menetét, mivel a drón vezérlése és sok más hasznos funkció már integrálva volt ebbe a modulba, így ezeket nekünk már nem kellett lefejlesztenünk.

A szimuláció fejlesztésében az Unreal Engine 4.27-es verzióját használtuk, aminél van már újabb verzió az Unreal Engine 5.3. Az az oka, hogy nem az új verziót használtuk, hogy az AirSim még nem támogatja az új Unreal Engine verziót és ez a modul a fejlesztésben elengedhetetlen volt annak érdekében, hogy ne az alapoktól kelljen lefejlesztenünk egy teljes drónos szimulációt. A későbbiekben még lehetséges, hogy áttérünk az AirSim egy újabb verziójára, a Project AirSim-re, ami jelenleg még fejlesztés alatt áll, és publikusan még nem elérhető.



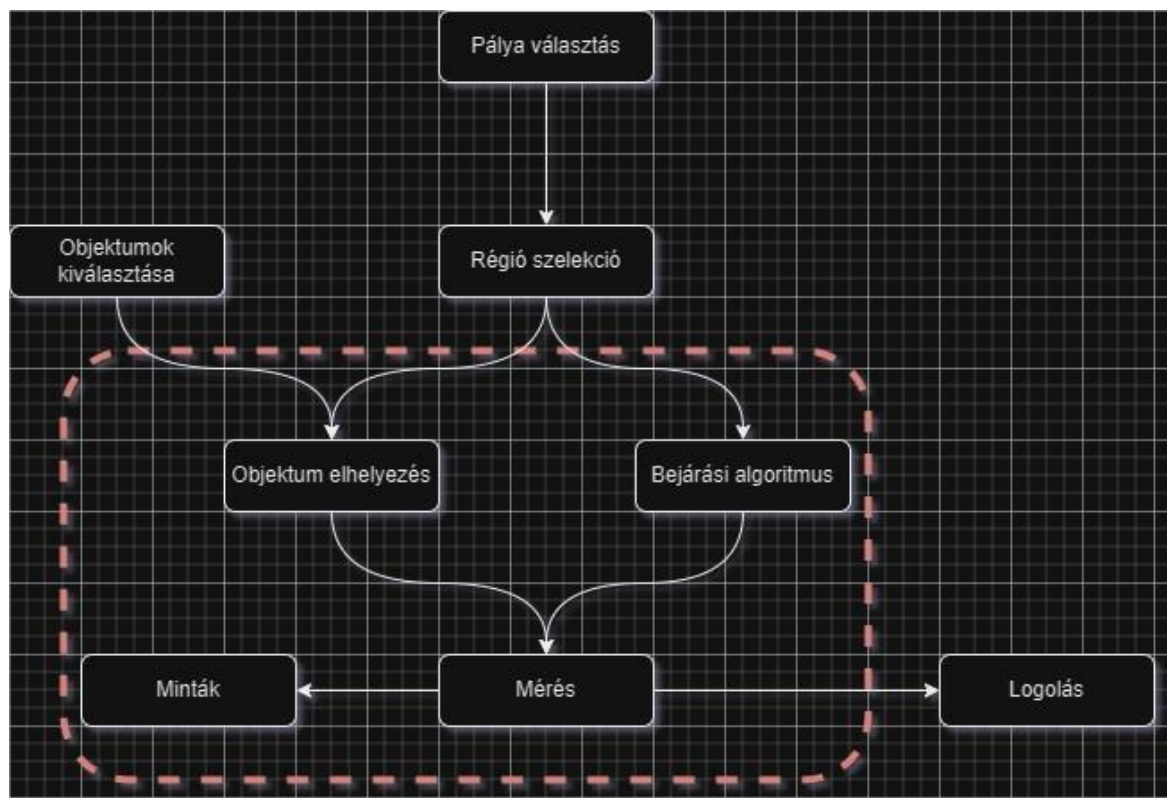
2. ábra AirSim futás közben

⁹ Application Programming Interface – Alkalmazás programozási interfész

A 2. ábrán egy AirSim szimuláció látható, ahol a főképernyőn egy a drónt követő külső kamera képe, míg a jobb oldali alsó ablakban a drón kamerájának képe látható.

1.3. Szimuláció menete

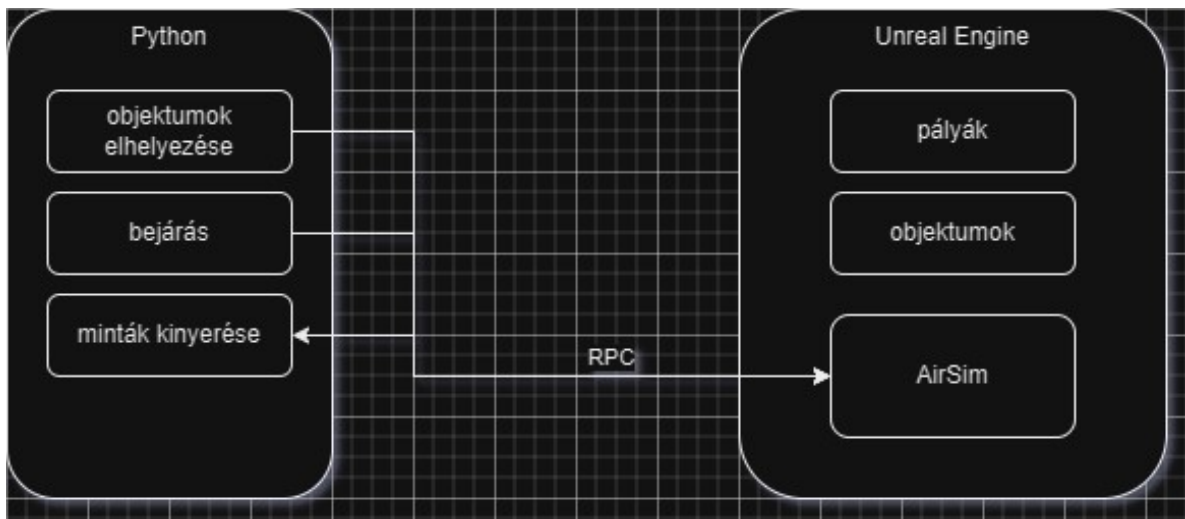
A szimuláció futtatásának folyamatában (3. ábra) első lépésként meghatározzuk a szimulációs paramétereket, amibe beletartozik a pálya, ahol a mérést végrehajtjuk és azon belül a pontos terület, amit a drón be fog járni, majd a terepen az objektumok elhelyezését jelöljük ki, azaz hogy az egyes pozíciókban milyen objektumok fognak megjelenni. A második lépés a program tényleges futásának folyamata, ahol a megadott paraméterekkel az eszköz végrehajtja a szimulációt és kimentí a mérés során generált képeket. Az ennek során megjelenő objektumok már címkézve kerülnek lementésre. A szoftver ezen felül még elmenti a drón pozícióját és orientációját is a képek készítésének pillanatában.



3. ábra Szimuláció menete

A szimuláció két nagy moduljának felépítése látszik a 4. ábrán. A Python modul végzi a drón vezérlésével kapcsolatos feladatokat, mint a terep vagy az objektumok bejárása. Az objektumok lerakási konfigurációk betöltése, a szimulációból a minták kinyerése is ezen modul feladata. Az Unreal Engine a háromdimenziós környezet megjelenítésével foglalkozó modul,

ami tartalmazza az AirSim Plugin-t. A két egység a szimuláció folyamán az AirSim API segítségével kommunikál egymással RPC¹⁰ protokollon.



4. ábra Modulok felépítése

1.4. CVAT – Címkéző program

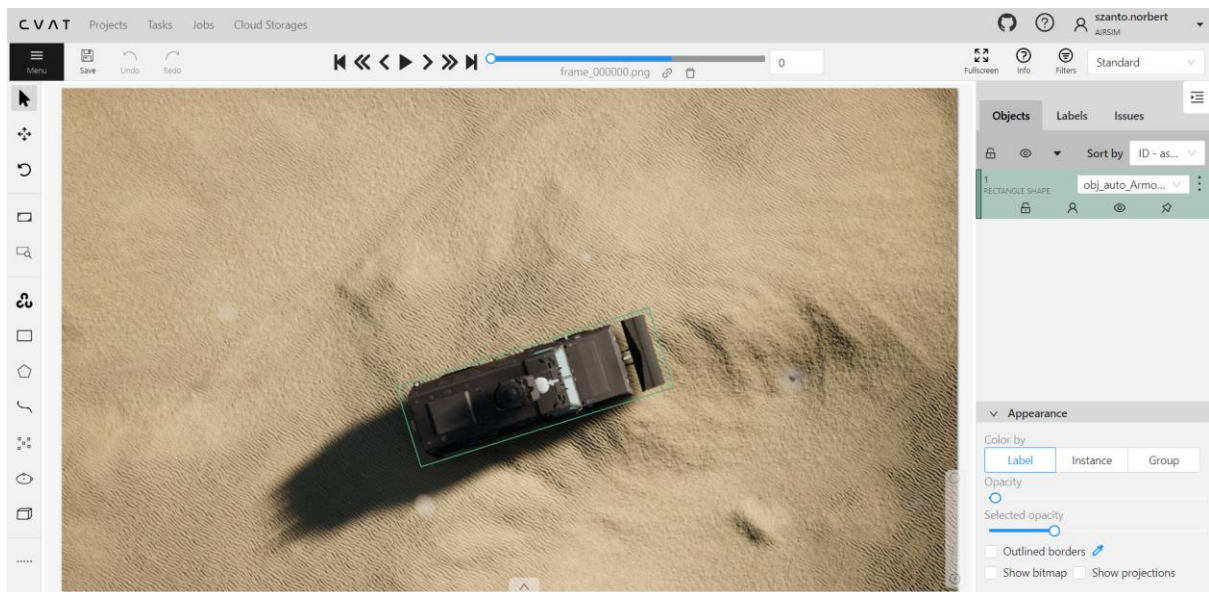
A CVAT vagyis a Computer Vision Annotation Tool egy gépi látási algoritmusok tanítását segítő eszköz, amelynek működéséhez a címkézett adatok elengedhetetlenül szükségesek a neuron hálók tanítása során. A szoftver különböző címke megjelöléseket támogat, mint például a négyzet és annak elforgatott változata, vagy a poligon. Azért szükségesek a különböző keretezések, mert egyes tanítási algoritmusok például csak négyszög alakú objektumkeretezéseket támogatnak, és azokban az esetben nem lehetséges a poligon kijelölés használata. Különböző annotáció formátumok kimentésére is lehetőség van a szoftverben, mint a COCO¹¹, YOLO, PASCAL VOC vagy a CVAT formátum. Ezek a formátumok azért előnyösek, mert így különböző féle tanítási módszerek esetén nem kell nekünk az adatokat átkonvertálni más formátumokba, hanem ezt már a szoftver maga megoldja.

A program egy webes felületet használ, aminek az az előnye, hogy nem kell letöltenie minden felhasználónak lokálisan a saját gépére, hanem a böngészőből egyszerűen el lehet érni. Alapvetően ingyenesen lehet használni a programot a CVAT weboldalán, de lehetőség van rá,

¹⁰ Remote Procedure Call – Távoli eljárashívás

¹¹ Common Objects in Context – Gyakori objektumok a környezetben

hogy saját szervert hozzunk létre, ami a mi esetünkben történt az adatok biztonsága érdekében, valamint a limitált mennyiségű ingyenes tárhely miatt. A szerver telepítése Docker konténerből is lehetséges, ami nagyban leegyszerűsíti a telepítés folyamatát.



5. ábra A CVAT szoftver felülete

A webes felület könnyen átlátható, ahogy az 5. ábrán látszik. A képernyő közepén az adott kép látható, amit címkézni szeretnénk, bal oldali panel a különböző kijelölési módokat tartalmazza és a jobb oldalon a címkék listája látható.

1.5. Címkék feltöltése

A szimuláció által címkézett adatokat fontos volt, hogy vissza tudjuk tölteni a címkéző programban, amire a CVAT lehetőséget biztosít egy Python SDK¹² segítségével [13]. Ezután létrehoztuk saját Python függvényeinket integrálva azokat az SDK által megadott függvények közé, melyek elvégzik a szoftveres autentikációt és létrehoznak új „task”-okat a programban az általunk megadott struktúra szerint, majd a képeket és az azokhoz tartozó annotációkat feltöltik a rendszerbe. A címkék feltöltése egy adott bejárást követően történik meg és az a

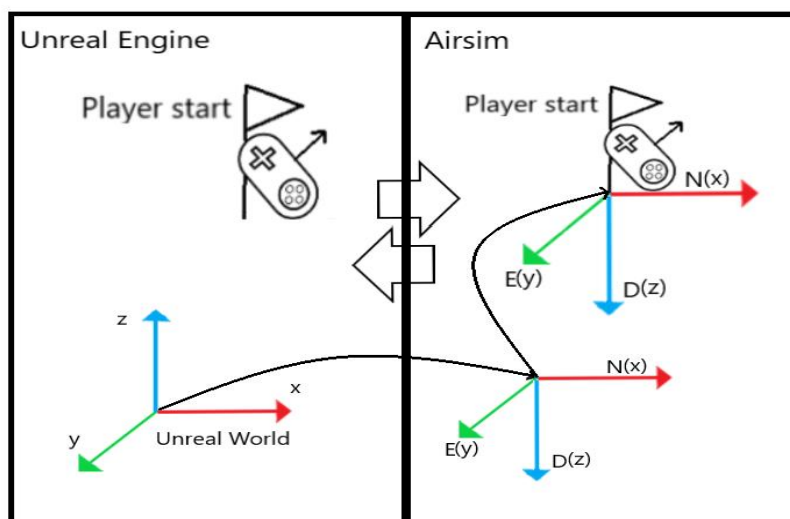
¹² System Development Kit – Rendszerfejlesztő csomag

programnak egy külön szálán fut annak érdekében, hogy közben a szimuláció futhasson tovább, amíg a képek feltöltésre nem kerülnek.

1.6. Koordináta-rendszerek

Az Unreal Engine játékmotor balsodrású xyz-koordináta-rendszert használ, tehát x koordináta előre, y jobbra és z felfelé mutat. A koordináták égtáj szerinti angol megnevezése szerint, ezt a koordináta-rendszert NEU (North, East, Up) koordináta-rendszernek nevezzük. Az Unreal Engine játékmotor standard mértékegysége centiméter. Az objektumok rotációit XYZ konvenciót alkalmazó Euler szögekkel kell megadni, ahol az x tengely szerinti rotáció matematikailag pozitív, míg az y, z tengelyek körüli rotáció matematikailag negatív. A rotációs szögeket fokban kell megadni.

Az AirSim beépülő modul jobb sodrású NED (North, East, Down) koordináta-rendszert alkalmaz, ahol az északra mutató tengelye egybeesik az Unreal Engine x tengelyével. A rotációkhoz ZYX konvenciót alkalmazó Euler szögeket használ. Az Unreal Engine-el ellentétben itt a standard mértékegység a méter, de a szögeket fokban kell megadni.



6. ábra Alkalmazott koordináta-rendszerek viszonya

A 6. ábra szemlélteti az Unreal Engine és AirSim által használt koordináta-rendszereket és a közöttük levő kapcsolatot. Látható, hogy az AirSim koordináta-rendszer origója és az Unreal Engine origója nem egyezik meg. Az AirSim koordináta-rendszer az Unreal Engine pályán elhelyezett 'player start'-hoz helyezi (0, 0, 0) kezdeti pontját. A két koordináta-rendszer közötti konverzió nagyon egyszerű, a koordinátákat felírjuk vektoriális formában és

megszorozzuk a transzformációs mátrixal, majd korrigáljuk a ‘player start’ koordinátaival, ahogy az lent is látható. [14]

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix} \quad (1.)$$

A drónszimuláció kialakítása során fontos volt, hogy valós méréseket is fel tudjunk dolgozni szimulációs környezetünkben, ezért geodetikus koordináta-rendszerből érkező adatokat is tudni kellett feldolgozni a rendszerünknek. A geodetikus koordináta-rendszer a föld hosszúsági-, szélességi körei és a tengerszinttől mért magasságból áll. Az első két érték fokban, míg a harmadik méterben van megadva.

Az ENU koordinátából a geodetikus koordináta-rendszerbe a következő képlet alapján lehet átalakítani, ahol ‘ α ’ a szélességi kör fokban és ‘ θ ’ a hosszúsági kör fokban meghatározva.

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} -\sin(\alpha) & \cos(\alpha) & 0 \\ -\sin(\theta)\cos(\alpha) & -\sin(\theta)\sin(\alpha) & \cos(\theta) \\ \cos(\theta)\cos(\alpha) & \cos(\theta)\sin(\alpha) & \sin(\theta) \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix} \quad (2.)$$

A 2. képlet segítségével megoldható a szimulációs rendszer különböző komponenseinek összehangolása.

2. Objektumok és utasítások

A szimuláció megvalósítása során fontos szempont volt számunkra, hogy a program futtatása lehető legnagyobb mértékben automatizálva legyen, ezért az nem is jöhetett szóba, hogy az objektumokat kézzel helyezzük le egyenként a pályára az Unreal Engine szerkesztőjében. Másik fontos szempont volt, hogy az objektumok elhelyezkedését el is tudjuk tárolni valamilyen adatstruktúrába.

