Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Irányítástechnika és Informatika Tanszék

Bertalan Bálint

KITERJESZTETT VALÓSÁG

beltéri objektumkövetéssel

KONZULENS

Dr. Magdics Milán

BUDAPEST, 2021

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

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

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

Kelt: Budapest, 2021. 12. 08.

...…….

Bertalan Bálint

Összefoglaló

Az elmúlt évek során egyre inkább elterjedt, hogy a telefon kijelzője helyett valamilyen más módon kerüljön megjelenítésre az információ a felhasználó számára. Ezek közül az egyik megoldás, hogy amennyiben az egyén egy okosszemüveget hordana, akkor lehetősége lenne az alkalmazásnak az információkat a kijelző helyett a valóságra rajzolva megjeleníteni. Ennek előnye, hogy így a kapott adat könnyeben feldolgozható, hiszen a kontextust a felhasználó sokkal könnyebben megérti. Például amennyiben egy térkép alkalmazást használnánk, akkor nem a kétdimenziós rajzból kellene tájékozódnunk és kitalálnunk, hogy az egyes utak a térképen a valóságban melyiknek felelhet meg, hanem egy olyan megoldást is létre lehetne hozni, ahol ez az előttünk található járdára egyértelműen megjelenítésre kerül. Ekkor ezt az információt csak mi látnánk, azonban mozgás közben, további eszközök használata nélkül meg tudnánk mondani, hogy merre kell folytassuk az utunkat.

A diplomatervezésem folyamán, elkészítésre kerül egy olyan alkalmazás, melynek célja, hogy a kiterjesztett valóság segítségével egy megfigyelési területen belül megjeleníti az ott tartózkodó személyeket. Ez alatt a rendszer azt érti, hogy minden szereplőhöz tartozik egy azonosító – mely emberek esetén a névét jelenti – és ez valamilyen módon a látott képre kerül rárajzolásra. Ennek előnye, hogy amennyiben egy alkalmazott keres egy másik ott dolgozó embert, akkor nem kell bejárnia a teljes területet, hanem az alkalmazás segítségével pontosan meg tudja állapítani, hogy milyen irányban is van tőle. Annak érdekében, hogy azt is meg lehessen mondani, hogy milyen távolságra, a két személy között egy távolság meghatározására is szükség van.

Ennek a folyamatnak a megvalósításához 2 eltérő komponensre van szükség, melyek együttes valós idejű működése kell ahhoz, hogy a rendszer megfelelő módon működjön. Ezek közül az első egy beltéri helymeghatározó rendszer, melynek segítségével képesek vagyunk az egyes karaktereket követni. Ezt tovább kell küldeni a második komponensnek, mely a megjelenítést és a további kalkulációt végzi. Ahhoz, hogy ez a rendszer valós időben egyszerre több felhasználót is ki tudjon szolgálni, szükségessé vált egy további elem, melynek célja, hogy a feldolgozott adatokat tárolja és lehetőséget biztosítson azok a lekéréséhez.

A diplomán során az előző bekezdésben bemutatott rendszer implementálását valósítottam meg, kezdve a beltéri helymeghatározó rendszer kiválasztásával, valamint a hozzá tartozó szoftver megvalósításával. Ezt követte a szerver elkészítése a hozzá tartozó elérésekkel, melyekkel az adatok tárolhatóvá és módosíthatóvá váltak. Végül a kliens alkalmazás létrehozása következett, mely során az előző lépések alatt létrehozott adatstruktúra megjelenítése került megvalósításra. Végezetül a rendszer teljes egészében tesztelésre került, leellenőrizve, hogy a megvalósított funkciókat megfelelő módon támogatja-e, majd az így kapott adatokból egy konklúzió került kiolvasatra.

Abstract

Over the past few years, there has been a growing trend to present information to the user in a different way than on the phone screen. One of these solutions is that if the individual were to wear a pair of smart glasses, the application would be able to display the information by drawing it on the realty instead of the display. The advantage of this is that the data can then be processed more fluently, as the context is much easier for the user to understand. For example, if we were to use a map application, we would not have to navigate from the two-dimensional map and figure out which of the roads on the map might correspond to which but could create a solution where this is clearly displayed on the pavement in front of us. Only we would see this information, but we would be able to tell where we should continue our journey while moving, without the use of additional tools.

During my thesis, I will create an application that aims to use augmented reality to display the people within a surveillance area. By this, the system means that each actor is associated with an identifier - which in the case of humans is their name - and this is somehow superimposed on the image they see. The advantage of this is if an employee is looking for another person working there, they don't have to walk the entire area, but can use the app to find out exactly which direction they are in. In order to be able to tell how far away, it is also necessary to determine a distance between the two people.

To implement this process, 2 different components are required, which need to work together in real time for the system to function properly. The first of these is an indoor positioning system that allows us to track individual characters. This has to be passed on to the second component which does the rendering and further calculation. In order for this system to be able to serve several users simultaneously in real time, an additional component is needed to store the processed data and provide the possibility to retrieve it.

During the thesis, I have implemented the system described in the previous paragraph, starting with the selection of the indoor positioning system and the implementation of the corresponding software. This was followed by the creation of the server with the corresponding accesses to store and modify the data. Finally, the client application was created, which implemented the data structure created in the previous steps. Finally, the system was tested in its entirety, checking that the implemented functions were supported properly, and a conclusion was drawn from the resulting data.

1 Bevezetés

A diplomatervem célja egy olyan kiterjesztett valóság alkalmazás létrehozása melynek segítségével alkalmazottak nyomon követhetőek egy területen belül és pozíciójuk megjeleníthető más alkalmazottak számára. Azaz amennyiben van például egy raktárunk, ahol több ember is dolgozik, akkor minden emberhez rendelünk egy azonosítót. Ez célszerűen a nevét jelenti, de akár becenevet is, amennyiben azzal is egyértelműen beazonosítható a területen belül. Ezt követően egy szem előtt található kijelző segítségével a valóságra rajzolva megjelenítjük az egyes emberekhez tartozó személyek azonosítóját, továbbá az egymástól való távolságukat.

1.1. ábra Minecraft képernyőkép

Ennek elképzeléséhez a legjobb példa egy videojáték. A cél egy hasonló megoldás létrehozása a valóságban. Ahogyan az 1.1. ábra is mutatja, a szemben lévő játékos felett megjelenik a neve. Ez kerülne kiegészítésre annyival, hogy a név alatt megjelenne a felhasználótól való távolság is. Az egyes személyeken felül ez alkalmazásban célszerű a területen – akár autonóm, akár irányított formában – mozgó bármely más jármű követése is, ideértve az AGV-ket, targoncákat és drónokat is. Ezeknek elnevezése hasonló az emberekhez, viszont itt egy minden alkalmazottnak egyértelmű elnevezés választása a célszerű. Ennek haszna, hogy így biztonságosabbá lehet tenni a területet, hiszen amennyiben a felhasználók látják a járművek pozícióját azok mellett és előtt óvatosabban fognak közlekedni.

A diplomaterv során megvalósítandó program esetén a követendő objektumokat a tényleges emberekre korlátoztam, mivel a tesztelés során nem férhetem hozzá önvezető targoncához, vagy drónhoz. A megjelenítésnek a megvalósítása egy több lépcsős folyamat, mely több eltérő eszköz és rendszer együttes működését igényli. A következő

alfejezetben ezeknek a bemutatása következik, rátérve az egyes komponensekkel szemben támasztott követelményekre is.

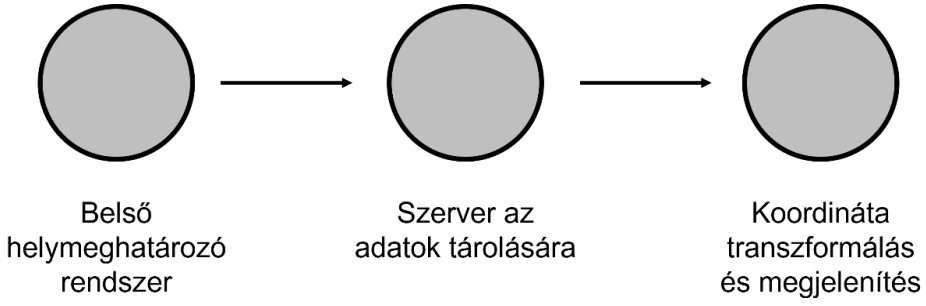
1.1 A folyamat bemutatása

A teljes folyamat 4 eltérő részből épül fel. Első lépésként meg kell határozni az egyes követendő objektumok pozícióját egy 3 dimenziós koordináta-rendszerben. Ezt követően az így kapott adatokat el kell tárolni egy szerveren, ahonnan a további alkalmazások ezt szükség esetén ezt el tudják érni. Az így kapott adatokat az egyes személyek saját koordináta-rendszerébe kell alakítani, ezzel kiiktatva, hogy ugyan azt a pontot más emberek más helyen lássák. Végezetül utolsó lépésként az adatok ismeretében további módosítások segítségével meg kell jeleníteni azokat a valóságra vetítve.

Fontos megjegyezni, hogy abból kifolyólag, hogy az emberek eltérő magasságúak, máshogy viselik a megjelenítőt, valamint az egyes AR megjelenítő eszközök koordináta-rendszere bármilyen helyzetben lehet, az azokban található koordináták nem feleltethetőek meg egy az egyben a beltéri helymeghatározóból származó adatokkal. Hogy ezt megtehessük egy transzformáció meghatározása szükséges, a két rendszer között, mellyel a koordináták oly módon módosíthatóak, hogy azok orientációja és pozíciója megegyezzen.

1.1.1 Eszközök

A fenti folyamat megvalósításához több eltérő eszközre van szükség, melyeknek képesnek kell lennie az egymással való együttműködésre és egy megbízható, megközelítőleg valós időben működő megoldás biztosítására. A folyamat 3 eltérő komponensen keresztül történik, melyeket a 1.2. ábra mutat. Ezek egy valós rendszerben nem biztos, sőt a kliens alkalmazás esetén biztosan nem egy azonos számítógépen helyezkednek el, így a belső működésen felül biztosítani kell egy-egy interfészt, mellyel az egymás közötti kommunikáció megvalósítható, akár interneten keresztül is.



1.2. ábra: A folyamat felépítése

Az alkalmazáshoz szükséges egy beltéri helymeghatározó rendszer melynek segítségével meg tudjuk állapítani, hogy a területen belül tartózkodó objektumok valós időben hol helyezkednek el. Ennek a rendszernek képesnek kell lennie egy 2D-s

koordináta-rendszerben a pozíció meghatározására, tovább egy felületet kell biztosítani a benne található információk folyamatos lekérésére. Mivel beltéri alkalmazásról beszélünk, így bizonyos megoldások, alapvető működésükből kiindulva nem jöhetnek szóba, ezért először egy az elvárásoknak megfelelő eszköz megtalálása volt a cél. Ehhez meg kell vizsgálni a jelenleg is piacon lévő megoldásokat, majd azokat egymással összehasonlítani. Ennek az eredménye adja a folyamat szempontjából legalkalmasabb eszközkészletet. Amennyiben ez a kiválasztás megtörtént, akkor szükséges egy program elkészítése, mely a pozíció adatokat lekérdezi helymeghatározásra használt eszközből és továbbítja azokat a szerver felé.

A rendszer második összetevője egy szerver, melynek segítségével az egyes jeladókhoz tartozó információkat lehetséges tárolni. Itt egy felületet kell biztosítani mind a beltéri helymeghatározóból származó adatok feltöltésére, mind pedig a megjelenítést végző alkalmazás számára az adatok letöltésére. A szervernek ezen felül tárolnia kell, hogy az egyes pozíciók, pontosan mely személyekhez is tartoznak, hogy a megjelenítés során a megfigyelt egyén felett a megfelelő név jelenjen meg. A diplomaterv során a szerver mögött lévő infrastruktúra kidolgozása nem cél, így arról a dolgozatban nem lesz szó.

Harmadik és egyben utolsó komponens a klienseken külön-külön futó megjelenítést végző alkalmazások, melyek első lépésként áttranszformálják a helymeghatározó rendszerből származó koordinátákat a felhasználó saját lokális koordináta-rendszerébe. Az így kapott módosult koordinátának segítségével pedig a felhasználó előtt található megjelenítőre kirajzolásra kerül a megfelelő pixeleken a személyhez tartozó név és távolság. A megvalósításhoz egy kiterjesztett valóság szemüveg használata lenne az ideális, azonban annak hiányában az eszköz és egyben a számításokat elvégző komponens is cserélhető egy Android, vagy iOS alapú eszközzel. Emiatt került az átalakításokat elvégző folyamat is ezen komponensre, mivel alapvető, a megjelenítés szükséges számításokat egy mai eszközzel valós időben elvégezni nem probléma.

1.2 Jelenlegi megoldások

A fenti rendszer nem egy teljesen egyedi megoldás, léteznek ehhez hasonló fejlesztések az iparban. A jelen fejezet tartalmazza ezen alkalmazások rövid bemutatását, kitérve arra, hogy milyen rendszerek működnek jelenleg is a vállalati szférában, mellyel a felhasználók mindennapjai valamilyen módon egyszerűbbé tehetők.

1.2.1 Általános alkalmazások

Az AR alkalmazásokat két eltérő csoportba soroltam, melyek közül az első az általános alkalmazások. Ezek azok a programok, melyeket nem egy specifikus problémára fejlesztettek ki a vállalatok, hanem egy bárki által használható, a legtöbb esetben ingyenesen elérhető megoldás. Az alfejezetben a bemutatott szoftverek nagyobb

változatosságok fednek le, melynek célja, hogy egy átfogó képet nyújtson a napjainkban használt AR lehetőségeiről.

Google Maps Live View

Az első példát az AR használatára a Google vállalat hozta létre a Google Maps keretein belül [1]. Ennek célja, hogy a felhasználó nem a térképen kell, hogy tájékozódjon, hanem amennyiben a telefont maga elé tartja, akkor a látott képre az alkalmazás rárajzolja, hogy hol szükséges az embernek megfordulnia, valamint, hogy egy kereszteződésben melyik útnak mi a neve. A megoldás később bővült olyan lehetőségekkel is, hogy amennyiben egy járókelő elhalad egy étterem vagy egy bolt előtt, amely a Google Maps-en megtalálható, akkor amennyiben ránéz a bolt ikonjára, ott megjelenik annak neve és nyitvatartási ideje. Az alkalmazás azonban nem folyamatos használatra lett kitalálva, hanem arra, hogy amennyiben a felhasználó egy olyan helyre ér, ahol nem biztos az útirányban, akkor a telefont feltartva az hamar meg tudja állapítani.