A lerakást először az általunk használt AirSim Python API-ban beépített függvénnyel szerettük volna megoldani, de ott problémába ütköztünk a megoldás során. Az API függvény az objektumok elhelyezéséhez X, Y és Z koordinátát kér. Amennyiben mi egy X, Y koordinátára akartunk objektumot lehelyezni, meg kellett határozni a magasságot is abban a pontban, amit a Python környezetből nem lehetett megvalósítani. Végül Unreal Engine C++ függvénnyel oldottuk meg a lerakási algoritmusokat, melyek az első verzióban a szimuláció indításának pillanatában lettek meghívva. A későbbi fejlesztésekben egy AirSim API függvényt hoztunk létre erre a célra. Azért volt szükség az API függvényre, mivel hogyha csak a szimuláció indításakor tudunk lerakni új objektumokat, akkor nem lehet egymás után több szimulációt lefuttatni megszakítás nélkül különböző konfigurációkkal.

2.1. Szintmagasság meghatározása

A pályák felülete nem sík, hanem vannak rajta hegyek, völgyek, dombok és egyéb más domborzati formák is. Ebből kifolyólag meg kellett oldani, hogy bármilyen koordinátára le tudjuk kérni a felület pontos magasságát, amit az Unreal Engine-be beépített LineTrace függvény segítségével sikerült megvalósítani. A LineTrace függvény működésének alapja, hogy van egy kezdő pont és van egy végső pont, a függvény pedig azt nézi, hogy a kezdő pontból elindulva a végpontig egy egyenes mentén hol találkozik valamilyen olyan objektummal, aminek van fizikai kiterjedése. [15]

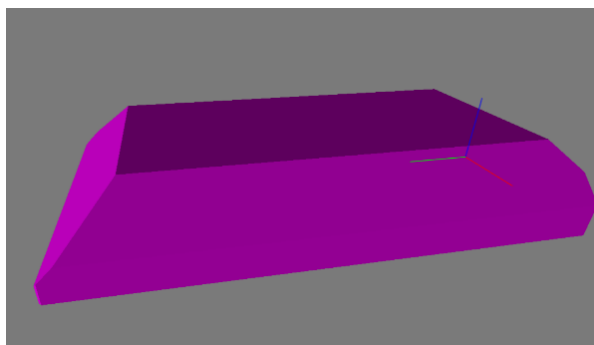
Az 7. ábrán kirajzoltuk a szimulációban azt az egyenes, ami meghatározza a terep magasságát a pálya felületén. Jól látszik, hogy a függvény helyesen kapja meg a felület magasságát az adott pontban.



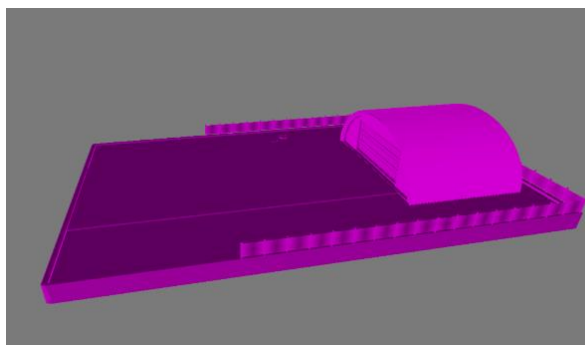
7. ábra LineTrace

2.2. Elhelyezési hibák

Az objektumok elhelyezésénél előfordult egy hibajelenség, mivel a beépített LineTrace függvény nem helyesen térítette vissza a terep magasságát. Ez azért fordult elő, mert a függvény alapbeállításban egy egyszerűsített modellt használt az objektumok kiterjedésének meghatározására.



8.1. ábra Hangár egyszerűsített modellje



8.2. ábra Hangár összetett modellje

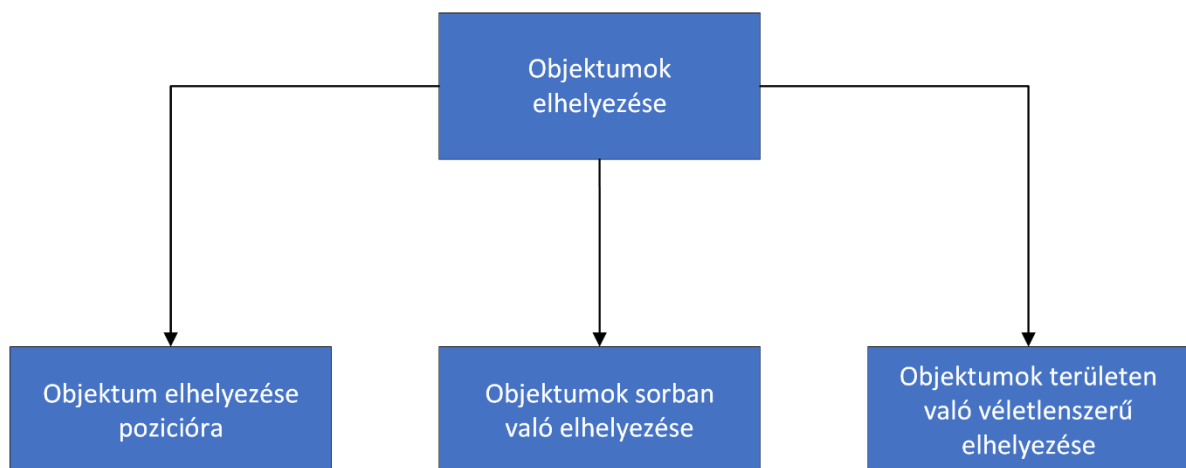
Az 8.1. ábrán egy hangár leegyszerűsített modellje szerepel, amin látszik, hogy egy nagyon kevés szögből álló poligon jelenik meg. Az Unreal Engine azért használja ezt a modellt, mert így kevesebb számításba kerül az objektumok fizikai kiterjedésének meghatározása és ezzel sok erőforrást meg tud takarítani. Akkor lehet ez fontos, a mikor minden képkockában végre kell hajtani ilyen a műveletet. Ilyen eset lehet, amikor figyelni kell arra, hogy a drón neki

ütközött egy falnak, és nem szabad engedni, hogy ebben az esetben tovább menjen, mivel a valóságban sem mehet át egy falon.

A 8.2. ábrán a hangár összetett modellje jelenik meg, ami már sokkal több poligonból épül fel, és ennek használatával már nem jelentkezik az a hiba, hogy az objektumokat a terep fölé helyezi el az algoritmus.

2.3. Objektum lerakási módszerek

Az objektumok lerakásánál fontos volt, hogy változatosan tudjuk elhelyezni azokat a terepen, valamint a valóságnak megfelelően legyenek elhelyezve egymáshoz képest, így több lerakási módszert is létrehoznunk a szimulációban. Ezeket a módszereket egyszerre használhatjuk egy konfigurációban, tehát a pálya egyik pontján akár egyenként is lerakhatjuk az objektumokat, míg máshol például sorban.



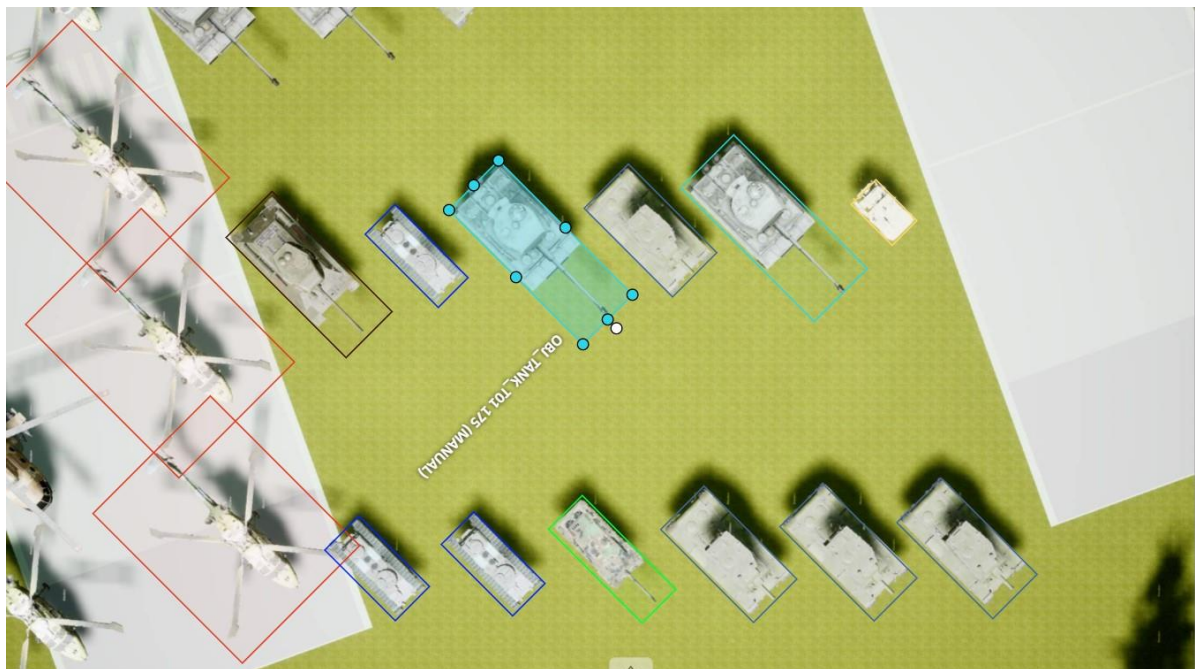
9. ábra Objektum lerakási módszerek

Az 9. ábra mutatja, hogy jelenleg a szimulációban milyen lerakási módszerek vannak implementálva az objektumok elhelyezésére.

Az első lerakási forma az adott objektum adott helyre való elhelyezése. Itt egy konkrét koordinátára illeszti a megadott objektumot, miközben meg lehet adni a függőleges tengely szerinti elfordulás mértékét is. Amikor nem egyetlen objektumot szeretnénk elhelyezni,

lehetőség van objektumok véletlenszerű kiválasztására is egy-egy kategóriából. Ezt a lerakási módot akkor alkalmazzuk, amikor csak a lerakási helynek van jelentősége.

Másik megoldás, ha az objektumok egy sorban helyezük le. Ennek a módnak a használatához a sor kezdeti pontjának koordinátáját, az objektumok közötti távolságot, az elemek z tengely szerinti elfordulását kell megadni, valamint azt, hogy hány elemből álljon a sor. Utolsó paraméterként az objektum kategóriát kapja meg a szoftver, aminek az elemeiből véletlenszerűen választja ki a sorban lévő elemeket. Ha ennek a paraméternek a helyén egy objektum nevét adjuk meg, akkor a sor csak azt az objektumot fogja tartalmazni. Abban az esetben, ha az objektumok darabszámához mínusz egyet adtunk meg, az adott kategóriából lerak minden elemből egy darabot a kategória fájlba megadott sorrend szerint.



10. ábra Objektum lerakás sorban

A 10. ábrán párhuzamos sorokban elhelyezett harcjárművek jelennek meg, valamint rájuk merőleges sorban elhelyezett helikopterek a lerakási módban rejlő lehetőségek szemléltetésére.

A harmadik lerakási mód az objektumok elhelyezése egy téglalap alapú területen. Itt a téglalap sarkának koordinátáját kell megadni, a téglalap hosszúságát és szélességet, a területen elhelyezkedő objektumok számát, valamint a kategóriát. Ebben a módban is érvényes, hogy a

kategória név helyett meg lehet adni objektum nevet is, valamint az objektumok számának mínusz egyet, hogy minden elemet lerakjon egyszer a kategóriából.



11. ábra Objektum lerakás téglalap alakú területen

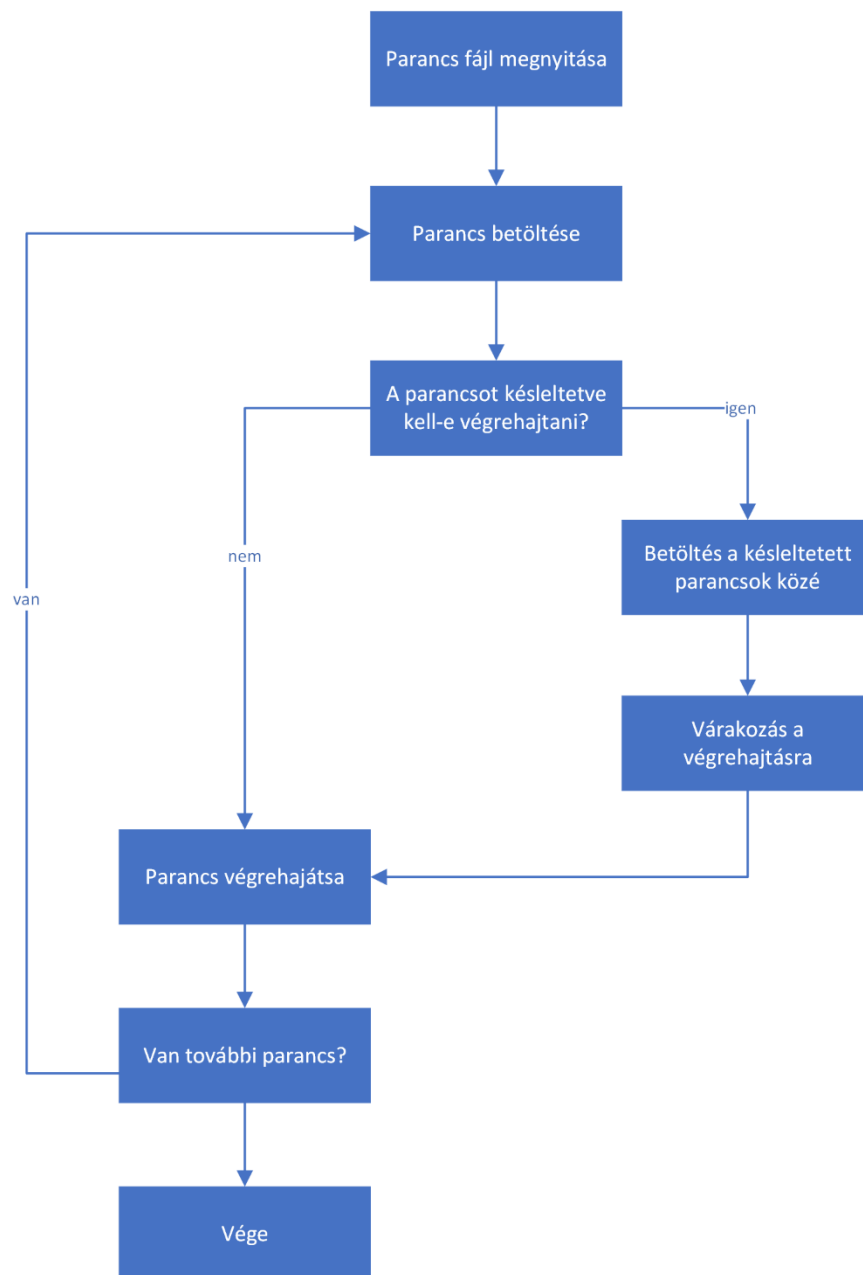
A repülőtéri kifutópályák között egy kis négyzet alakú területre véletlenszerűen elhelyezett harcjárművek láthatók a 11. ábrán.

2.4. Parancsértelmező

A szimulációkból való tanító halmaz generálás során fontos volt, hogy amikor az egyik szimuláció lejárt, a program automatikusan tudja indítani a következőt. Eztért hoztuk létre a szoftverhez saját parancsértelmezőnket, ami különböző előre meghatározott parancsok alapján tud végrehajtani egymás után több szimulációt. A szerveren való tesztelések során bebizonyosodott, hogyha elegendő konfigurációs fájlt készítünk elő a program futtatásához, akkor akár több napig is képes folyamatosan futtatni a szimulációkat és ezáltal nagymennyiségű tanítóhalmaz számára hasznos címkézett képet hoz létre.

Ez egy Python programként van megírva, ami különböző AirSim API függvényeket segítségével vezérli a szimuláció menetét. Az API beépített függvényei között vannak, amiket módosítottunk, mivel nem voltak megfelelőek számunkra, de hoztunk létre új függvényeket is. A parancsértelmező tervezése során az állapot automata struktúrát használtuk, amikor is a

program bizonyos állapotokban lehet, ahonnan különböző folyamatok hatására lép tovább egy következő állapotba.