Működése során az alkalmazás csak olyan területeken lehetséges használni, ahol elérhető a Street View. Ennek oka, hogy a rendszer a jelenleg látott kép és az elmentett közötti hasonlóságokból próbálja meghatároznia a felhasználó orientációját és pozícióját. Amennyiben ez nem áll rendelkezésre, akkor azt jelzi a felhasználó felé. Az AR-rel való pozíciómeghatározás, csak egy második lépés, alapvetően GPS-t is alkalmaz a rendszer. A fentiekből következően a megoldásnak vannak előnyei, mivel egy gyorsan átlátható tájékozódási megoldást biztosít. Azonban vannak hátrányi is, miszerint folyamatos internetkapcsolattal kell rendelkezni, és a Google-nek a környező tájról rendelkezni kell elégséges mennyiségű információval.

Pokemon Go

1.3. ábra: A Pokémonok a fizikai környezetben [2]

Az AR megoldások megjelentek a játékokban is [3]. Ezek közül az egyik legismertebb a Pokémon Go. A játékot a Nintendo adta ki Android és iOS eszközökre. A játék célja,

hogy a Pokémonok összegyűjtéséhez és fejlesztéséhez nem csupán egy alkalmazás elindítására volt szükség, hanem fizikailag egy megfelelő helyen kellett lenni, hogy egy karaktert megszerezzen a játékos. A játék során két eltérő terület volt, melyek a „PokéStops", melyhez eljutva Pokémonok szerezhetőek meg, valamint a „Pokémon Gyms", ahol a játékosok egymás ellen játszhattak. A felhasználók ezeken a helyeken az egyes karaktereket a kiterjesztett valóság segítségével a fizikai környezetben láthatták. Ezt mutatja a 1.3. ábra.

A játéknak több előnye és ugyan annyi hátránya is volt. Az egyik legnagyobb előny, hogy a játékosokat rávette a mozgásra, így létre sikerült hozni egy olyan kikapcsolódást, mely a szabad levegőn történt. Elérte az felhasználóknál, hogy olyan területekre is elutazzanak, ahova alapesetben nem mentek volna és így felfedezzék a környezetüket. Ennek oka, hogy egyes Pokémonok csak bizonyos helyeken jelentek meg, így, ha minden karakter össze akarta gyűjteni a játékos, akkor be kellett járnia nagyobb területet. Azonban pontosan ez okozta az egyik veszélyforrását is a játéknak, hogy egyes játékosok olyan helyekre is eljutottak, ahová nem lett vonna szabad, mint például az Észak és Dél-Korea között elhelyezkedő demilitarizált övezetbe.

Marxent 3D

Harmadik általános alkalmazás a kereskedelemben jelent meg [4]. Ennek szerepe, hogy amennyiben az vásárló egy bútor megvásárlásán gondolkodik, akkor biztosítanak számára egy felületet, melyen keresztül a kiválasztott termék elhelyezhető a saját lakásában és össze tudja így hasonlítani, hogy bele illik-e a már ott lévő bútorok közé. Ez a megoldás a már létező AR platformokra épül (lásd később: 3. Kliens alkalmazás) és használja fel a megvalósításra. Emiatt a rendszer gyorsan működik mind a két mobil termékcsaládon.

1.2.2 Vállalati alkalmazások

Az AR alkalmazások egy másik nagy csoportja azok a megoldások, amelyben egy vállalat egy saját magánál specifikus problémára hoz létre applikációt. Ezek az átlag felhasználók számára nem érhetőek el, hanem a céljuk, hogy a munkavállalóknak biztosítsanak valamilyen szintű segítséget a feladataik elvégzése során. Ilyen megoldásokra példa a következő AR program.

Pick-by-Vision

A megoldás célja [5], hogy segítse, egy logisztikai vállalatnál a raktárban történő komissiózás végrehajtását. A komissiózás „az áruk megadott megrendelések szerinti kigyűjtését és összeválogatását jelenti. A folyamat a megrendelés átvételével kezdődik és a kigyűjtött áruk rendelésenkénti összeállításával fejeződik be." [6] Ennek során a felhasználó rendelkezik egy okosszemüveggel, mely folyamatosan információt szolgáltat az alkalmazottnak, arról, hogy mely termékeket kell kiválasztania és azokból hány darabot.

Ezen felül a megoldás kiegészülhet további funkciókkal is, melyek között szerepel, hogy nem csak azt mutatja meg a rendszer, hogy mely termékből hány darabot szükséges kivenni, hanem azt is, hogy az adott tároló hol található. Ennek többféle megvalósítása is van, például egy olyan, ahol nyíllal mutatják, hogy hova kell mozognia a felhasználónak, majd amennyiben eljut a célhoz, akkor megmutatják, hogy milyen cikkszámú termékből hány darabot kell kivenni. Erre mutat egy példát a 1.4. ábra, ahol a sárga nyíl mutatja, hogy melyik tárolóból kell kiválasztani a terméket, a bal oldali szám pedig a hátralévő elemek számát.

1.4. ábra: Egy pick-by-vision rendszer használat közben

Ez a lépcső kiegészíthető, hogy az egyes rekeszekhez hozzáadunk valamilyen QR kódot, vagy egy vonalkód felismerő rendszert helyezünk el az okosszemüvegbe, továbbá adunk egy QR kódot minden dolgozónak is. Amennyiben sikeresen kiszedte az árut, akkor a két azonosítót egymás mellett tartva nyugtázza a rendszer felé, hogy sikeresen megtörtént a folyamat, lehet a következő lépésre ugrani. Ennek előnye, hogy a teljes folyamat automatikusan történhet, így nincs szükség arra, hogy a komissiózást végző egyénnek egy másik eszközt is használnia kelljen, ahol vezeti, hogy hol tart a folyamattal és emellett amennyiben rendelkezik a vállalat vállalatirányítási rendszerrel, úgy annak frissítése is magától megtörténhet.

A fenti megoldások megmutatják, hogy az AR az elmúlt időszakban egyre nagyobb teret kezd nyerni mind a fogyasztói piacon, mind pedig a vállalati szférában. Az elmúlt alfejezet erre mutat példákat, mely alátámasztja, hogy egy a diplomatervben is részletezett megoldásnak lehet helye. A fejezetben ezen felül bemutatásra került az alap koncepciója is az alkalmazásnak, melyek közül a következő fejezet a beltéri helymeghatározó rendszer kiválasztásával és a hozzá tartozó alkalmazás elkészítésével fog foglalkozni.

2 Pozíció adatok meghatározása

Az előző fejezetben részletezett folyamat első lépéseként meg kell határozni a területen tartózkodó objektumok pontos helyét. Ehhez, szükségessé vált egy valós idejű beltéri helymeghatározó rendszer kiépítése. Azonban ilyen rendszerből több eltérő fajta is létezik, melyek mind más és más előnnyel rendelkeznek. Annak érdekében, hogy ezek között sorrendet lehessen felállítani meg kellett határozni a rendszerrel szemben támasztott követelményeket, melyek a következők:

- beltérben való működés,
- valós idejű pozíció meghatározása,
- könnyű bővíthetőség
- és pontosság (~10 cm).