12. ábra Parancsértelmező folyamatábrája

A 12. ábra a parancsértelmező folyamatábráját mutatja, ahol a program különböző állapotai vannak feltüntetve a folyamat kezdetétől a végéig. A program egy fő ciklusból áll, ami egymás után olvassa be az utasításokat és eldönti, hogy a beolvasott parancsot azonnal

végre kell-e hajtania, vagy bizonyos idővel késleltetve. A késleltetett programok áthelyeződnek egy külön szálra és ott csak akkor hajtódnak végre, mikor letelt a késleltetési idő.

```
1 SET MEASURE_DESCRIPTION "Egy objektumra teszt"
2 LOAD_CONFIG configs/config_9_egy_obj
3 LOAD_LEVEL level_01_basic
4 OBJECT_SURVEY 30 0.80 60 45
5 SET MEASURE_DESCRIPTION "Egy objektumra teszt, RAIN-0.5"
6 SET_WEATHER RAIN 0.5
7 OBJECT_SURVEY 30 0.80 60 45
8 SET_WEATHER RAIN 0
9 SET MEASURE_DESCRIPTION "Egy objektumra teszt, DUST-0.5"
10 SET_WEATHER DUST 0.5
11 OBJECT_SURVEY 30 0.80 60 45
12 SET_WEATHER DUST 0
13 SET MEASURE_DESCRIPTION "Egy objektumra teszt, FOG-0.5"
14 SET_WEATHER FOG 0.5
15 OBJECT_SURVEY 30 0.80 60 45
16 SET_WEATHER FOG 0
17 SET MEASURE_DESCRIPTION "Egy objektumra teszt, SNOW-0.5"
18 SET_WEATHER SNOW 0.5
19 OBJECT_SURVEY 30 0.80 60 45
20 DISABLE_WEATHER
21 SET MEASURE_DESCRIPTION "Egy objektumra teszt, MAPLELEAF-0.5"
22 SET_WEATHER MAPLELEAF 0.5
23 OBJECT_SURVEY 30 0.80 60 45
24 DISABLE_WEATHER
25 SET MEASURE_DESCRIPTION "Auto kategóriára teszt"
26 LOAD_CONFIG configs/config_auto
```

13. ábra A parancsértelmező bemeneti fájlja

A 13. ábrán egy parancsértelmező fájl példa programja látszik, amit szimulációkban használtunk mérések generálására. A fájlban látható, hogy az elején ugyanazt az objektum lerakási konfigurációt járja be a drón többször, csak más időjárási viszonyok között.

2.4.1. Utasítások

Konfigurációs fájlok betöltése

A drón kezdeti pozícióját és az objektumok elhelyezését konfigurációs fájlokban tároljuk, ez a parancs betölti a megadott helyről a konfigurációs fájlokat. Abban az esetben, ha GPS koordinátában vannak megadva, átkonvertálja az adatokat az Unreal Engine saját koordináta-rendszerébe. Ezek után meghívja az általunk megírt API függvényt, ami a konfigurációs fájlok alapján lehelyezi az objektumokat. Abban az esetben, ha már vannak lehelyezve objektumok a terepen, törli azokat és azután tölti be az új elrendezést. A drónt is áthelyezi az utasítás az általunk

megadott kezdeti pontba. Ezt az utasítást a LOAD_CONFIG paranccsal használjuk, ahol még azt kell megadni, hogy melyik mappából szeretnénk betölteni a konfigurációs fájlokat.

Terep betöltése

Amikor egy szimuláció lejárt és a következő bejárást már egy másik pályán szeretnénk végrehajtani, akkor hívjuk meg ezt a függvényt a LOAD_LEVEL paranccsal. Ennek a függvénynek csak egy paramétere van, a betöltendő pálya neve.

Időjárás beállítása

Az időjárás változásával nagyban változnak a látási viszonyok is. Esős vagy ködös időben sokkal nehezebb dolga van az algoritmusnak a képfelismerésben. Ezt kiküszöbölni úgy lehetséges, ha a tanító halmazba mi is behozunk ilyen időjárási körülmények során készült képeket. Ebben az esetben nagyban megmutatkozik a szimuláció előnye a valós drónnal szemben, mivel ilyen időjárási körülmények között nem biztonságos drónt reptetni és a drónok többsége nem is képes erre. A másik probléma, hogy specifikusan csak abban az esetben lehetne ilyen képeket készíteni a valóságban, amikor az időjárás is megfelelő.

Az időjárás beállításához a SET_WEATHER parancsot használjuk, aminek két paramétere van az első az időjárás típusa, ami lehet: RAIN, DUST, FOG és SNOW. A második paraméter meg egy nulla és egy közötti szám, ami az erősségét adja meg.

Látószög beállítása

A drón kamera látószögének beállítására a SET_FOV parancsot használjuk, ami paraméterként Euler szögben kapja meg a kívánt értéket. A látószög változásával változik a terület mérete, amit a drón egyszerre belát, tehát nagyobb látószögnél nagyobb területet is lát be. Itt arra kell oda figyelni, hogy túl nagy értékeknél a kép torzítása is nagy lesz.

Leírás hozzáadás a mérésekhez

A SET MEASURE_DESCRIPTION paranccsal megjegyzéseket lehet fűzni egy adott méréshez. A szimulációk végeztével a konfigurációs adatokat kimentjük egy táblázatba. Ezeknek a megjegyzéseknek egy rövid leírást kell tartalmazniuk a szimuláció fontosabb

paramétereiről. A leírások hozzáadása azért fontos, hogy gyorsan tudjunk keresni a szimulációk között amikor egy tanításhoz, vagy teszteléshez válogatunk képeket.

Objektumok bejárása

Az OBJECT_SURVEY parancs hatására a drón körbejárja a pályán elhelyezett objektumokat. Ennél a parancsnál meg lehet adni a távolságot, amiről az objektumokat rögzíteni fogja, és egy listát, hogy milyen szögekből készítsen képeket. A CVAT címkéző programba fontos volt megoldani, hogy a mérések automatikusan feltöltődjenek az elkészülésük után, ezt a parancs után az UPLOAD kulcsszóval valósítottuk meg, ami után még meg kell adni a projekt nevét, ahová fel szeretnénk tölteni az adott mérést. A mérések feltöltése úgy valósul meg a gyakorlatban, hogy amikor a szimuláció lejárt, egy külön szálon elindítja a program a feltöltést, és közben a fő szálon folytatja a szimulációk futtatását. Azért volt fontos, hogy a képek feltöltése közben már fusson egy következő szimuláció, mert így kevesebb idő alatt több szimuláció futtatható le.

Terület bejárása

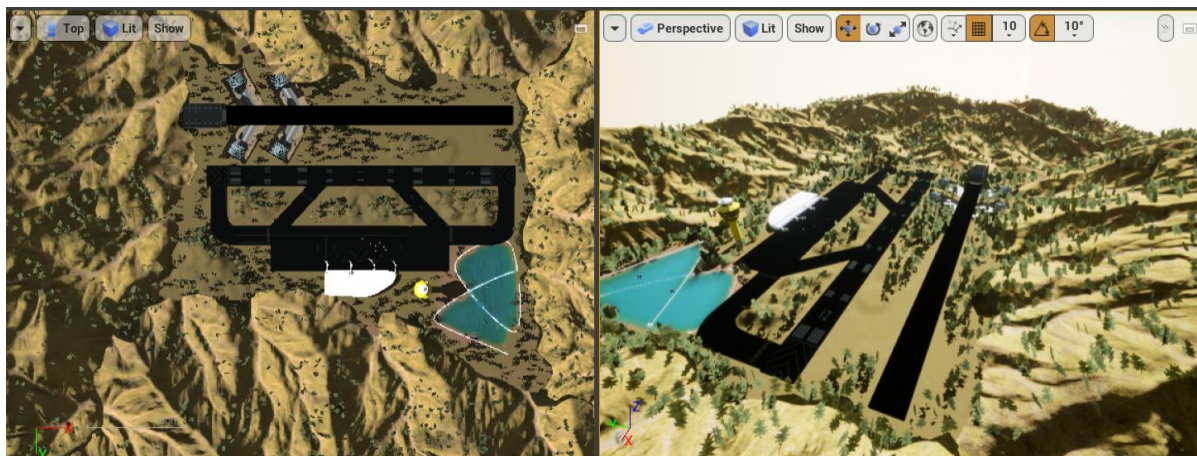
A SURVEILLANCE parancssal hívható meg ez a függvény. Ez a bejárás abban különbözik az objektum bejárástól, hogy itt egy négyszög alakú területet repül be a drón, míg az objektum bejárásnál az összes objektumot körbeszkenneli egyesével. Az UPLOAD paraméter ennél a parancsnál is hasonlóan működik, mint az objektum bejárásnál. Ennél a megoldásnál megadatható paraméter a szkennelendő terület nagysága, kezdőpontja, a drón repülési magassága, valamint sebessége. További két paraméterrel meg lehet adni, azt hogy a drón által készített képek mennyire fedjék át egymást egy nullától egyig való skálán, és azt hogy a címkék mennyire lóghatnak ki egy adott képből.

Parancsok késleltetett futtatása

A késleltetett futtatást a DELAY parancssal lehet használni, ahol meg kell adni a késleltetés mértékét másodpercben, valamint utána a parancsot, amit késleltetve kívánunk futtatni. A késleltetésre azért volt szükség, hogy miközben a drón egy terepet bejár közben dinamikusan lehessen változtatni például az időjárási viszonyokat, vagy más paramétereket. Ez a fajta időjárás változtatás a valósághoz is közelebb állna mivel, az az idő múlásával tényleg változhat, például az eső, vagy havazás intenzitása, vagy a köd mértéke.

3. Terepek

A szint a meghatározás szerint magába foglalja az úgynevezett virtuális környezetet, valamint minden objektumot, amelyet a virtuális kamera érzékel, vagy van fizikai kiterjedése.



14. ábra Saját fejlesztésű terep Unreal Engine környezetben

Az Unreal Engine-ben a szint statikus hálók, térfogatok, fények, tervrajzok és egyébek gyűjteményéből áll, amelyek együtt kerülnek alkalmazásra, hogy megteremtsék a kívánt valósághű hatást, mintha az egy teljesen valóságos környezet lenne. Az Unreal Engine szintek mérete a hatalmas terepalapú világoktól a nagyon kis szintekig terjedhet. Egy saját építésű terep látható az Unreal Engine környezetben a 14. ábrán.

3.1. Terepek kialakítása

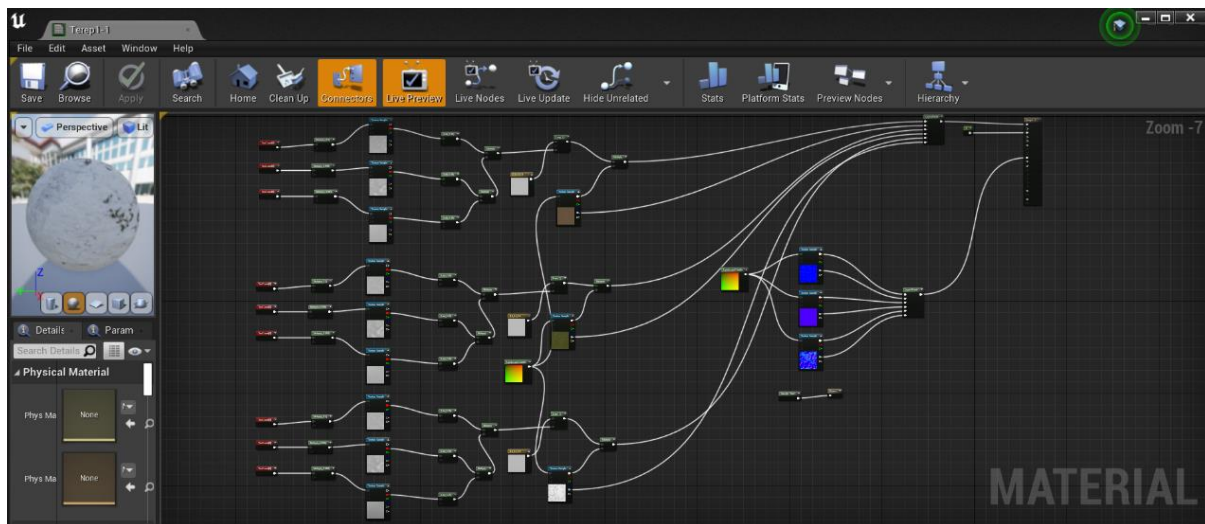
A szintek létrehozása, mentése és megnyitása során néhány szempontot figyelembe kell venni, melyek a következők: játékszint struktúrája, fényezés és textúrázás, Blueprint és szkriptek, optimalizáció, kollíziós dobozok, hő térképek és pontfények, melyek mind nagyon fontosok egy terep kialakításakor. Ugyanakkor ide sorolható a megfelelő fájlnev és elérés, valamint a verziókövetés is.

A szimulációs környezet kialakítása közben nagyon fontosnak találtuk a megfelelő terep diverzifikációt, hogy minél színesebbé tegyük a tanítóhalmaz generálását. Ezért is használtunk több kialakítási módszert és szoftvert a terepek létrehozásakor. A szimulátorunkban ugyanis

többféle teret is ki tudunk alakítani, mint a teljes egészében virtuális alapon létrehozott terek, de drónfelvételekből is tudunk terepet visszaépíteni és azon méréseket végezni.

3.1.1. Terek anyaga

Az anyag egy olyan 3D-s számítógépes grafikában használt eszköz, amely a jelenetek vizuális megjelenését szabályozza. Másként megfogalmazva, az anyag hasonlíthat egy „festékhez”, amelyet egy tárgy felületén alkalmaznak. Azonban még ez a hasonlat sem fedi a teljes igazságot, mivel az anyag szó szerint azt definiálja, hogy a tárgy felülete hogyan viselkedik a fény hatására. Az anyag meghatározhatja a tárgy színét, fényességét, átláthatóságát, és számos egyéb tulajdonságát.



75. ábra Unreal Engine szintek anyagának hálóját

Technikailag megfogalmazva, amikor a jelenetben lévő fény eléri egy tárgy felületét, az anyagot arra használják, hogy kiszámítsák, milyen kölcsönhatásba lép a fény a felülettel. Ezeket a számításokat olyan bemeneti adatok alapján végzik, amelyek a textúrákból származnak, matematikai kifejezésekből állnak, valamint az anyag tulajdonságaitól függenek. Az anyag magában foglalja azokat a paramétereket és beállításokat, amelyek meghatározzák, hogyan viselkedik a felület a fény hatására, például milyen színű, mennyire fényes vagy matt, és hogy átlátszik-e vagy sem. Az anyagok segítségével a 3D-s grafikus szoftverek képesek valósághűen megjeleníteni különböző anyagokat és felületeket a számítógépes jelenetekben.

Az általunk létrehozott terek anyaga nagyon fontos mivel nagy részben ez határozza meg, hogy mennyire valósághű egy terep. A külső szoftverekkel létrehozott tereknek van saját anyaguk, azoknak a tereknek azonban amelyeket mi hozunk létre Unreal Engineben,

nincsen. Ezért is nagy hangsúlyt fektettünk, mind az objektumok, mint pedig a tereptárgyak anyagára. A 15. ábrán megfigyelhető egy általunk létrehozott terep anyagrendszere. Ez az anyagrendszer nem csak magát a textúrát foglalja magába, hanem ugyanitt meg kell határozni egy anyagnak a különböző fényhatások közepette való viselkedését is. A 15. ábrán látható példában egy terep estén három anyagszintet hoztunk létre, egy földeset, egy füves zöld anyagot, meg egy fehér havast. Ez azért szükséges, hogy egy terepen több évszakot is tudjunk szimulálni, valamint, hogy egy terepen, ha vannak magas havas hegyek, szántóföldek meg zöld füves területek akkor arra megfelelő anyagokat tudjunk elhelyezni.

3.2. Terepek kialakítására használt szoftverek

Az általunk kifejlesztett szimulációs környezetben több szoftvert is használtunk a terepek kialakítására, hogy minél jobban diverzifikáljuk a terepek karakterét. A terepek kialakítására használt szoftverek:

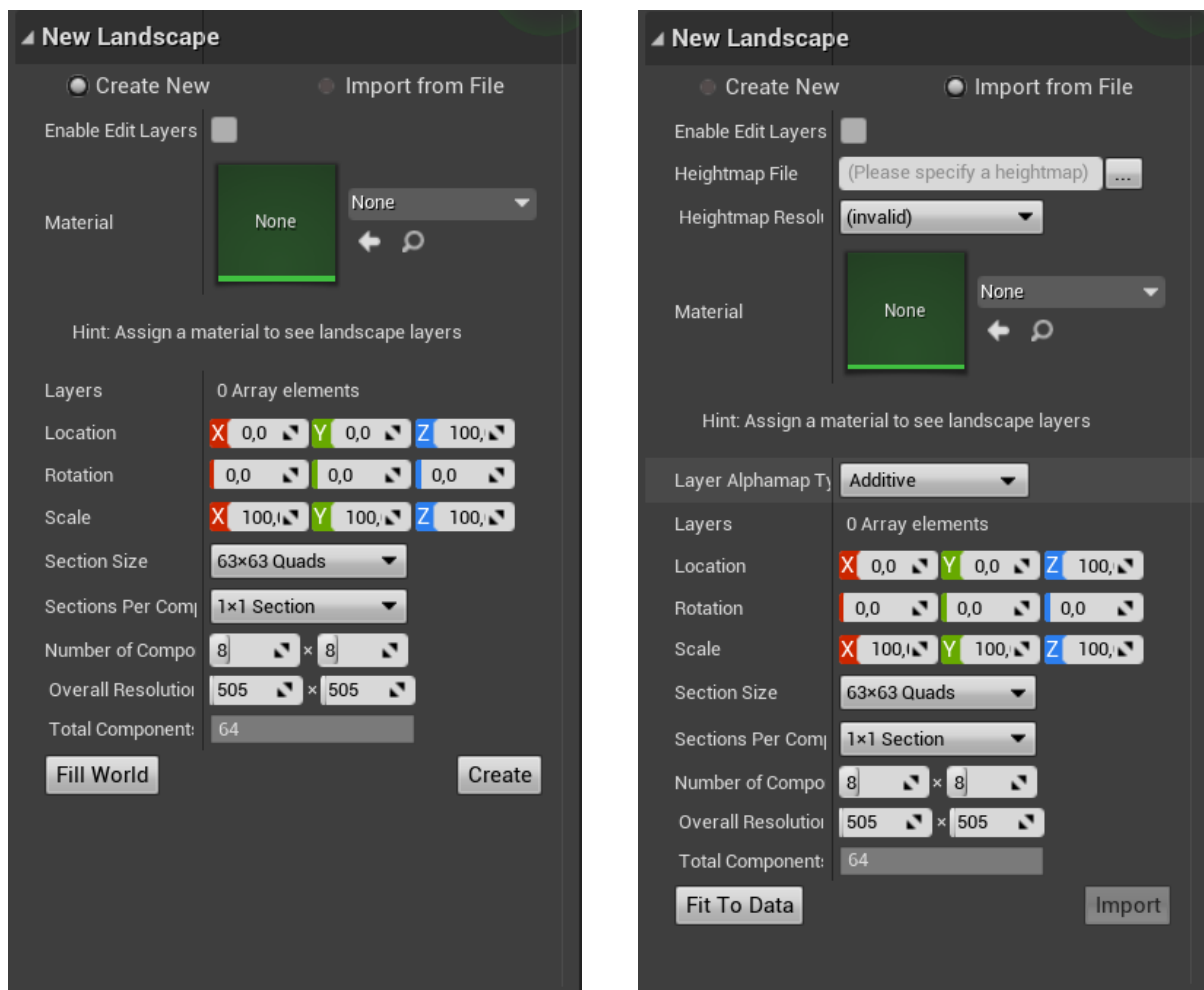
3.2.1. Unreal Engine színtszerkesztő

A terepek létrehozása, menedzselése és szerkesztése során ez a legfontosabb szoftver, ugyanis itt nyerik el a terepek a végleges formájukat. E szerkesztő segítségével tudunk terepeket importálni a szimulátorba más külső szoftverekből, de ugyanakkor létre is tudunk hozni ennek a szoftvernek a használatával új terepet.

A 16. ábrán látni lehet az Unreal Engine környezetben a terep létrehozására, valamint importálására használatos menüt. Ezekben a menükben nagyon hasonló beállítások láthatóak, ami azért van mert, ha a domborzatot egy új terepen hozunk létre az Unreal Engineben, akkor úgy kell megadni a méreteket, mintha egy terep domborzatát egy külső terepszerkezetből importálnánk. A különbség az, hogy míg egy új létrehozott terep kezdetben teljesen sík, addig egy importált terepnek már megvan a domborzata.

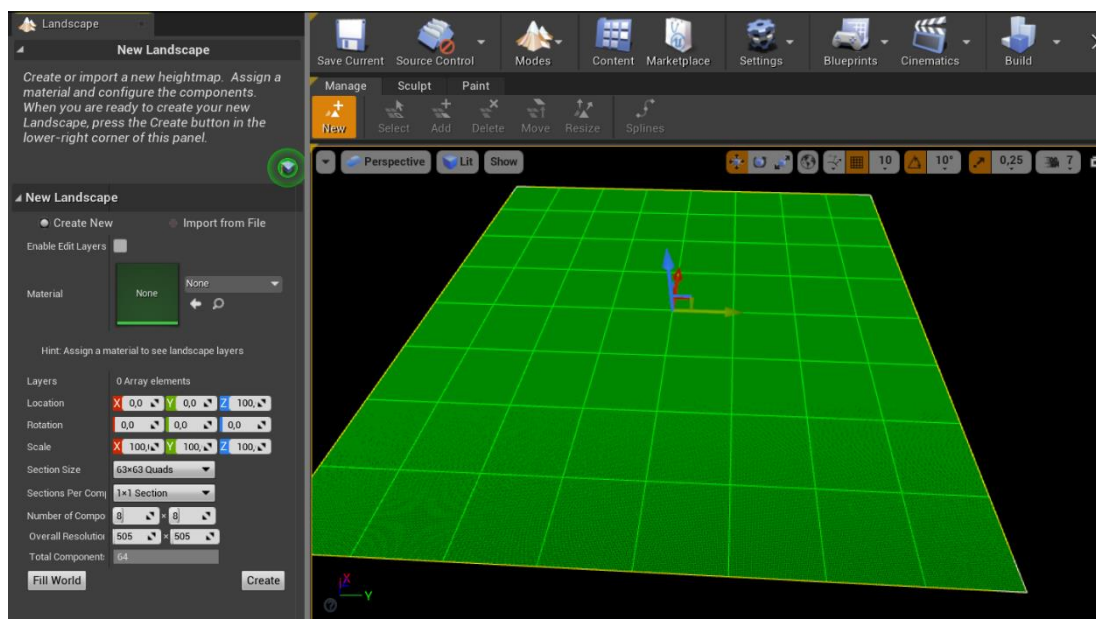
A Terepszerkesztő az Unreal Editor alapvető szintű létrehozási funkcióját biztosítja. Használni lehet szintek létrehozására, megtekintésére és módosítására. A teret (szintet) főként a benne elhelyezett objektumok átalakításával és szerkesztésével módosíthatja. Az Unreal Editorban azokat a területeket, amelyekben létrehozuk a játékélményt, általában szintnek nevezik. A szintet úgy kell elképzelni, mint egy 3D-s környezetet, amelyben tárgyak és geometriák sorozatát helyezzük el annak érdekében, hogy meghatározzuk a játékosok által

tapasztalt világot. Minden tárgyat, amelyet a virtuális világban helyezünk el, legyen az fény, háló vagy karakter, Actornak számít. Technikailag az Actor egy programozási osztály, amelyet az Unreal Engine-en belül használnak olyan objektumok meghatározására, amelyek 3D-s pozícióval, forgással és léptékadatokkal rendelkeznek. Minden olyan tárgyat, amit el lehet helyezni a pályán, az Actornak számít. [16] A 17. ábrán azt lehet látni, ahogy létrehozunk egy új, teljesen üres terepet az Unreal Engine szint szerkesztőben. Ugyanitt be tudjuk állítani a terep méreteit, valamint ezen a ponton hozzá tudjuk adni az új terephez a textúrát is.



16. ábra új terep létrehozása és importálása

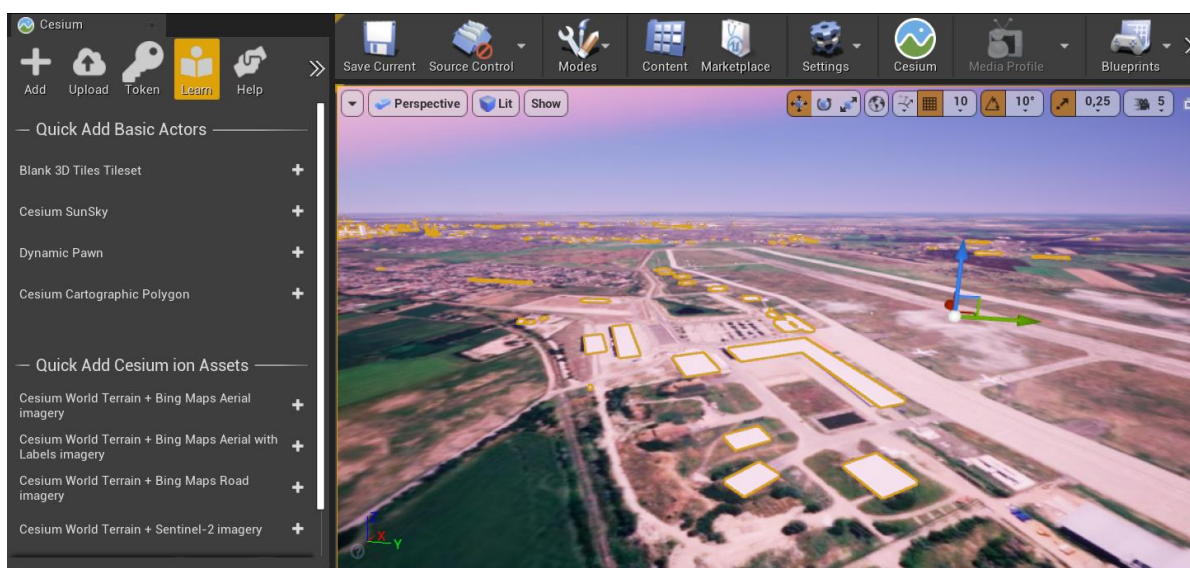
A szintek (terek) létrehozása úgy kezdődik, hogy elemeket helyezünk el egy térképen az Unreal Editorban. Ezek lehetnek világgeometria, dekorációk, statikus hálók, lámpák, játékosindítások, fegyverek, vagy járművek. Azt, hogy mely elemek mikor kerülnek hozzáadásra, általában a szinttervező csapat által használt munkafolyamat határozza meg.



17. ábra Unreal Engine szerkesztőben új terep létrehozása

3.2.2. Cesium

A Cesium története 2011-ben kezdődött, amikor az Analytical Graphics, Inc. (jelenleg Ansys) repülőgép-szoftvercég fejlesztői csapata egy olyan alkalmazás létrehozását tűzte ki célul, amely az űrben lévő objektumok megjelenítésére szolgál. A 18. ábrán látható egy terület, amit Cesium szoftver segítségével hoztunk létre. Valamint ugyanitt látni lehet a Cesium különböző hozzáadandó eszközeit, mint a „Blank 3D Tiles Tileset”, ami a terepen lévő 3D objektumok, mint például különböző épületek hozzáadását teszi lehetővé.



18. ábra Cesium Plugin

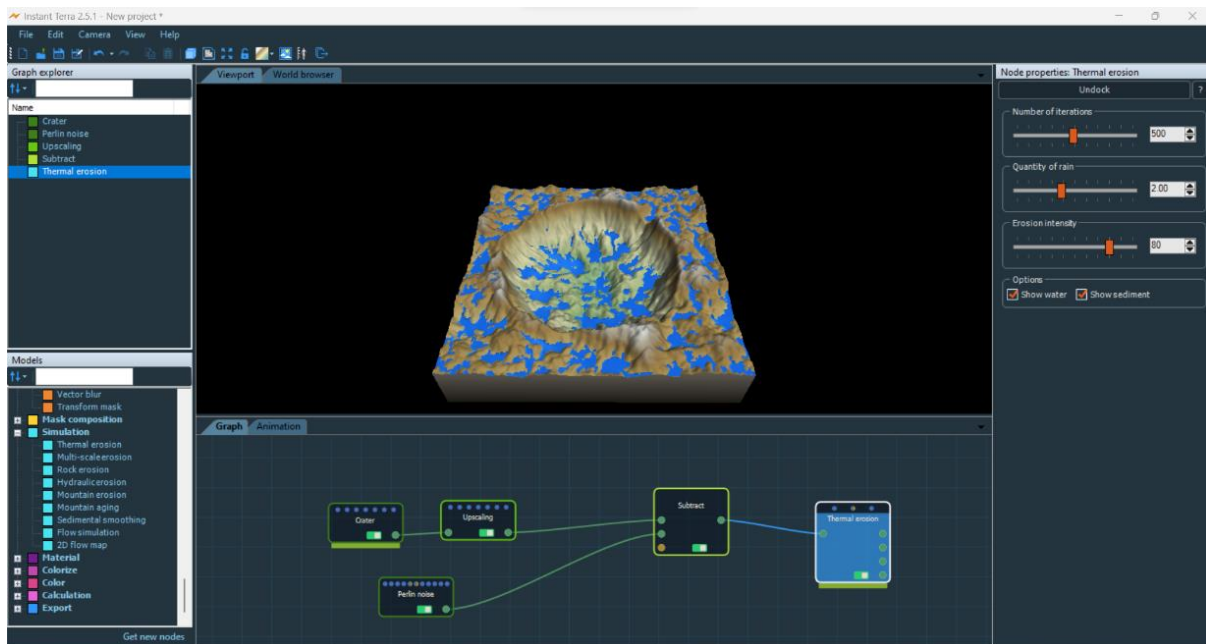
A Patrick Cozzi által vezetett projekt a világ legpontosabb, legteljesebb és legdinamikusabb virtuális földgömbjét készítette el. Az atomórákat híresen pontossá tevő elem után Cesiumnak keresztelték, és 2012-ben nyílt forráskódúként adták ki. Ugyanakkor a 3D-s adatgyűjtés világszerte elterjedt, ami olyan szoftverek iránti igényt váltott ki, melyek kiaknázzhatják a benne rejlő lehetőségeket. Ahogy az iparágak elkezdtek gyűjteni a 3D-s helyadatokat számos felhasználási esetre, Cozzi és csapata meglátta a lehetőséget, hogy a Cesiumot az űrhajózáson túlra is kiterjessze. A Cesium 2019-ben független vállalként alakult ki, és ma egy interoperábilis térinformatikai ökoszisztéma alapját képezi. A nyílt forráskódú és a kereskedelmi forgalomban kapható szoftverek keverékét kínáló Cesium platform az eszközök teljes készletét kínálja bármilyen 3D térinformatikai alkalmazás létrehozásához. [17]

A mi rendszerünkben is nagyon hasznos szoftvernek bizonyult a Cesium, ugyanis e szoftver segítségével egyszerűen létre tudunk hozni egy 3D-s szintet (terepet), amely tartalmazza a teljes bolygót, és akár olyan valósághű területeken is tudunk mérést végezni, ahol a repülés a valóságban nem lehetséges.

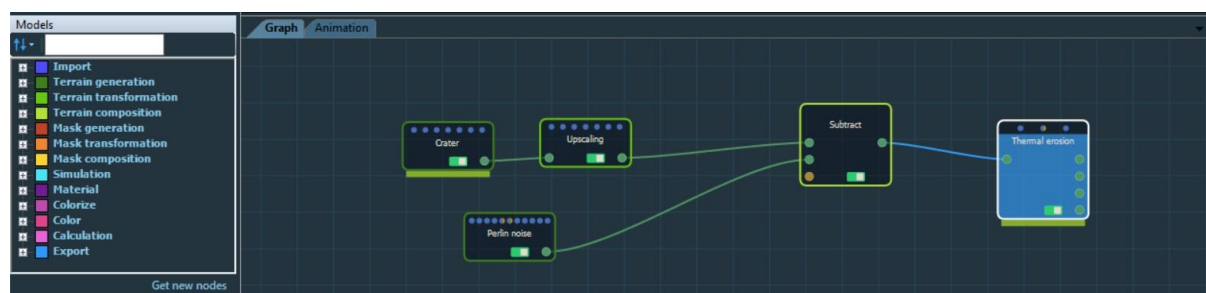
3.2.3. Instant Terra

Ma a valósághű terepek létrehozása hosszú folyamat, amely jelentős számítási időt és számos iterációt igényel. Az innovatív technológiákon alapuló Instant Terra segítség abban, hogy terepeket gyorsan és zökkenőmentesen hozzunk létre, mérettől függetlenül. Ez az ellentéte olyan szoftvereknek, mint a Houdini, ami például alacsony felbontással dolgozik, hogy gyorsabban dolgozzon, és talán nem éri el a kívánt eredményt a végső számítás elvégzésekor. [18]

Az Instant Terra segítségével közvetlenül a végső felbontással dolgozhatunk, ez látható a 19. ábrán. Bármilyen legyen a terep mérete, az Instant Terrain mindig megmutatja a végső terepet annak minden részletével együtt. Eljárási terepalkotás: A különböző csomópontokból álló grafikon létrehozásával könnyedén megépíthetjük a terepet. Minden csomópont egy műveletnek felel meg több terepen. A csomópontokat linkek kötik össze. Bármikor visszatérhetünk bármely csomópontához, módosíthatunk egy paramétert, és azonnal láthatjuk a végeredményre gyakorolt hatást. Úgy tehetjük ezt meg, hogy mindig abba a csomópontba lássuk a terepet, amelyik fázisban éppen vagyunk. A 20. ábrán is szemléltetve vannak az egyes csomópontok, melyeket külön-külön tudunk vizsgálni.



19. ábra Instant Terra terepszerkesztő szoftver



20. ábra Instant Terra terepszerkesztő programozó blokkok

3.2.4. Open Drone Map (ODM)

Az Open Drone Map egy nyílt forráskódú fotogrammetriai eszközkészlet, amellyel a légi felvételeket (általában drónról készítetteket) térképekké és 3D modellekké alakíthatjuk [19].

A szoftvert a GitHubon szabadon hozzáférhető módon tárolják és terjesztik. Ennek a szoftvernek a segítségével alakítottunk ki terepeket saját mérések alapján. Ennél a kialakítási módszernél nem elég csak maga a szoftver, szükség van egy 3D modellezőre is, amely célra mi a Blendert használtuk. Azért volt rá szükség, hogy a modelleket átalakítsuk az FBX¹³ formátumba, valamint megfelelően tudjuk méretezni, amit később könnyedén importálhattunk

¹³ Filmbox

is az Unreal Engine-be, és kialakíthattuk a területeket a valós mérések alapján. Egy valós mérés alapján visszaépített terület látható a 21. ábrán is az ODM környezetben, itt fel tudjuk tüntetni a kamerák pozícióját is ebből látjuk, hogy a terep mely része milyen szögből volt rögzítve.