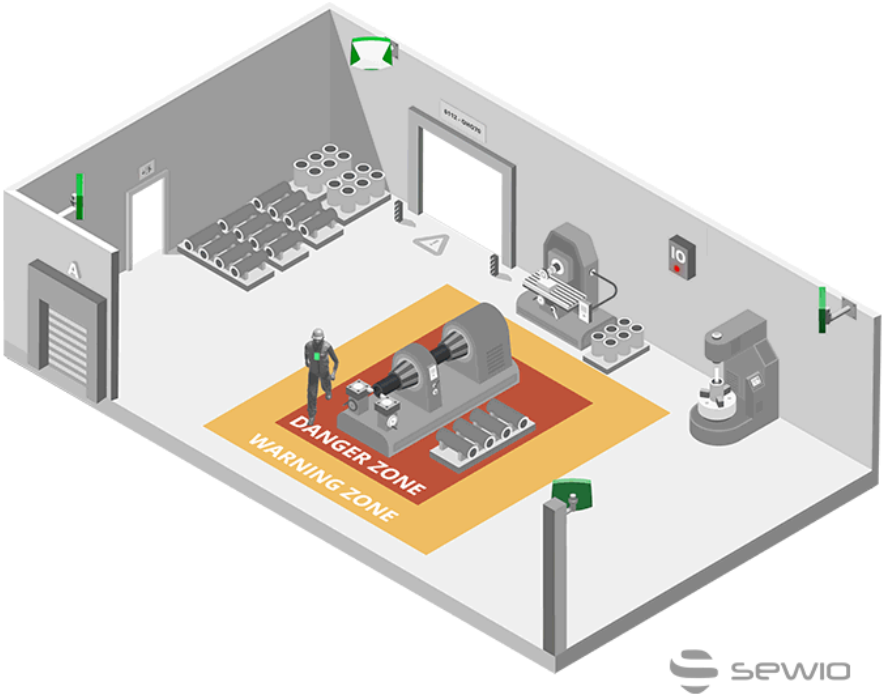
2.1 Választható eszközök

Hogy a fenti elvárásoknak megfelelő eszközt találjunk össze kellett gyűjteni a piacon elérhető megoldásokat [7] és azokat több szempont szerint is osztályozni kellet. Ennek eredményeként kialakul, hogy melyik rendszercsoport felel meg a támasztott igényeknek a legjobban. A következő részben röviden felsorolásra kerülnek a lehetséges opciók, melyek többséges már elterjedt megoldás, valamint további részletesebben bemutatásra kerül a kiválasztott eszköz.

- GPS: első opcióként, amennyiben helymeghatározásra kerülne sor, a GPS ez első, amely eszünkbe jut. Ez egy globálisan működő megoldás, továbbá plusz eszközt egy vevőn kívül nem igényel. A jelenlegi helyzetben azonban ez az opció nem volt egy valós lehetőség, mivel a GPS használhatósága beltérben kétséges. Ezen felül, a jel precizitásával is probléma van, hiszen az átlag polgári felhasználók számára elérhető jel pontossága közel 5 méter, így ezzel emberek követése valós időben nem lehetséges.
- Wifi: második opcióként a belső vezeték nélküli hálózat használata merült fel, mint nyomkövető rendszer. Ehhez a legtöbb felhasználó alapvetően már amúgy is csatlakozik, továbbá a legtöbb vállalat számára nem kerül plusz költségbe, hiszen a wifi hálózat a terület legnagyobb részén már ki van építve. Ezzel szemben azonban nem teljesül a pontossággal kapcsolatos elvárás, mivel csak 10-15 méteres pontosság biztosítására képes.
- Kamera: következőként egy kamera rendszer felszerelése tűnt egy választható megoldásnak. Felépítését tekintve ez úgy működik, hogy több kamerát helyezünk el a területen, melyek folyamatosan feldolgozzák a saját képüket és amennyiben egy objektum több kamera képén is megjelenik, ki lehet számolni a köztük lévő távolság függvényében a pontos pozíciót. Ennek a rendszernek a pontossága már

megfelelő, valamint egy alkalmas hardverrel a frissítési ráta is elégséges, azonban a rendszer kiépítése meghaladta az elvárt költségeket.

- UWB: korábbi ismereteink alapján felmerült az UWB (Ultra Wide Band) alapú eszközök használata. Ezek a beltéri működést már megvalósítják, azaz stabil jelet biztosítanak beltérben is, továbbá széles körben használják beltéri helymeghatározáshoz. Ilyen megoldások között található személy követés, mint az alábbi 2.1. ábra estén is, ahol a gép – amennyiben a személy beér a berendezés egy bizonyos környezetébe – egyre lassabb mozgást hajt végre, végül ha a személy a veszélyzónába ér, úgy a gép teljes mértékben megáll.



2.1. ábra: UWB használata személyek követésére [8]

A fent részletezett megoldás pontosságát tekintve nem volt alkalmas arra, hogy a személyeket és AGV-ket megfelelő pontossággal követni tudjuk. A legnagyobb probléma az volt, hogy az objektumok esetén a sebesség növekedése a pontosság további csökkenését eredményezte.

2.2 Marvelmind és eszközrendszere

Utolsó megoldásként merült fel ennek az eszköznek a neve, amely egy valós idejű beltéri helymeghatározó rendszer. Leírása szerint 2cm-es pontosságban képes beltérben a mozgó objektumok 3D-s koordinátáinak meghatározására. A választás során az előző fejezetben említett szempontokat figyelembe véve esett rá a választás, mivel a lehetséges megoldások közül ez rendelkezik a legnagyobb pontossággal, a bővíthetősége jobb, mint a kamera alapú rendszereknek, valamint nagyobb frissítési rátával is dolgozik.

A Marvelmind több eltérő eszközt használ, jeladókat, egy vevőt, valamint egy eszközt, mely képes összefogni a rendszert. Ezen felül találhatóak további segítő elemek is, amelyeket aszerint lehet csoportosítani, hogy a rendszerben jeladó, vagy pedig vevő szerepet töltenek be, valamint, hogy támogatja-e a kültéri használatot. Ezeket az alábbi 2.2. ábra mutatja be.

2.2. ábra: Super-Beacon (bal), modem (közép), Mini-Rx (jobb) [9]

Érdemes megemlíteni, hogy az egyes eszközökből létezik ipari kivitel is, amely IP67-es védelemmel rendelkezik, de a feladat során nem ezek, hanem a fenti ábrán látható hagyományos eszközök kerültek felhasználásra.

2.2.1 Super-Beacon

Az első eszköz, amelyről mindenképpen említést kell tenni, az a Super-Beacon, mely nem sorolható be egyértelműen adónak vagy vevőnek, mivel képességét tekintve mind a kettőre képes. Az, hogy ezek közül melyik kerül használatra az attól függ, hogy a rendszert milyen architektúrában használjuk, továbbá, hogy az eszköz milyen szerepet tölt be a megfigyelés során. Ennek egy összefoglalását tartalmazza az alábbi 2.1. táblázat, melynél az egyes oszlopok az architektúrákat (ld. Architektúrákfejezet), a sorok pedig azoknak a használatát jelenti.

2.1. táblázat: A Super-Beacon működése különböző helyzetekben [10]

	NIA architektúra	**IA architektúra**
Mozgó egység	Adó	Vevő
Fix pozíciójú egység	Vevő	Adó

2.2.2 Mini-Rx

Ahogy neve is jelzi, ez nem képes adatok küldésére, csak azok fogadására (Rx, receiver), cserébe viszont kisebb méretű, így akár egy emberen található láthatósági mellényre is felszerelhető. Egyetlen hátránya, hogy amennyiben ezt az eszközt szeretnénk használni, úgy az architektúrának olyan elven kell működnie, hogy a mozgó elemek

legyenek a fogadó eszközök. Ennek hátránya, hogy a későbbiekben, amennyiben járműveket is támogatni szeretnénk, abban az esetben ez a megoldás csak nehezen használható, mivel ilyen esetben a fogadó oldal lenne a mozgó járművön, mely zavarhatja a jelet. Amennyiben viszont a rendszer csak személyek követésére készül, akkor egy tökéletes megoldást tud biztosítani

2.2.3 Modem

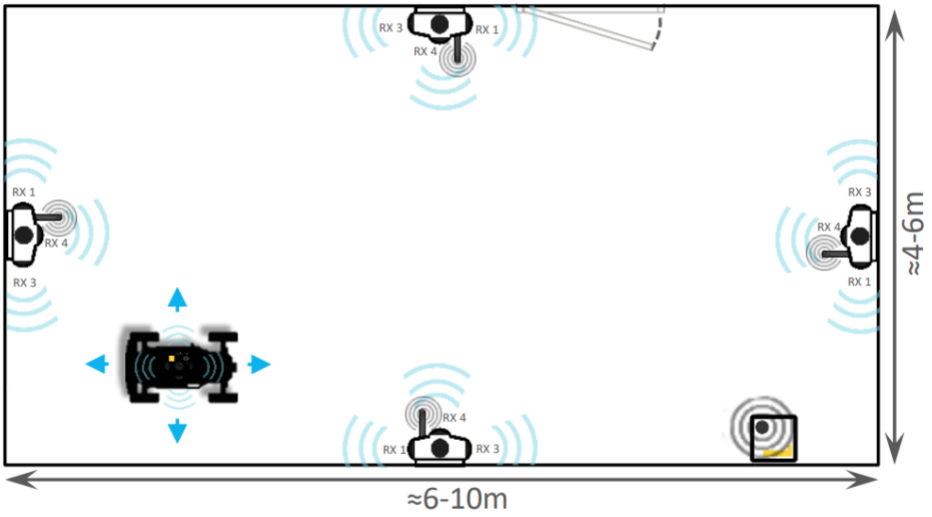
A működéshez szükséges még egy modem, amelynek szerepe, hogy a többi eszközt összefogja és ezen felül egy interfészt biztosítson, melyen keresztül a felhasználó képes a követendő objektumok pozícióját lekérni. Ez az interfész kerül felhasználásra a későbbiekben a Marvelmindból származó adatok kiolvasására, majd azok feldolgozására.

Ahhoz, hogy a modemből az adatok kiolvashatóak legyenek, ezt egy USB porton keresztül csatlakoztatni kell egy számítógéphez, ahol az adatok kiolvasása történik. Ezen felül ezen az eszközön keresztül kell a rendszert felkonfigurálni is, azaz itt lehet megadni, hogy mely elemek milyen pozícióban helyezkednek el, valamint a koordináta-rendszer elhelyezkedését is itt lehet módosítani.

Azonban azt megelőzően, hogy az eszköz használatra kerülhessen, fel kell telepíteni a megfelelő firmware-t az egyes elemekre. A firmware verziója esetén cél, hogy azonos legyen az összes eszközön, különben a kommunikáció nem épül fel közöttük. Ehhez első lépésként le kell tölteni a Marvelmind honlapjáról a legfrissebb, az eszközünkkel is kompatibilis verziót, majd azokat egy kiadott program segítségével frissíthetjük. Fontos, hogy a használt architektúra megválasztását is már itt el kell végezni, mivel a különböző változatok eltérő firmware-eket használnak és ezek lecserélése minden mentett adatot töröl az eszközökről.

2.2.4 Pozíció meghatározása

A Marvelmind [11] egy valós idejű beltéri helymeghatározó rendszer, mely működését tekintve a korábbiak ötvözése, de legközelebb a ToA (Time of Arrival) szenzorokhoz áll, hiszen ez is háromszögelést használ a pozíció meghatározásához. A jeladók esetén két elérő típusról beszélünk, ezek a „mobil” és a „stationary” jeladók (vagy beacon-ök). Míg az előbbi magán a mozgó tárgyon található (erre utal az elnevezése is), addig az utóbbiak a falra vagy egyéb fix pozíciókba vannak felszerelve, ezek a detektálás során nem mozognak. Ezen felül a modem lehelyezése is szükséges, ez a tényleges jeladótól legalább 2 méterre, de akár 100 m távolságra is lehelyezhető. Az alábbi 2.3. ábra esetén 4 stacionárius vevő található a négy falon egymással szemben, az 5. adó magán a távirányítós autón helyezkedik el, a modem pedig a jobb lenti sarokban.



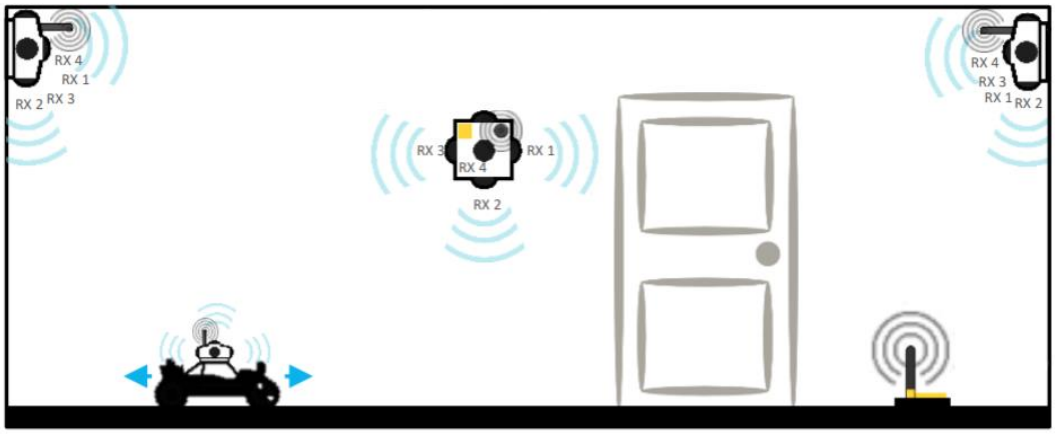
2.3. ábra Az eszközök egy lehetséges lehelyezése felülnézetben [11]

A jel sugárzása ultrahangon történik. Az egyes eszközök egy előre meghatározott frekvencián sugároznak, melyeket a további eszközök vesznek. Az egyes architektúrák és felépítések függvényében a sugárzott jel folyamatosan változhat, de akár a teljes működés során egységes is lehet. Továbbá az architektúra határozza meg azt is, hogy mely elemek fognak sugározni és melyek fogják fogadni a jelet.