21. ábra ODM terep visszaépítő szoftver

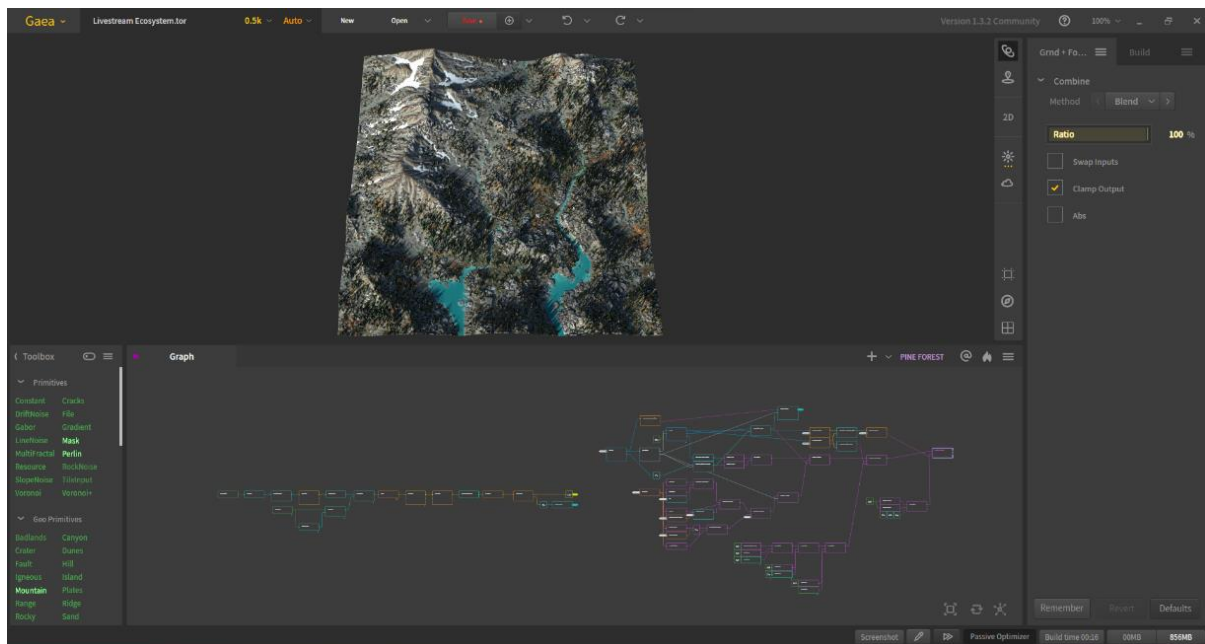
3.2.5. Gaea

A Gaea az iparág vezető tereptervező alkalmazása VFX¹⁴-ek létrehozásához, játékokhoz és virtuális gyártáshoz. [20] A Gaea szoftvert a QuadSpinner LLC fejlesztette ki. A szoftver célja a valósághű, részletgazdag tájtervezés és a térképek készítése, különös tekintettel a digitális játékok és a filmipar igényeire. A Gaea egy erőteljes, Node-alapú tervezési eszköz, amely segítségével a felhasználók magas felbontású domborzati térképeket, textúrákat és egyéb földrajzi elemeket hozhatnak létre.

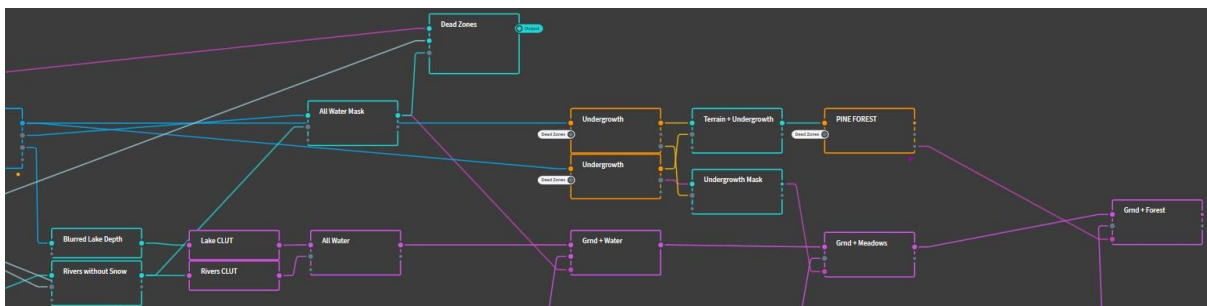
A Gaea számos funkciót és eszközt kínál a tájtervezéshez és a domborzati elemek létrehozásához, például a területi csúcsok és völgyek kialakítását, az erózió szimulációját, valamint különböző domborzati rétegek kezelését, például talaj, növényzet és víz. Ezt szemlélteti a 22. ábra is. A 23. ábrán láthatjuk a különböző blokkok használatát, természetesen ebben az esetben is, mint az Instant Terra szoftver esetén, minden blokkban meg lehet vizsgálni

¹⁴ Visual Effects – Vizuális effektek

a terep készültségi állapotát. A szoftver kompatibilis más tervező és 3D-s szoftverekkel, így könnyen integrálható a meglévő munkafolyamatokba. Ennek következtében mi is könnyedén tudtuk használni ezt a szoftvert projektünkben.



22. ábra Gaea terepszerkesztő szoftver

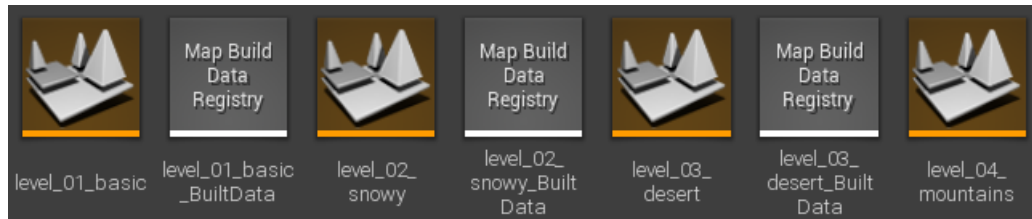


23. ábra Gaea terepszerkesztő programozó blokkok

3.3. Terepek menedzselése

A terepek menedzselését a projektünkben úgy oldottuk meg, hogy az elnevezésük alapján rendszereztük őket. Az elnevezésük a következő: „level_01_basic”. Itt a „level” a teret jelöli, a „01” a terepnek a sorszámát, valamint a „basic” a terep jellegét, ami lehet „snowy”, „desert”, „mountains”, vagy bármilyen elnevezés, amely leírja a terep jellemzőit, amint azt a 24. ábrán is látni lehet. Ezen az ábrán láthatjuk magát a terepet, valamint a hozzájuk tartozó

„BuildData” fájlt is, ami a terepen lévő fények beállításait tartalmazza. Ez a fájl a terep első betöltésekor generálódik.



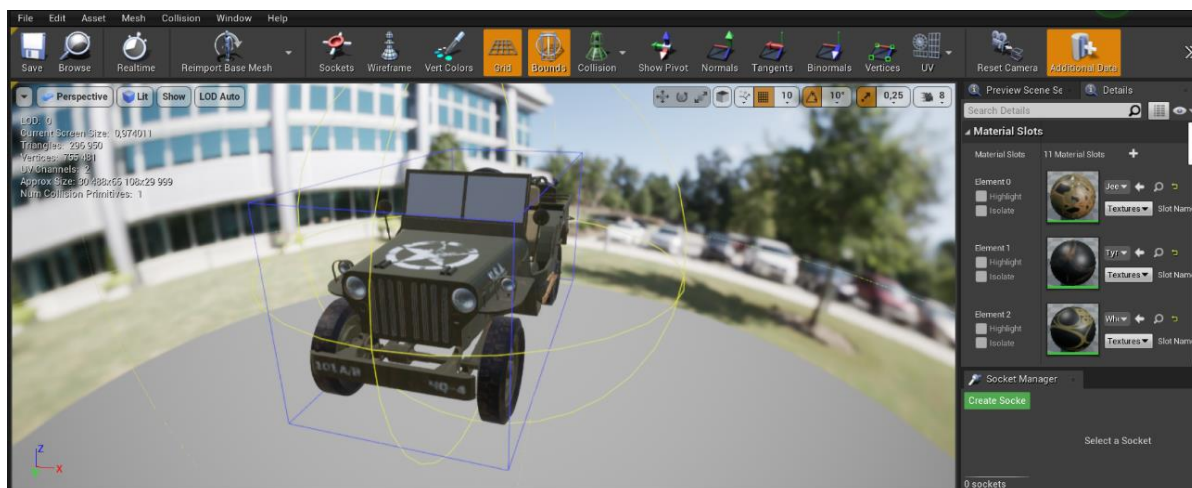
24. ábra terepek elnevezése

A terepek menedzselése és tárolása egy nagyon fontos szempont, amit a továbbiakban is fejleszteni szeretnénk adatbázis alkalmazásával, mivel adatbázisban sokkal jobban átláthatók lesznek a területek, és rendszerezhetjük őket tartalom, méret vagy kialakítás szerint.

4. Objektumok menedzselése, bejárások

4.1. Objektumok projektbe való integrálása

A projekt során több különböző objektumot használtunk, melyeket két nagyobb kategóriába sorolhatunk. Terepek létrehozásához tartozó objektumok, mint fák, épületek, aljnövényzet és különböző tárgyak, valamint a tanítóhalmaz generálásához szükséges objektumok, mint például földi, vagy légi járművek. Az objektumokat több módon próbáltuk minél színesebbé tenni és minél több objektummal bővíteni szimulációs környezetünket. Próbálkoztunk az online elérhető modellek letöltésével, valamint saját felvételekből is építettünk vissza 3D-ben objektumokat. A 25. ábrán látható egy objektum, annak textúrái, valamint a „Boundary Boxa”, ami az objektum méreteit szemlélteti. Ez a box mindig az objektum egymástól legtávolabb levő pontjain található.



25. ábra Egy objektum az Unreal Engine környezetben

Az interneten bárki számára elérhető modellekkel az egyik nagy probléma, hogy nincsenek egységes formátumban feltöltve, ezért letöltés után konvertálnunk kell azokat, mielőtt importálni tudnánk a célkörnyezetbe. Konvertálásához a Blendert használtuk, mivel az egyik legelterjedtebb 3D modellezésre alkalmas szoftver, így rengeteg 3D modell formátumot támogat. Az általunk használt formátum az importáláshoz az FBX volt.

Az FBX formátum előnye, hogy a Statikus Mesh, Csontváz Mesh, animáció és morf célpontok egyetlen fájlformátumban vannak kódolva, más megoldásokkal ellentétben, amelyeket támogat az Unreal Engine importálásra. Több eszköz/tartalom helyezhető el

egyetlen fájlban. Több LOD¹⁵ és Morf/Blendshape importálása megoldható egyetlen importművelettel, például anyagok és textúrák importálása és mesh-ekre alkalmazása. Ezért is használtuk ezt a formátumot, így sokkal hatékonyabban tudtuk importálni az objektumokat.

4.1.1. Objektumok konvertálása

Az előzőleg előkészített objektumokat, melyek letöltésre kerültek, vagy általunk kerültek visszaépítésre, rendszereztük típus alapján.



26. ábra Blender szoftverbe importált objektum

Ezt követően az objektumok egyenként importálásra, majd megvizsgálásra kerültek a Blender 3D modellszerkesztő szoftverében, hogy minden textúra helyesen illeszkedik-e a felületre (26. ábra). Ezt követően az átvizsgált objektumokat exportáltuk a nekünk megfelelő FBX formátumba.

4.1.2. Objektumok importálása és menedzselése

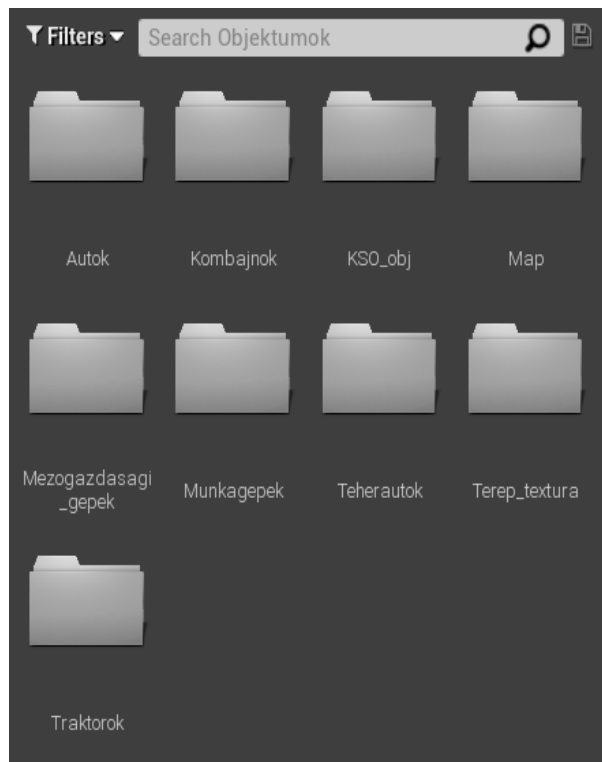
Az objektumok importálása csak egy alapos ellenőrzést követően történhetett, amikor minden objektumról biztosan tudtuk, hogy minden textúra megvan hozzá és az helyesen található-e

¹⁵ Level of Detail – Részletességi szint

meg az objektumon. Az importálás első lépéseként létre kell hozni az objektumnak egy almappát a kategóriájának megfelelő mappában (27.2. ábra) az objektum nevével. Ezt követően az objektum importálásra kerül az új mappába a textúrákkal együtt. Az importálás beállításai láthatók a 27.1. ábrán. Itt meg kell határozni az objektum elfordulását, skáláját, valamint, ha az objektum több különálló részből áll, akkor azt egyben vagy külön-külön objektumként importáljuk.



27.1. ábra Objektumok importálásának beállításai



27.2. ábra Objektumok kategóriái

A szimulációs környezetbe importált objektumok menedzselése a következőképpen történik. Az objektumok különböző kategóriákra vannak felosztva. Esetünkben két fő kategória található meg, a kutatásspecifikus és a nem kutatásspecifikus objektumok. Ezt követően ezek a kategóriák több kisebb kategóriára vannak felosztva. Nem kutatásspecifikus objektumoknál: autók, traktorok, mezőgazdasági gépek, teherautók. Kutatásspecifikus objektumoknál: harcjárművek, repülő, helikopterek. Ezek az objektumok kategóriánként külön könyvtárakba vannak rendszerezve. Minden objektumnak specifikus neve van, mint például „obj_auto_Willi-Jeep”, amelynek nevében az „obj” azt jelzi, hogy objektum az „auto”. Az objektumainkról vezetünk egy táblázatot is (28. ábra), amelyben megtalálható minden objektum esetén egy sorszám, az objektum csoportja, ami lehet „kso” azaz kutatásspecifikus objektum, valamint lehet „nonkso”, ami nem kutatás specifikus objektumot jelöl. Ugyanitt

található a típus, amelyben az objektum típusa van meghatározva, az objektum neve, valamint minden objektum mérete. Unreal Engineben ez az importálás utáni méret, amikor még az objektum nincsen skálázva. Ebben a táblázatban található az objektumok valós mérete is, ezek a méretek azért szükségesek, hogy minden objektumnak be lehessen állítani a megfelelő skálázását.

Sorszám	Csoport	Típus	Név	H	Sz	M	H_realis	Sz_realis	M_realis
53	kso	auto-kso	obj_auto_Willy-Jeep	66108	30488	30000	330	150	175
54	kso	tank-kso	obj_tank_T-34-85	61899	22727	21593	815	260	270
55	kso	teherauto-kso	obj_teherauto_AMAC35	43704	13297	20247	4370	250	420

28. ábra Objektumok rendszerezése

A jelenlegi rendszerezés egyszerű és előnyös, gyors kivitelezhetősége miatt, ugyanakkor nem megfelelő, ha nagyon sok objektumot kell rendszerezni. Fejlesztési terveink között szerepel az objektumok adatbázisban való menedzselése, ugyanis több objektumot adatbázis segítségével sokkal egyszerűbb rendszerezni.

4.2. Bejárások

A szimuláció fontos része volt, hogy egy adott területet és az azon elhelyezett objektumokat a virtuális drónunk automatikusan előre meghatározott algoritmusok alapján be tudjon járni. Fontos, hogy a drón a bejárás során információkat is tudjon rögzíteni, amelyeket később el tudunk menteni és felhasználni. Ilyen például a drón és kameráinak helyzete és orientációja, illetve a drón kameráinak képe.

Kétféle bejárési algoritmust készítettünk. Az első algoritmust terület bejárásnak neveztük el, mivel a bejárás során a drón egy téglalap alakú területet térképez fel. A terület bejárás során kizárólag a drón mozgásával járjuk be a területet. A drón nem változtatja főkamerájának orientációját, a kamera, a stabilizátorának köszönhetően, a föld síkjához képest konstans szöget fog bezárni. A második algoritmust objektum bejárásnak neveztük el, melynek során a drón egy előre megadott mozgást végez az objektum körül. A mozgása során előre meghatározott koordinátáknál megáll, majd miután kameráját az objektumra fordítja elmenti kamerájának képét és más számunkra fontos adatokat.