Hogy a jelenlegi helyzetben meg tudjuk határozni az autó pozícióját, első lépésként meg kell mérni a jeladók egymástól való távolságát és így a saját koordinátájukat. A rendszer az alkalmazásában erre automatikusan képes, így ennek megvalósításával nem kell foglalkozni. Ez visszaad egy mátrixot melyben láthatóak az egymástól milliméterben vett távolságok. Ez a mérés pontatlanságából fakadóan folyamatosan változik, ezért célszerű a mozgó objektumok követését megelőzően ezeket „lefagyasztani”, azaz egy fix értéket megadni (ez megvalósítható az automatikusan kapott értékek mentésével is).

Következő lépésként már képesek vagyunk a követésre, amennyiben a távirányítós autón is bekapcsoljuk a jelsugárzót. Ebben az estben az érzékelés úgy történik, hogy a kiskocsin elhelyezett adó sugároz egy jelet, melyet a fix elhelyezésű vevők fogadnak. A jel kiadása és fogadása között eltelt időt pedig az egyes szenzorok mérik és ennek felhasználásával meg lehet határozni, hogy a háromszögben (több vevő esetén sokszögben) hol helyezkedik el a követendő objektum.

A fenti elhelyezési módszer főként 2D-s követésre alkalmas, mivel az egyes eszközök egy síkban találhatóak. Ennek a problémának a kiküszöbölésére célszerű egy beacont a többitől eltérő síkba helyezni. Erre egy példa a lenti 2.4. ábra, ahol a korábbi felülnézethez tartozó oldalnézetben látható, hogy az egyik beacon alacsonyabban helyezkedik el a többinél. Ezen kiegészítéssel a rendszer már alkalmas 3D-s követésre is és nem csak egy *X* és *Y* koordinátát kapunk vissza a rendszertől, hanem ez kiegészül egy *Z*-vel, ami a magasságot jelöli.



2.4. ábra Az eszközök egy lehetséges lehelyezése oldalnézetben [11]

A fenti leírás az alapértelmezett NIA (Non Inverse Architecture) koncepciót írja le, azonban az egyes architektúrák ettől eltérnek, amelyek a következő alfejezetben kerülnek bemutatásra.

2.2.5 Architektúrák

A rendszer működését tekintve három lehetséges opciónk van, melyeket a Marvelmind architektúráknak nevez [10]. Ezek a NIA (Non-Inverse Achitecture), az IA (Inverse Architecture) és az MF-NIA (Multi-Frequency NIA). Utóbbi a korábbi két esetnek a keveréke, mely ötvözi azok előnyeit, cserébe viszont lesznek paraméterek, amelyek rosszabb és lesznek, amelyek kicsit jobb értéket fognak felvenni. Ezeket a paramétereket a következő 2.2. táblázat foglalja össze.

2.2. táblázat Architektúrák összeghasonlítása [10]

	NIA	**IA**	**MF-NIA**
Frekvencia	Eszközök számával csökken	Fix	Eszközök számával csökken, lassabban, mint a NIA
Automata térkép	Igen	Igen	Igen

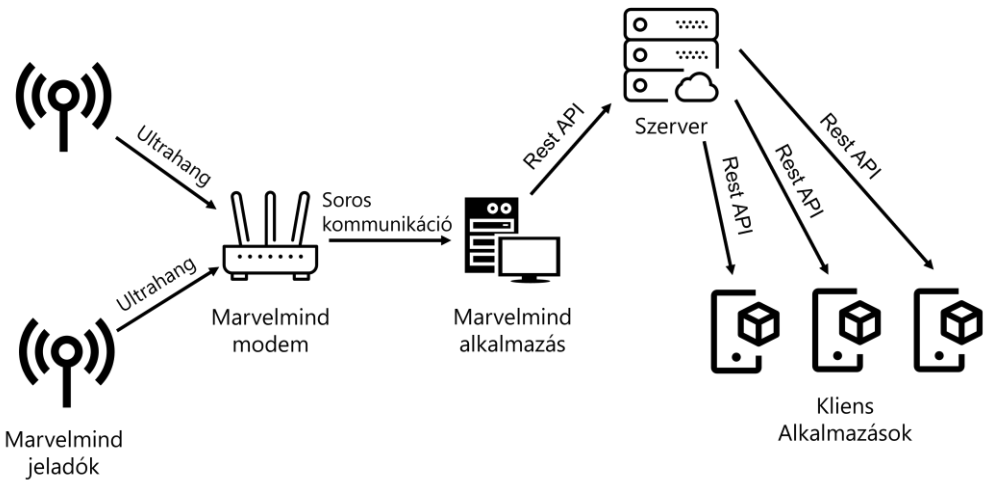
A **NIA felépítés** esetén a stacionárius jeladók nem sugároznak jelet, azok csak fogadják a mozgó adókból származó adatokat (ld. Pozíció meghatározása fejezet). Ennek a megoldásnak a hátrányában rejlik az előnye: a vevőknek lehetőleg zajmentes helyen kell elhelyezkednie, de ezáltal lehetővé válik, hogy a jel létrehozását és küldését tegyük a zajos járműre vagy drónra. Negatív tulajdonsága viszont, hogy ebben az esetben minél több eszközünk van egy területen, annál kisebb frissítési rátával dolgozunk. Ez abból ered, hogy az egyes mozgó egységek azonos frekvencián sugároznak, így meg kell határozni egy időablakot, melyben az egyik, valamint a másik elem sugároz.

IA struktúra esetén a rendszer fordítottan működik. Ebben az esetben a mozgó rész fogadja a jeleket, míg a fix pozíciójú részek küldik. Ennek egyik előnye, hogy maga a mozgó eszköz számolja ki a saját pozícióját, ezáltal a rendszer sokkal jobban skálázható több eszközre. Pici hátrány, hogy a fix pozíciójú adók gyorsabban merülnek, de pont az elhelyezkedésükből fakadóan ezek tápellátása sokkal könnyebben megvalósítható. Sajnos ez a szcenárió nem használható drónok és egyéb járművek követésére, mivel a fogadás oldalán a drón környezete zajos, így a mérési terület lecsökken a 30 méterről alig 5-10 méterre.

Harmadik architektúra a **MF-NIA**. Ebben az esetben a fentieket ötvözzük, azaz a számítást a modem végzi el, így kisebb a frekvenciája, mint az IA rendszereknek, azonban azokkal szemben az egyes fogadó eszközök több frekvencián is képesek fogadni. A küldő eszköz csak egyetlen egy frekvencián sugároz, de amennyiben 5, vagy annál kevesebb eszközünk van, akkor lehetséges, hogy mindegyik mobil beacon eltérő frekvencián sugározzon, így teljes mértékben ki lehet hagyni az időablakozást. Természetesen, amennyiben ennél több eszközt szeretnénk követni, úgy itt is megjelenik a NIA architektúra hátránya.

A tesztelés során csupán 6 beacon állt rendelkezésünkre, így annak eldöntése, hogy MF-NIA, vagy pedig NIA rendszert használunk, nem számított nagy mértékben. Emiatt az egyszerűség kedvéért utóbbit használtuk, mivel ennek kiépítése és módosítása is könnyebb, mint a másik esetben, és a tesztelési terület folyamatos változtatása miatt ez előnyösebb, mint egy komplex rendszer folyamatos átalakítása.

2.3 Felépítendő rendszer struktúrája



2.5. ábra: A rendszer felépítése

Az előző lépésben sikeresen kiválasztásra került egy beltéri helymeghatározó rendszer, melynek segítségével képesek voltunk az egyes szereplők mozgását követni. Ezt azonban meg is kellett valamilyen módon jeleníteni egy AR eszköz segítségével.

Ehhez azonban minden egyes felhasználó esetén szükség volt a pozícióadatokra, így ezt továbbítani kellett. Így a teljes folyamathoz szükségessé vált 3 további komponens. Ezek pedig a Marvelmindból érkező adatok feldolgozásáért felelős alkalmazás, a szerver komponens, mely ezeket az értékeket tárolja, valamint a kliens alkalmazások. A teljes rendszer felépítését a 2.5. ábra mutatja.

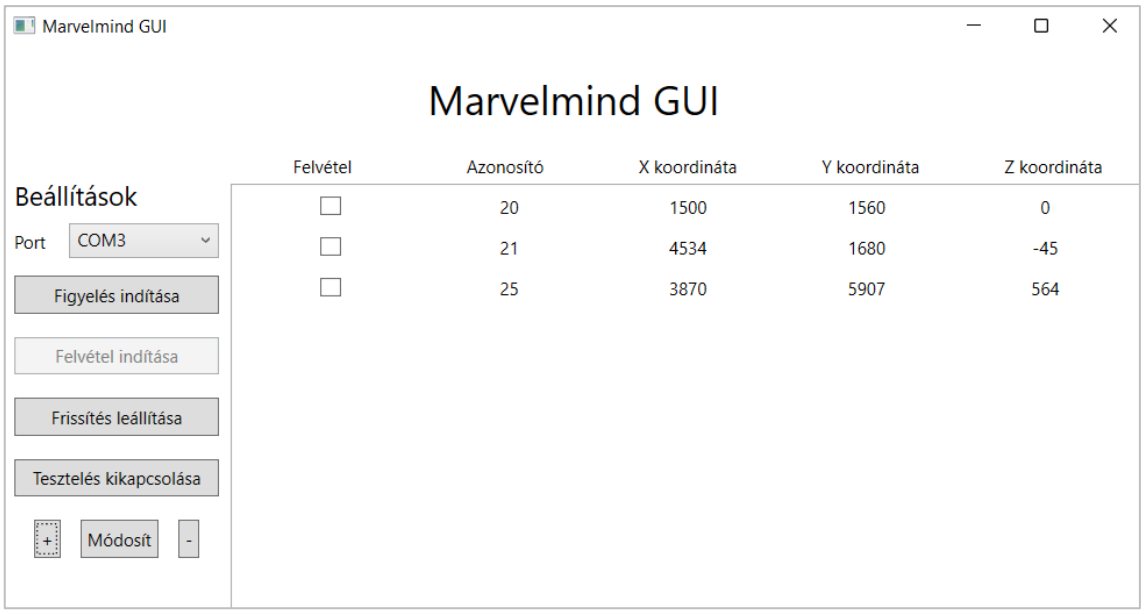
Az ábrán ezen felül látszódik, hogy az egyes komponensek között milyen módon zajlik a kommunikáció. Ezek közül a jeladók és a modem közötti kommunikáció a Marvelmind része, azzal nem kellett foglalkozni. Ezt követte, hogy soros porton ki kellett olvasni az adatokat, majd azokat egy Rest API segítségével tovább lehetett küldeni a szerver felé eltárolásra. Az egyes kliensek végül innen tudták kiolvasni szintén ezen API segítségével. Jelen fejezetben a komponensek közül bemutatásra került már a Marvelmind, a fejezet további részében pedig részletesebben be lesznek mutatva a szerver komponenssel bezárólag a még hátralévő részek. A kliens alkalmazás megvalósítása ezzel szemben a következő nagyobb fejezet témáját alkotja.

2.4 Alkalmazás fejlesztése az adatok rögzítéséhez

Ahhoz, hogy a Marvelmind eszköz használható legyen a jelen helyzetben, szükség volt egy olyan alkalmazásra, mely képes kiolvasni az eszközben található koordinátákat, azokat a megfelelő formátumra átalakítani és továbbítani egy szerver felé. Ennek megoldása alapvetően nem igényel grafikus felületet, azonban ahhoz, hogy a fejlesztés során a Marvelmind teszteléséhez is alkalmas legyen, egy WPF alkalmazás került megírásra. Ennek felületét mutatja az alábbi 2.6. ábra.

A fenti alkalmazás több eltérő részből állt, melyek a Marvelmindból származó adatok fogadása, azt követő feldolgozása, ezek megjelenítése a felületen és igény szerinti továbbítása a szerver felé, vagy annak mentése egy fájlba. Ehhez első lépésként csatlakozni kell a megfelelő COM porthoz Windows rendszeren, melyen keresztül a Marvelmind az adatokat továbbítja. Az alkalmazás magja – a grafikus felületet leszámítva – több platformmal is kompatibilis, így amennyiben szükségessé válik a fenti alkalmazás egy Linux szerveren való futtatása, akkor az is megvalósítható ezen alkalmazás alacsony mértékű módosításával.

A kommunikáció bináris formában történik, minden üzenet végén egy CRC16-os ellenőrző szakasszal, mely biztosítja azt, hogy a megkapott üzenetnek minden pontja helyesen érkezett meg. Amennyiben az ebből számított ellenőrzés elbukik, az üzenetet figyelmen kívül hagyja a rendszer. A protokoll során az üzeneteket a Marvelmind modemje folyamatosan írja a megfelelő portra, így nem kell kéréseket kiküldeni, elég csak a folyamatosan érkező üzenetek feldolgozásával foglalkozni.



2.6. ábra: A rögzítő alkalmazás grafikus felülete

Ezek az üzenetek sokfélék lehetnek, azonban az alkalmazás kezdeti szakaszában elég volt az alapvető funkciók megvalósítása. Ezek közé tartozik, hogy le tudjuk kérni az egyes mozgó jeladók koordinátáját, azok azonosítóját, melyeknek segítségével meg tudjuk határozni a követendő objektumok helyzetét. Ezen két adaton felül segítésként az egyes eszközök belső telemetriáját is átküldi a rendszer, mely tartalmazza, hogy az eszköz milyen irányba mekkora sebességgel halad vagy gyorsul, továbbá egy kvaterniót annak jelenlegi elforgásáról. Ez az utolsó paraméter akár alkalmas lehetett volna az egyes szereplők orientációjának meghatározásához, azonban a tesztelés során az volt a tapasztalat, hogy ezen értékek nem oly pontosak, mint vártuk volna, így a tényleges megvalósítás során nem kerültek felhasználásra.

Egy sikeres csatlakozást követően több opció közül választatunk, ezeket a bal oldali gombok szimbolizálják. Ennek első lépése, hogy lehetőségünk van elindítani a változások figyelését, azaz a korábbiakban részletezett kommunikáció feldolgozását. Amennyiben ezt nem kapcsoljuk be, akkor a folyamat nem kerül figyelésre és így az adatok sem frissülnek le. Annak érdekében, hogy az egyes eredmények nyomon követhetőek legyenek, a legutoljára kiolvasott adatok megjelennek a jobb oldali táblázatban. Ezek a sorok tartalmazzák az azonosítókat, valamint a hozzájuk tartozó 3D-s koordinátákat.