Azért ezt a két bejárást valósítottuk meg, mivel tanítóhalmazunk generálásához ezek voltak a legfontosabbak. Fontos, hogy tanítóhalmazunk tartalmazzon valóságghű méréseket.

Drónokkal általában a magasból szoktunk területeket feltérképezni, ezért a terület bejárás nélkülözhetetlen számunkra. Az objektum bejárás segítségével adott objektumokról több szögből, tudunk méréseket készíteni, így a neuronháló jobban fel fogja ismerni az adott objektumokat a valós mérések során a készített felvételek orientációjától függetlenül.

4.2.1. Drón mozgatása

A szimulált drón mozgatása az AirSim beépülő modul alkotói által szolgáltatott Python API segítségével történik. A drón mozgatására több előre megírt függvény áll rendelkezésünkre, de nekünk egy olyan függvényre van szükségünk, amelynek megadunk egy célkoordinátát, ahova a drón elrepül. Szerencsénkre van egy függvény, ami pont ezt valósítja meg. Az említett függvény a 29. ábrán látható, négy paramétert vár. Az x, y, z paraméterek a kívánt célpozíció koordinátái, a „velocity” paraméternek pedig megadható a drón utazási sebessége m/s mértékegységben. A függvény aszinkron, aminek köszönhetően egy külön folyamatként fut. Ezt a hasznos tulajdonságot kihasználjuk, amikor a drón mozgatása közben lekérjük különböző adatait.

```
client.moveToPositionAsync(x= x, y= y, z= z, velocity= velocity)
```

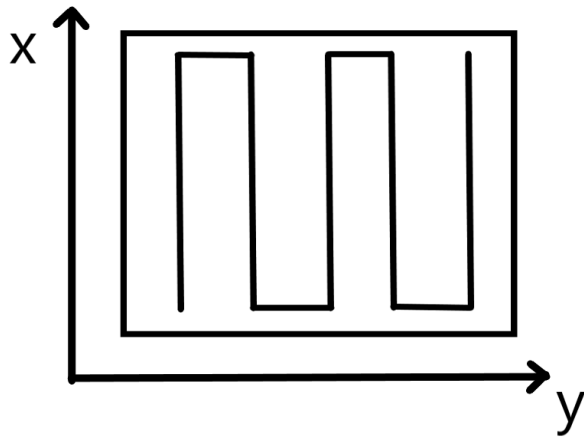
29. ábra Drón mozgatását végző függvény

Ezzel az alap mozgatási függvénnyel minden olyan feladatot meg tudunk oldani, amelynél szükséges a drón elmozgatása egyik pozícióból egy másikba.

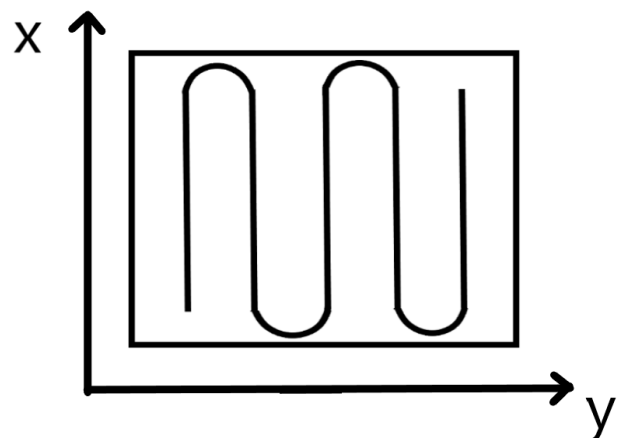
4.2.2. Területbejárás

A területbejárás egyszerű algoritmus alapján vezeti végig a drónunkat egy téglalap alakú területen. A drón a terület bal alsó sarkából indul el, majd a 30.1 ábrán szemléltetett útvonalon halad végig. A mozgást a 30.3. ábrán látható függvénnyel oldjuk meg. Az „előre tekint” paraméter segítségével a 30.2 ábrán látható útvonalhoz hasonlóra alakítjuk az előző útvonalunkat. Így a drónnak nem kell minden sarokpontnál megállnia, hanem lekerekíti az

útvonal végéit és megtartja sebességét. Ennek köszönhetően a drón mozgása folytonos, egyenletesebb lesz. Az útvonal paraméterének az útvonalunk sarokpontjait kell megadnunk.



30.1. ábra Bejárás derékszögű fordulattal



30.2. ábra Bejárás ívelt fordulattal

```
client.moveOnPathAsync(előrettekint , útvonal, sebesség)
```

30.3. ábra MoveOnPathAsync függvény

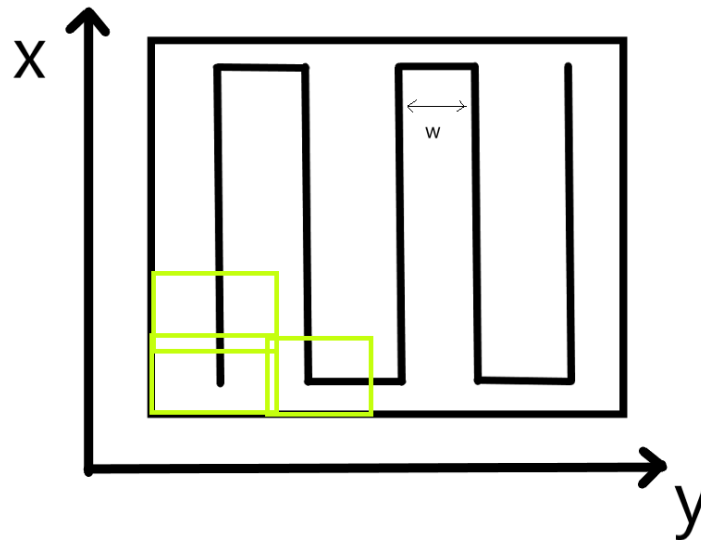
Az útvonal meghatározásánál fontos tudnunk, hogy a pályáink között mekkora távolságra van szükségünk, ami a 31. ábrán „w” betűvel van jelölve. Az ábrán látható zöld téglalapok a bejárás során készített képek vetületeit szemléltetik. Látható, hogy a képeknek kell, hogy legyen átfedése annak érdekében, hogy véletlenül se maradjanak rögzítetlen objektumok a területen. Ezt a távolságot a (3.) képlettel tudjuk kiszámolni. A képletben az „m” változó a drón kezdőpozíciójától mért magasságát jelöli. Az „FOV¹⁶” változó a kamera látómezőjét jelöli. Az „á” változó a képek kívánt átfedését jelöli százalékos értékként megadva. Tehát a lenti képlettel a képek y koordináta szerinti átfedését tudjuk kiszámolni. Ezek után könnyen meg tudjuk határozni az útvonal sarkpontjainak koordinátáit.

$$w = 2 * m * \tan\left(\frac{FOV}{2}\right) * á \quad (3.)$$

Ha a képek középpontjai közötti távolságot az x koordináta szerint szeretnénk kiszámolni, akkor a fent említett képlet eredményét meg kell szorozni a kamera képarányának

¹⁶ Field of View – Látómező

fordítottjával. Ez az érték nem befolyásolja a sarokpontok koordinátáit. A kiszámolt értéket, ha elosztjuk a drón sebességével, megkapjuk az időmennyiséget, amelynek két kép készítése között kell eltelnie ahhoz, hogy a képek között megfelelő legyen az átfedés.



31. ábra Átfedés a képek között

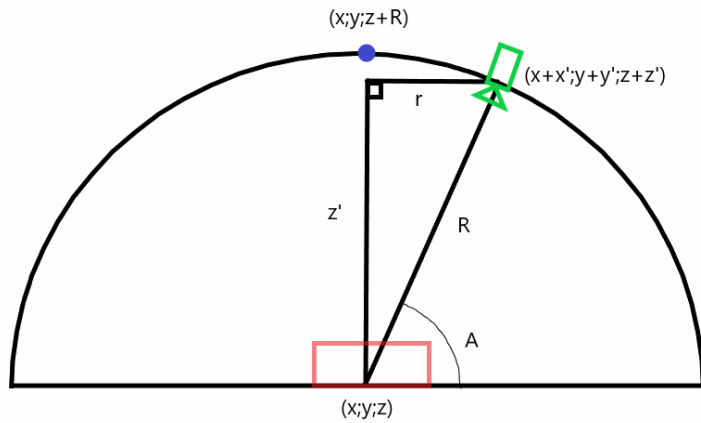
4.2.3. Objektum bejárás

Egy másik általunk készített mozgási algoritmus az objektum bejárás. Ez az algoritmus hasznos a neuronháló tanítása szempontjából, mivel minden objektumról több különböző szögből készít képet. Abban az esetben, ha kellően sok szögből készítünk képet, lehetséges az objektumok háromdimenziós visszaépítése.

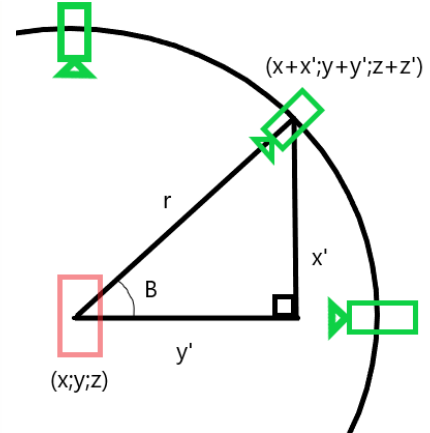
Az algoritmusunk itt is igazán egyszerű. A drón a bejárás során egy konstans távolságot tart az objektumtól, tehát vehetjük úgy, mintha egy láthatatlan félgömb venné körbe a feltérképezett objektumot. A drón először elfoglalja kezdeti pozícióját az objektum felett egy adott távolságra, majd, körbejárja az objektumot, érintve azokat a pontokat, amelyek a félgömb felületén vannak. A drón egy bejárás során többször is körbejárhatja az objektumot. Nem változtatja távolságát a képzeletbeli félgömb alapjától egy körbe járás során, ennek köszönhetően olyan mintha a drón a félgömbön kiválasztott gyűrűkön haladna végig. A gyűrű körbe járása során többször megállhatunk, hogy felvételeket készítsünk.

A bejárás kezdeti pozícióját a 32.1. ábrán kék ponttal jelöltük. Ha az objektum középpontjának koordinátája (x, y, z) és a láthatatlan félgömb sugara R , akkor a kezdeti pont koordinátája $(x, y, z + R)$. Az „A” betűvel jelölt szög adja meg, hogy a drón milyen sugarú

gyűrűn fog végigmenni a félgömb felületén. Egy bejárás során több „A” értéket is megadhatunk, ha szeretnénk, hogy többször körbejárja az objektumot.



32.1. ábra A körbe járt objektum oldalról



32.2. ábra A körbe járt objektum fentről

Az 32.1. és 32.2. ábrákon zöld kamerákkal azokat a pontokat jelöltük, ahol képeket készítünk az objektumról. Ezeknek a pontoknak a koordinátái általánosítva legyenek jelölve, mint $(x + x'; y + y'; z + z')$. Az 32.1. ábráról ki tudjuk számolni az „r” és „z” változókat az alábbi 4. és 5. képletekkel.

$$z' = -R * \sin(A) \quad (4.)$$

$$r = R * \cos(A) \quad (5.)$$

Ezek után a 32.2 ábrán kiszámoljuk az „x’ ” és „y’ ” változókat a lent megjelölt 6. képletekkel.

$$x' = r * \sin(B) \quad y' = r * \cos(B) \quad (6.)$$

Látható, hogy bármely $(x + x'; y + y'; z + z')$ pont meghatározható „R”, „A” és „B” ismeretében. „A” szög változtatásával a félgömb felületén különböző gyűrűket tudunk választani. „B” szög változtatásával a kiválasztott gyűrűn tudunk körbe menni.

4.2.4. Kamera mozgatása

Az objektumbejárás során a drón pozíciója folyamatosan változik az objektumhoz képest, ezért amikor képet akarunk készíteni az objektumról először a kamerát rá kell fordítanunk. A kamera mozgatása AirSim környezetben a 33. ábrán látható függvénnyel történik. A „pose” változó egy Python osztály, ami két vektort tartalmaz. Az egyik vektor a kamera relatív pozíciója.

Mivel azt akarjuk, hogy a kamera maradjon a drónon, ezért ezt a vektort lenullázzuk. A másik vektor egy rotáció quaternion formában elmentve, ez alapján forgatjuk a kameránkat [21].

```
client.simSetCameraPose(pose)
```

33. ábra *SimSetCameraPose* függvény

AirSim környezetben a quaternion-ok felépítése $\{w, x, y, z\}$, ahol „x, y, z” a vektor, amely körül végezzük a rotációt és „w” a szög fele, amellyel elforgatunk az előbb meghatározott vektor körül. A kezdeti orientáció quaternion-ja: $\{0, 1, 0, 0\}$.

Ahhoz, hogy kiszámoljuk a quaternion-t, amellyel el kell forgatni a kameránkat a kezdeti orientációtól úgy, hogy a forgatás után az objektumra nézzünk, több lépést kell végrehajtanunk. Mivel tudjuk a kamera és az objektum abszolút pozícióit és a kamera forgatás előtti orientációját, ezért kiszámoljuk a forgatás előtti orientáció vektoriális formáját a 7. képlet alapján.

$$Q_{\text{bázisvektor}} = Q_{\text{bázis}} * Q_{\text{kezdeti}} * Q_{\text{bázis}}$$

$$V_{\text{bázis}} = Q_{\text{bázisvektor}}(x, y, z) \quad (7.)$$

Ezután kivonjuk az objektum pozíció vektorjából a kamera pozíció vektorját. Normalizáljuk a most kiszámított vektort és a bázis vektort, majd utána a két vektort keresztben összeszorozzuk.

Mivel a két vektor normalizálva van, köztük lévő szög egyenlő a vektorok skaláris szorzatával, amit jelölünk „ θ ”-val.

A következő lépésben felépítjük a rotációs quaternion-t a 8. képlet segítségével:

$$Q_{\text{rotáció}} = \left\{ \cos\left(\frac{\theta}{2}\right), \sin\left(\frac{\theta}{2}\right) * V_{\text{kereszt}_x}, \sin\left(\frac{\theta}{2}\right) * V_{\text{kereszt}_y}, \sin\left(\frac{\theta}{2}\right) * V_{\text{kereszt}_z} \right\} \quad (8.)$$

Ezután már csak meg kell szoroznunk a rotációs quaternion-t a bázis quaternion-al, és megkapjuk az orientációt, amit az AirSim függvénynek meg kell adnunk, hogy ráfordítsa a kamerát az objektumra.

5. Automatikus címkézés

A szimuláció során képeket mentünk el, amelyeket a későbbiekben fel kell használnunk neuronhálónk tanítására és tesztelésére. Ahhoz, hogy az elmentett képeket erre a célra felhasználhassuk, címkéznünk kell az objektumainkat. Az 34. ábrán egy címkézett objektum látható. A címke lefedi az objektumot, úgy, hogy a címke területe a lehető legkisebb legyen. Mivel ez a feladat sok kép esetében nagyon munkaigényes lenne, ezért megoldást kerestünk arra, hogy automatikusan címkézzük objektumainkat.



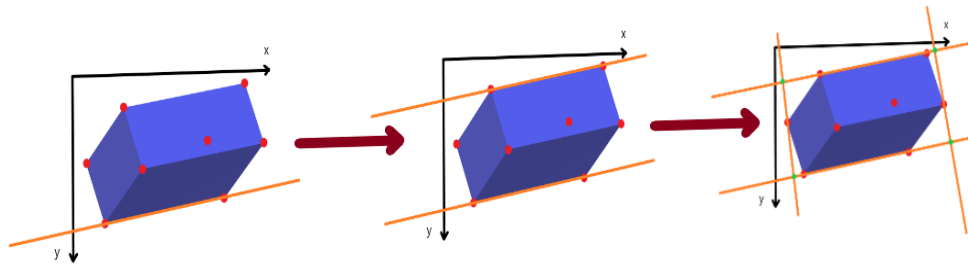
34. ábra Címkézett objektum

Az automatikus címkézés folyamatát három részre lehet osztani. Első lépés, hogy felépítünk egy téglatestet, amely magába foglalja a címkézett objektumot. Második lépésben a téglatest sarkait rávetítjük a kamera képére. Harmadik lépésben a képre vetített pontok alapján úgy helyezünk el egy téglalapot, hogy az a lehető legjobban illeszkedjen az objektumra.

Az első két lépés megoldására már létezett egy függvény, amelyet az AirSim készítői írtak meg. A függvény kiszámolja az objektumot határoló téglatest sarokpontjainak koordinátáit, amelyeket levetít a kamera képére és végül ezeket a pontokat pixelkoordinátákban adja vissza.

A harmadik lépésben az előzőben meghatározott pontok alapján kell egy téglalapot meghatároznunk, amely az objektumot határolja. Erre a feladatra három féle algoritmust dolgoztunk ki, de mindháromnál ugyanazok a lépések vannak. Ezeket a lépéseket a 35. ábra szemlélteti. A sarokpontok alapján meghatározzuk a címke egyik oldalát, majd erre párhuzamosan húzunk egy újabb egyenest, amely a lehető legközelebb van az első egyeneshez úgy, hogy minden pont a két egyenes által közre zárt területen legyen. Ezután meghúzunk két egyenest, amelyek merőlegesek az első egyenesre, ügyelve arra, hogy minél kisebb távolságra legyenek egymástól. A közre zárt terület tartalmazza az összes pontot. Miután meghatároztuk

az egyeneseket, a címkék sarokpontjainak kiszámításához meg kell határoznunk az egyenesek metszeteit.



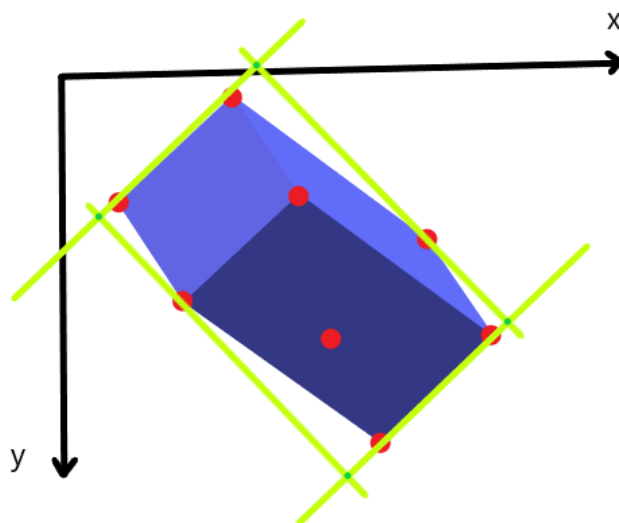
35. ábra Címkzés folyamata

5.1. Címkzési algoritmusok

Három algoritmust írtunk a címkék lehelyezésére. Ezek az algoritmusok csak abban különböznek, hogy a címkék első oldalát más módszerekkel határozzuk meg. A címkék többi oldala az első oldaltól függően kerül elhelyezésre. Kicsi változtatásnak tűnhet csak az első oldal megválasztásának változtatása, ám a gyakorlatban azt láthatjuk, hogy az nagy különbségeket jelent a címkék lehelyezésében.

5.1.1. Minimum pont szerinti oldalválasztás

Ahogy az algoritmus nevéből is következtethető, ennél az algoritmusnál az első oldalt a legkisebb y-koordináta szerinti érték alapján választjuk meg. Az első oldalt egyszerűen a két legkisebb pontot összekötő egyenes fogja meghatározni, ezt szemlélteti a 36. ábra.



36. ábra Minimum pont szerint címkézett objektum

Egy egyenes egyenletét általánosan a következőképpen írhatjuk fel a 9. képlet alapján:

$$y = m * x + c \quad (9.)$$

Itt az „y” és „x” paraméterek az egyenes koordinátái, az „m” paraméter az egyenes irányítányezője, amelyet 10. képlettel számítsuk ki. A „c” paraméter az egyenes magassága, ahol az egyenes az y tengelyt metszi.

Hogy meghatározzuk az első oldal egyenesének egyenletét a két sarokpontból, meg kell határoznunk az egyenes irányítányezőjét és „c” paraméterét a 11. képlettel. Ezt a következő képletekkel tehetjük meg, ahol az „A” és „B” sarokpontok és indexeik a megfelelő paraméterére utalnak.

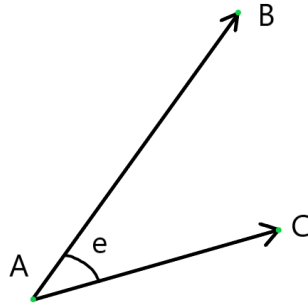
$$m = \frac{A_y - B_y}{A_x - B_x} \quad (10.)$$

$$c = A_y - m_1 * A_x \quad (11.)$$

Ezeknek a kiszámolt paramétereknek a segítségével meghatározzuk az első oldal egyenesének egyenletét, amely segítségével a továbbiakban meghatározhatjuk a többi oldal egyenesének egyenletét is.

Fennállhat olyan eset is, hogy miután összekötjük a két pontot, nem az objektum egy élét kapjuk meg, hanem az egyenes mindkét oldalán lesznek pontok. Ezt az esetet ki tudjuk szűrni, csak annyit kell megnéznünk, hogy az adott egyenesnek csak az egyik oldalán helyezkednek-e el pontok. Tehát végig megyünk az összes ponton, és megnézzük, hogy az egyenesnek melyik oldalán helyezkednek el, ha két pont az egyenes különböző oldalain van, akkor tovább lépünk.

A megoldás megértéséhez vegyünk a 37. ábrán látható vektorokat, itt az „A” és „B” pontok az egyenest meghatározó pontok, a „C” pont a tesztelni kívánt pont, „e” a két vektor által közre zárt szög. Megfigyeljük, hogy az „e” szög szinuszának előjeléből következtethetünk-e arra, hogy a „C” pont az „AB” vektor melyik oldalán helyezkedik el [22].



37. ábra vektoriális szorzat

Ahhoz, hogy az „e” szög szinuszának előjelét kiszámoljuk, írjuk fel a vektorok vektoriális szorzatait két különböző formában, amelyet a 12. és 13. képlet ad meg.

$$AB \times AC = (|AB| * |AC| * \sin(e)) * \vec{n} \quad (12.)$$

$$AB \times AC = \begin{vmatrix} x & y & z \\ AB_x & AB_y & 0 \\ AC_x & AC_y & 0 \end{vmatrix} \quad (13.)$$

Ha egyenlővé tesszük mindkét egyenletnek a jobb oldalát és kiszámoljuk a determinánst, akkor a következő összefüggést kapjuk, amit a 14. képlet mutat.

$$(|AB| * |AC| * \sin(e)) * \vec{n} = (AB_x * AC_y - AC_x * AB_y) * z \quad (14.)$$

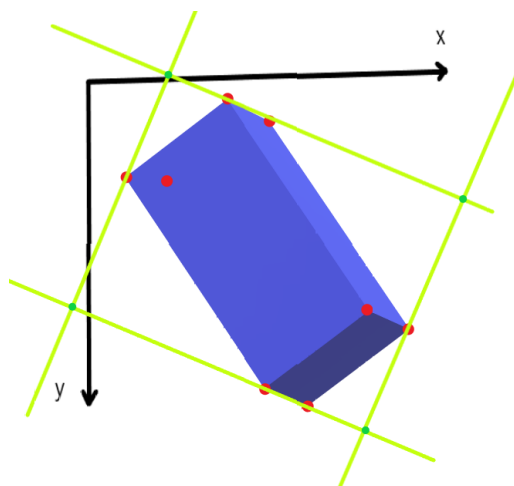
Itt látható, hogy az „e” szög szinuszának előjele meghatározható a 15. képlettel.

$$AB_x * AC_y - AC_x * AB_y \quad (15.)$$

Tehát a kiszámolt összefüggés előjelét figyelve meg tudjuk határozni, hogy az adott pont az egyenes melyik oldalán helyezkedik el, és miután végig vesszük az összes pontot, ami nincs az egyenesen meg tudjuk mondani, hogy az egyenes megfelelő-e a címkézéshez.

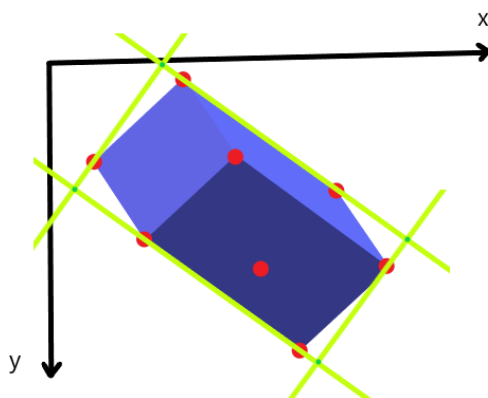
5.1.2. Objektum élei szerinti oldalválasztás

A minimum pont szerinti oldalválasztásos algoritmusnál a probléma az volt, hogy az első él akár két közeli pont által is meghatározásra kerülhetett, ahogy a 38. ábra is szemlélteti. Ennek elkerülése érdekében úgy döntöttünk, hogy a leghosszabb él kerül kiválasztása első címke oldalnak.



38. ábra Nem optimális címke

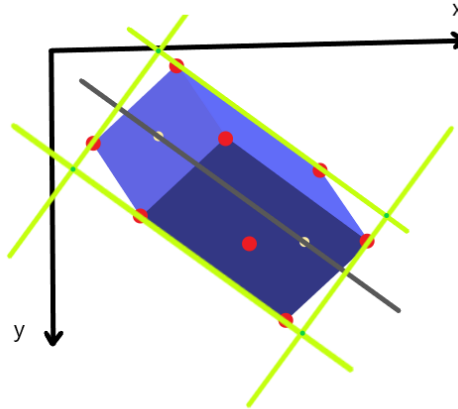
Az él alapján történő oldalválasztás a minimum szerinti algoritmusnak nagy részét átveszi. Mivel csak élek eshetnek a címke szélére, ezért csak azokat a pontpárokat teszteljük, amelyek éleket alkotnak az objektumnál. Keressük a legtávolabb lévő két pontot, miközben folyamatosan vizsgáljuk, hogy az egyenes ne kerüljön más pontok közé. Ezzel az algoritmussal a címkenk a 39. ábrán láthatóhoz lesz hasonló.



39. ábra Leghosszabb oldal szerint címkézett objektum

5.1.3. Objektum tengelye szerinti oldalválasztás

Ez az algoritmus az előző él szerinti oldalválasztás algoritmusának egy optimalizált változata. Itt minden objektumnak meghatározzuk egy tengelyét, ahogy a 40. ábrán is látható, majd ennek a tengelynek az irányítányezője alapján építjük fel a címkenket.



40. ábra Tengely szerint címkézett objektum

Az algoritmus pontosítása érdekében több tengelyt is felvehetünk, majd kiszámoljuk a területét mindegyik tengellyel készített címkének és a legkisebb területű címkét válasszuk.

5.2. Címke oldalainak meghatározása

Miután kiszámoltuk a címke első oldalát bármely tetszőleges algoritmussal, meg kell határoznunk a címke többi oldalát képező egyenest is. A második egyenes az első egyenessel párhuzamosan fekszik, mivel a két egyenes párhuzamos, ezért irányítányezőik megegyeznek. A pont, amelyen a második egyenes áthalad, az első egyenestől legtávolabb elhelyezkedő pont lesz, amit meghatározhatunk a 16. képlettel.

$$\frac{|a \cdot P_x + b \cdot P_y + c|}{\sqrt{a^2 + b^2}} \quad (16.)$$

A képletbe behelyettesítjük az első egyenes egyenletének paramétereit és a P vizsgálandó pont koordinátáit, miután felírjuk azt „ $a \cdot x + b \cdot y + c = 0$ ” formába.

Ha megkapjuk a legtávolabb levő pontot, egyenest húzunk rajta keresztül a kiszámolt paraméterekkel.

A címke utolsó két oldala merőleges az előbb meghatározott oldalak irányítányezőire, ezért ezeknek az oldalaknak irányítányezői $-\frac{1}{m}$ -el lesznek egyenlők, ahol „m” az első két oldal irányítányezője. A pontok, amelyeken átmennek az egyenesek úgy kerülnek meghatározásra, hogy sorra vesszük pontjainkat, és ha kapunk olyan pontot, amelyen ha áthúzzunk egy egyenest és annak csak egy oldalán vannak pontok (ahogy korábban is vizsgáltuk az oldalakat), akkor

azt meghúzzuk. Miután mindkét egyenes meg lett határozva, címkénk már teljes, és készen áll a következő lépésre.

5.3. Címkék sarokpontjainak meghatározása.

Mivel a címkék sarkainak koordinátáira szükségünk van a címkézőprogramba való feltöltéshez, miután kiszámoltuk az objektumot közre záró egyenesek egyenleteit, kiszámoljuk ezek metszéspontjait is. Ezt legegyszerűbben úgy oldhatjuk meg, hogy először meghatározzuk azt általánosan, majd a kapott képletbe behelyettesítjük az egyenesek paramétereit.

Felírjuk két egyenes egyenletét általános formában a 17. képlet alapján.

$$l_a: y_a = m_a * x_a + c_a \qquad l_b: y_b = m_b * x_b + c_b \qquad (17.)$$

Két egyenes metszéspontja esetén egy olyan pontot keresünk, amely minkét egyenesen rajta van, tehát ahol $y_a = y_b$ és $x_a = x_b$. Ezt a megfigyelésünket felhasználva egyenlővé tehetjük a két egyenletet, így az alábbi összefüggést kapjuk:

$$m_a * x + c_a = m_b * x + c_b \qquad (18.)$$

Ebből az egyenletből kifejezzük az „x” paramétert, amelyet visszahelyettesítünk az egyik egyenes egyenletébe. A kapott egyenletből kifejezzük az „y” paramétert. A rendezések után a következő egyenleteket kapjuk, amit a 19. és 20. képlet szemléltet:

$$x = \frac{c_b - c_a}{m_a - m_b} \qquad (19.)$$

$$y = m_a * \frac{c_b - c_a}{m_a - m_b} + c_a \qquad (20.)$$

A kapott értékek együtt az egyenesek metszéspontjának koordinátái. Tehát két egyenes metszéspontjának koordinátái általánosan felírhatók a 21. képlet szerint:

$$l_{ab} = l_a \cap l_b = \left(\begin{array}{c} \frac{c_b - c_a}{m_a - m_b} \\ m_a * \frac{c_b - c_a}{m_a - m_b} + c_a \end{array} \right) \qquad (21.)$$

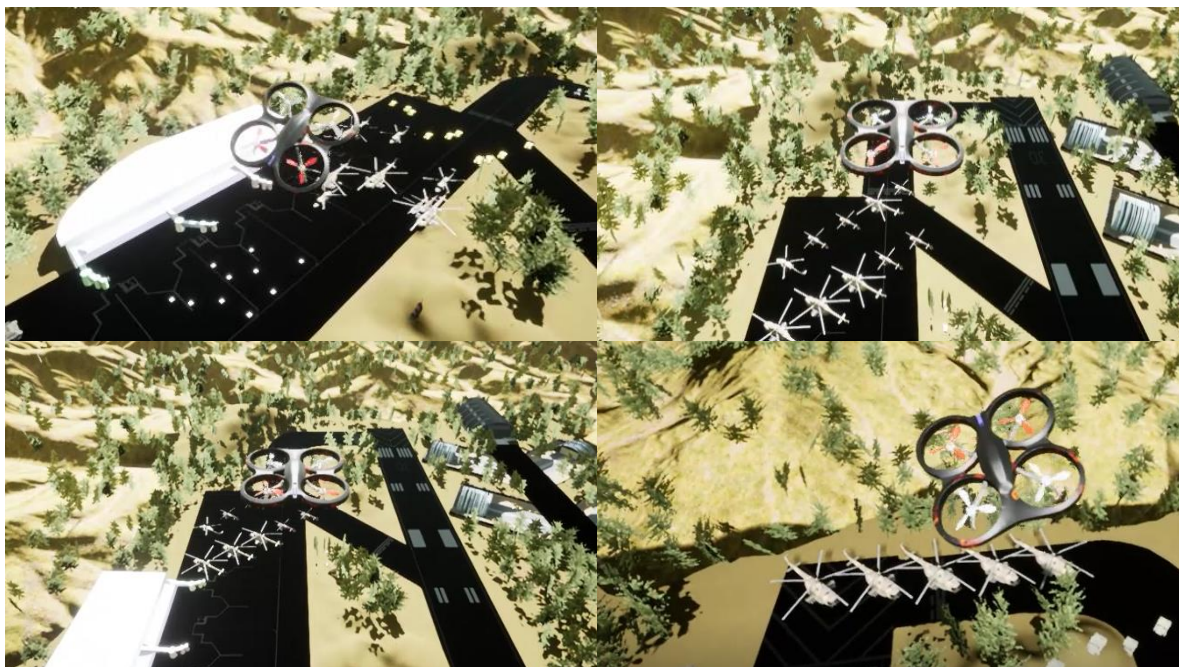
Az általános képlet segítségével felírható a négy sarokpont koordinátája a 22. képletekkel.

$$\begin{aligned}
I_{12} &= l_1 \cap l_2 = \left(\begin{array}{c} \frac{c_2 - c_1}{m_1 - m_2} \\ m_1 * \frac{c_2 - c_1}{m_1 - m_2} + c_1 \end{array} \right) \\
I_{23} &= l_2 \cap l_3 = \left(\begin{array}{c} \frac{c_3 - c_2}{m_2 - m_3} \\ m_2 * \frac{c_3 - c_2}{m_2 - m_3} + c_2 \end{array} \right) \\
I_{34} &= l_3 \cap l_4 = \left(\begin{array}{c} \frac{c_4 - c_3}{m_3 - m_4} \\ m_3 * \frac{c_4 - c_3}{m_3 - m_4} + c_3 \end{array} \right) \\
I_{41} &= l_4 \cap l_1 = \left(\begin{array}{c} \frac{c_1 - c_4}{m_4 - m_1} \\ m_4 * \frac{c_1 - c_4}{m_4 - m_1} + c_4 \end{array} \right) \tag{22.}
\end{aligned}$$

Ezeket a koordinátákat már felhasználjuk a neuronhálónk tanításában és a tesztelésében, átalakíthatjuk bármilyen címkézési adatstruktúrának megfelelően.

6. Mérések

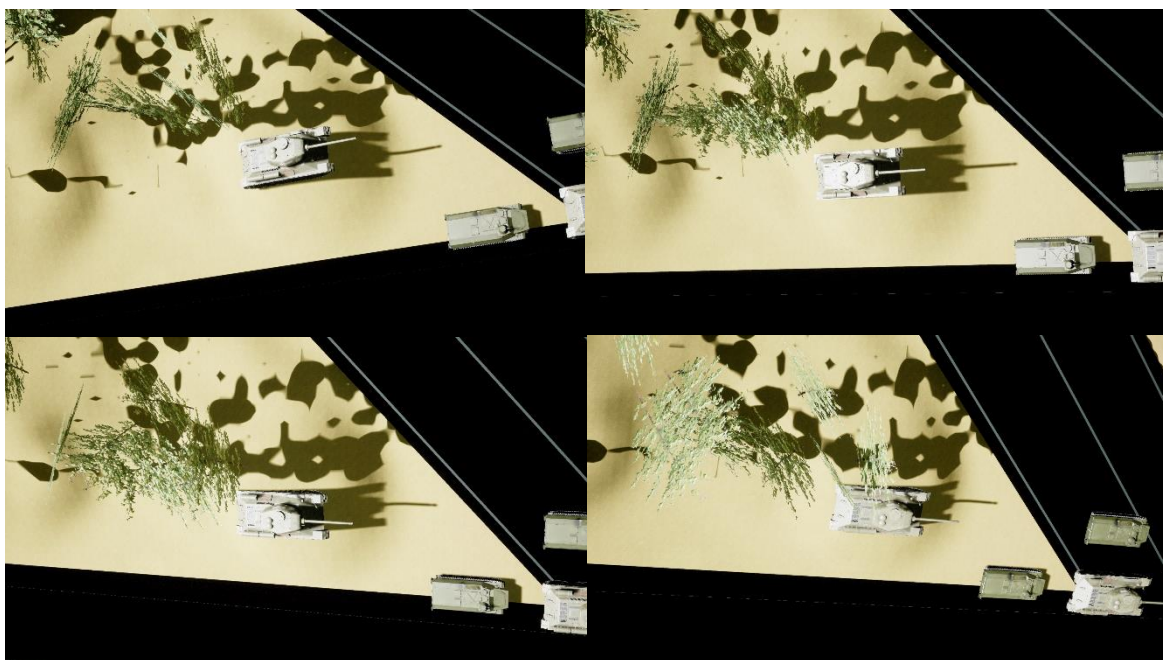
Mérésnek azt a folyamatot nevezzük a szimuláció során, amikor egy adott konfigurációs fájlra lefuttatunk egy bejárást. A mérési eredményekhez hozzá tartoznak a képek, a képekhez kapcsolódó annotációs fájlok, valamint a naplófájlok, amik tartalmazzák az objektumok listáját és azok pontos elhelyezkedést. A drón állapotát is kimentjük a képek készítésének pillanatában, ez tartalmazza a drón kamerájának relatív és abszolút rotációját, a drón helyzetét, a kamera zoomját, a fókusz távolságát, a kép készítésének idejét, és a kép elérési útvonalát. Kimentjük továbbá azokat a koordinátákat, ahová a drón által készített kép sarkai, illetve közepe esni fog a valós GPS koordináta-rendszerben. Ez abban segít, hogy ezáltal meg lehet határozni az objektumok elhelyezkedését a térképen, valamint több képet össze lehet fűzni egy térképpé.



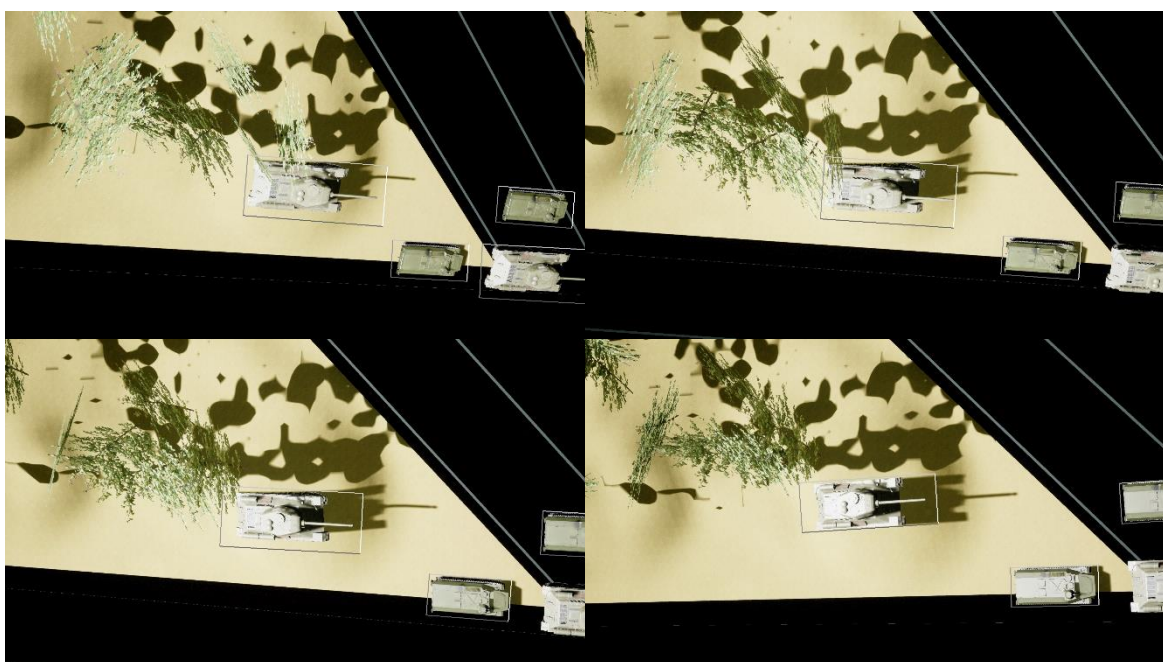
41. Drón külső nézetből

A 41. ábrán egy mérés látható, amit a „level_12_repter” terepen hajtottunk végre. Több típusú harcjárművet helyeztünk el különböző lerakási konfigurációk szerint. A fenti ábrán a drón egy külső kameraállásból látható, miközben a bejárást végzi a terepen.

A 41. ábrán a már 42. ábrán is látható mérés jelenik meg, csak itt már a drón kamerájának kimentett képei láthatók.



82. ábra Képek címkézés nélkül



43. ábra Képek címkézve

A szimuláció során megcímkézett képek láthatók a 43. ábrán. Ezek a képek azonosak a 42. ábrán látottakkal, csak itt megfigyelhető, hogy a megjelenő objektumokat körbeveszi egy keret, amelyet az automatikus címkézési algoritmus tett rá. Az algoritmus itt az elvárások szerint működött, mivel mindegyik objektumot címkézi, azok kivételével, amelyek már túl nagy százalékban lógnak ki a képből. A bejárési algoritmusok beállításainál lehet ezt a paramétert

változtatni az alapján, hogy az objektumok mennyire lóghatnak ki a képből, annak érdekében, hogy azok még címkézve legyenek.



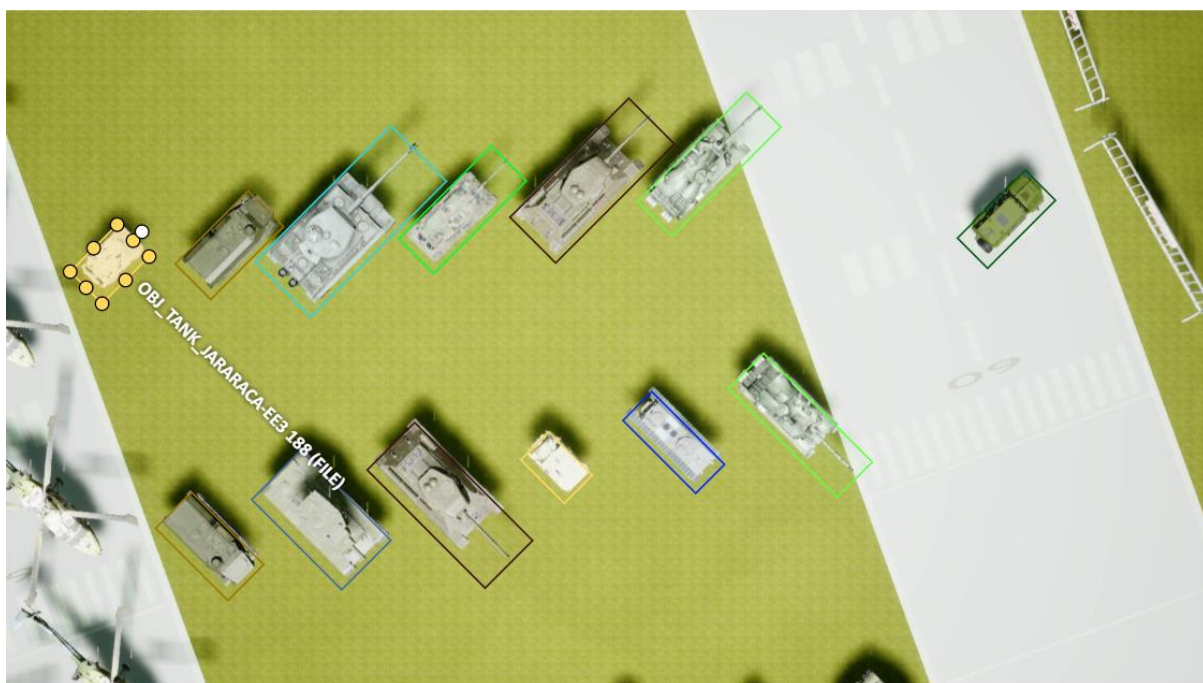
44. ábra Címkézés különböző időjárási viszonyok között és napszakokban

A címkézés nagy előnye, hogy változatos időjárási körülmények között és napszakokban is tudunk képeket generálni, amint az a 44. ábrán is látszik. A képfelismerő algoritmusok hatékonysága ilyen körülmények között nagyban csökken, de a szimuláció általi automatikus címkézést ezek a külső paraméterek nem befolyásolják. A szimuláció a későbbiekben elősegítheti a drónok valós körülmények közötti objektumfelismerését hasonló körülmények között is, a nagy mennyiségű címkézett adatnak köszönhetően.

Konklúzió

A fő célunk a projekt során egy olyan szimulációs környezet kialakítása volt, amely minimális emberi erőforrást igényel, és nagy mennyiségű adatot képes előállítani neuronhálók tanítása céljából. A projektet egy féléven keresztül fejlesztettük, főképpen a nyári gyakorlat ideje alatt. A fejlesztés első részében sok időt töltöttünk a szimulációs környezethez szükséges funkcionálisok megtervezésével, majd ez alapján választottuk ki a megfelelő fejlesztői környezetet, amely már tartalmaz a szimulációhoz szükséges alap funkcionálisokat, ezzel biztosítva, hogy minél hatékonyabb megoldásokat dolgozhassunk ki.

A projekt végrehajtása során elértük, hogy változatos terepekre, különböző objektumokat tudunk elhelyezni különböző formációkban. Ezeket a terepeket szimulált drónunk automatikusan képes berepülni az általun megadott bejárési módszerek alapján, miközben képeket készít a terepen elhelyezett objektumokról, valamint elmenti azok helyzetét is. A bejárások során a drón különböző pozícióit is rögzítjük a képek elkészítésének pillanatában. Az elkészült képeket automatikusan címkézi egy különálló folyamat a rögzített adatok felhasználásával, majd feltölti egy címkéző alkalmazásba, hogy felhasználhassuk azokat neuronhálók tanítására.



45. ábra Címkézett objektumok

A 45. ábrán látható néhány címkézett objektum, amelyek a szimulációs környezetben kerültek elhelyezésre az előre megadott terepen, majd a drón által készített képek címkézése automatikusan meg is történt. Megfigyelhető, hogy azon objektumok, amelyek kiálló alkatrészrel rendelkeznek, azoknál a címkék nem teljesen fedik az objektumokat. A további fejlesztések során az a címkézés algoritmusát még tovább szeretnénk optimalizálni annak érdekében, hogy minél kevesebb zaj kerülhessen a címkézett objektumra. Ugyanakkor az megállapítható a mérési eredmények alapján, hogy sikerült egy komplex automatizált rendszert összeállítanunk, ami hatékonyan képes támogatni a projekt végső célkitűzéseit.

A TKP2021-NVA-16 számú projekt, a Magyar Innovációs és Technológiai Minisztérium által a Nemzeti Kutatási, Fejlesztési és Innovációs Alapból, a TKP2021-NVA támogatási konstrukció keretében finanszírozott támogatásával valósult meg a kutatás.

Felhasznált irodalom

- [1] A. Németh: Technical Dimensions of the Development of Unmanned Aerial Systems and Their Impact on Public Service Uses, AARMS, pp. 149-163., 2018.
- [2] G. Udeanu, A. Dobrescu és M. Oltean: Unmanned aerial vehicle in military operations, Sci. Res. Educ. Air Force, 1. kötet 18. szám, pp. 199–206., 2016.
- [3] A. Németh, K. Virágh, Mesterséges intelligencia és haderő – A mesterséges intelligencia területei III. rész, Haditechnika, 56. évfolyam, 3. szám, pp. 2-7., 2022.
- [4] A. Németh és K. Virágh, Mesterséges intelligencia és haderő – Katonai alkalmazási lehetőségek VII. rész, Haditechnika, 57. évfolyam, 1. szám, pp. 2-6., 2023.
- [5] A. Németh és K. Virágh, Mesterséges intelligencia és haderő – További katonai alkalmazási lehetőségek VIII. rész, Haditechnika, 57. évfolyam, 2. szám, pp. 2-5, 2023.
- [6] B. Alvey, D. T. Anderson, A. Buck, M. Deardorff, G. Scott és J. M. Keller, Simulated photorealistic deep learning framework and workflows to accelerate computer vision and unmanned aerial vehicle research, in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021.
- [7] E. Bondi, D. Dey, A. Kapoor, J. Piavis, S. Shah, F. Fang, B. Dilkina, R. Hannaford, A. Iyer, L. Joppa és M. Tambe, „AirSim-W, A Simulation Environment for Wildlife Conservation with UAVs, in Proceedings of the 1st ACM SIGCAS Conference on Computing and Sustainable Societies, New York, NY, USA, 2018.
- [8] P. Mittal, R. Singh és A. Sharma, Deep learning-based object detection in low-altitude UAV datasets: A survey, Image and Vision computing, 104. kötet, p. 104046, 2020.
- [9] H. Yu, G. Li, W. Zhang, Q. Huang, D. Du, Q. Tian és N. Sebe, The unmanned aerial vehicle benchmark, Object detection, tracking and baseline,” International Journal of Computer Vision, 128. kötet, pp. 1141-1159, 2020.
- [10] C. Rui, G. Youwei, Z. Huafei és J. Hongyu, A comprehensive approach for UAV small object detection with simulation-based transfer learning and adaptive fusion, arXiv preprint arXiv:2109.01800, 2021.
- [11] J. Lee, J. P. Doran és N. Misra, Unreal Engine: Game Development from A to Z, Packt Publishing Ltd, 2016.
- [12] S. S. Revision, Airsim Documentation, [Online]. Available: <https://airsim-fork.readthedocs.io/en/docs/> [Hozzáférés dátuma: november 2023]
- [13] CVAT.ai, CVAT Python SDK, [Online]. Available: https://opencv.github.io/cvat/docs/api_sdk/sdk/ [Hozzáférés dátuma: november 2023].
- [14] C. Prodi és T. Out, Geodetic datum
- [15] Epic Games, Using a Single Line Trace, [Online]. [Hozzáférés dátuma: november 2023]
- [16] Epic Games, Level Editor in Unreal Engine, [Online]. Available: <https://docs.unrealengine.com/5.0/en-US/level-editor-in-unreal-engine/> [Hozzáférés dátuma: november 2023]

- [17] Cesium Gs, Learning Cesium, [Online]. Available: <https://cesium.com/learn/> [Hozzáférés dátuma: november 2023]
- [18] Wysilab, Instant Terra Documentation, [Online]. Available: <https://www.wysilab.com/OnLineDocumentation/index.html> [Hozzáférés dátuma: november 2023]
- [19] OpenDroneMap, OpenDromeMap's documentation, [Online]. Available: <https://docs.opendronemap.org/> [Hozzáférés dátuma: november 2023]
- [20] Gaea, Gaea Documentation, [Online]. Available: <https://docs.quadspinner.com/Guide/index.html>
- [21] J. Voight, Quaternion algebras, Springer Nature, 2021.
- [22] H. Jeffreys és B. Jeffreys, Methods of mathematical physics, Cambridge university press, 1999.