A fenti jobb oldali kiíráson kívül még három további gomb is található az alkalmazásban. Ezek közül az első, hogy egy vagy több kiválasztott jeladó koordinátáját minden egyes frissítés esetén kiírjuk egy CSV fájlba, hasonló struktúrában, mint a táblázatos megjelenítés. Ezen felül – hogy ezt össze lehessen vetni más forrásokkal – tartalmaz még egy időbélyeget a rendszeridővel, milliszekundum precizitással. A kiválasztás célja, hogy amennyiben több mozgó pont is van, viszont ezek közül csak egyet

(de legalábbis nem mindet) szeretnénk monitorozni, akkor erre is legyen lehetőség. A kiírás egészen addig történik, míg a felvételt le nem állítjuk (újra a 2. gombra kattintunk).

Abban az esetben, ha két pontot szeretnénk követni, például a 21-es és a 22-es azonosítóval ellátott eszközöket, akkor eredményként a lenti 2.3. táblázat szerinti adatokat kapjuk. Ebben egymást felváltva helyezkednek el az azonosítótok, tehát minden pillanatban kiolvassuk az összes mozgó pont koordinátáját, majd egy ideig semmit és újra a koordinátákat. Annak érdekében, hogy ez jól látszódjon utolsó oszlopban az idő esetén a pont utáni értékek milliszekundumok, ebből is látszódik, hogy 2 objektum követése esetén 4-5 Hz-es frissítési rátánk van.

2.3. táblázat: CSV fájlba kiírt adatokra példa

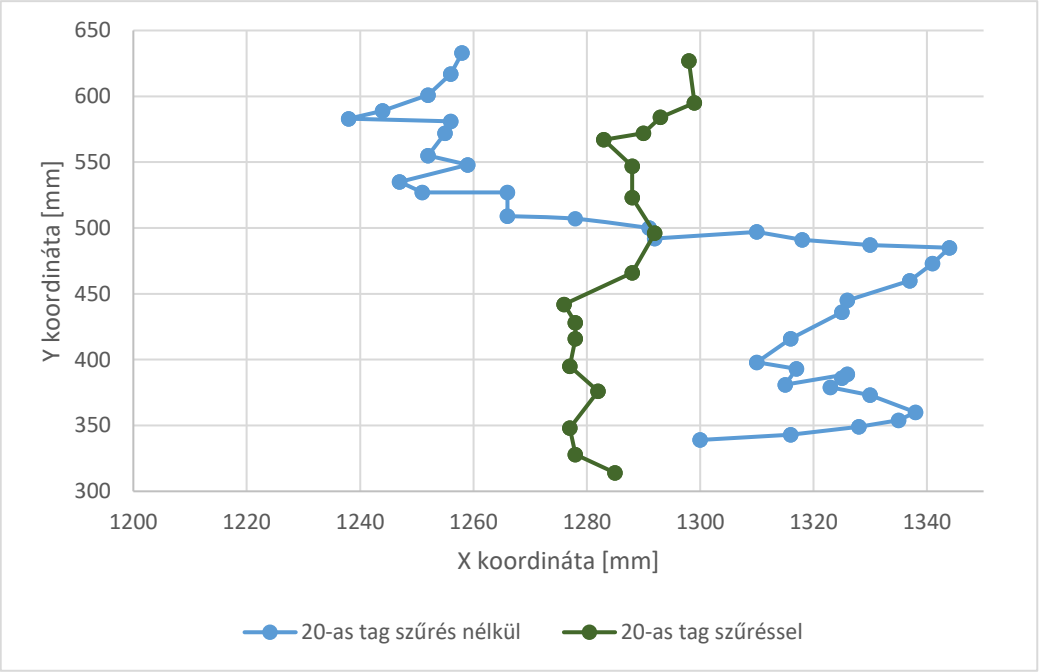
ID	X	Y	Z	Idő
21	2244	2899	1865	14:50:33.233
22	1462	1920	1217	14:50:33.234
21	2250	2908	1869	14:50:33.438
22	1468	1929	1219	14:50:33.439

Harmadik opció a tényleges adatfeltöltés indítását és leállítását segíti. Amennyiben a feltöltést elindítjuk, akkor a szoftver minden változást közöl a szerver felé (lásd: 2.5 Szerver megvalósítása). Ez a másik oldal által biztosított REST API-n keresztül történik, melynek segítségével a megfelelő tagek frissítésre kerülnek. Előfordulhat olyan, hogy egy jeladó valamilyen okból nem küld frissítéseket, ebben az esetben a megfelelő tag törlésre kerül a rendszerből. Amennyiben idővel visszajön a jeladó, úgy újra hozzá szükséges adni a fent lévő pontok közé.

Végezetül az utolsó opció egy teszt üzem bekapcsolása. Ebben az esetben a jeladónak nem kell ténylegesen a helyszínen lenni, szimulálható az, hogy milyen értékeket olvasna ki a rendszer. Ennek szerepe, hogy a tesztelés során a pozíció szabadon módosítható legyen, ne legyenek korlátozó tényezők. Ilyen lehet, hogy amennyiben ki szeretnénk próbálni, hogy mi történik akkor, amennyiben tőlünk 150 méterrel arrébb lévő pozíciót küld vissza a rendszer, akkor az így megtehetjük, szemben a tényleges fizikai rendszerrel, ahol 150 méteres terület lefedéséhez a rendelkezésre álló jeladók száma nem elegendő.

Ezen megoldás esetén a 2.6. ábra bal oldalán található utolsó gombsor megjelenik – ez alap esetben rejtett – és ezzel biztosít lehetőséget a három eltérő feladathoz, melyek az új érzékelő pont hozzáadása, egy már hozzáadott elem törlése, vagy pedig egy jelenleg is aktív elem módosítása. Ezek célja megfeleltethetőek valós eseményeknek. Az első opció biztosítja a lehetőséget, hogy a felhasználó belép a megfigyelési területre és onnantól kezdve lesz követve a pozíciója valós időben. A második eset szimulálja, amennyiben a megfigyel szereplő elhagyja a vizsgált tartományt, így azt el kell távolítani a követendő

karakterek közül. A harmadik lehetőség biztosítja annak kipróbálását, hogy mi történik akkor, amennyiben az elemzett mezőn belül mozog.



2.7. ábra: A szűréssel és szűrés nélkül mért koordináták (4,5 másodperc alatt)

A megvalósítás során azonban megfigyelhető volt, hogy a Marvelmind tagek érzékelése során erőteljes zaj jelentkezett. Ez álló helyzet esetén még nem jelentene problémát, azonban amennyiben elkezdjük mozgatnia egy egyenes mentén a taget, akkor a fenti 2.7. ábra esetén is látható oldalirányú zaj jelentkezik. Ez egy átlagos pozíció követése esetén nem lenne probléma, azonban amennyiben az alkalmazásban fontos, hogy a pozíció nagyon pontos legyen, úgy ez a 10 centiméteres ugrálás sem megengedhető. Ennek kiküszöbölésére a feldolgozást követendően egy szűrés került implementálásra az alkalmazásban, mely az előző 8 értéket vizsgálja. A folyamat során ezen 8 értéket sorba rendezi, majd az így kapott új listának az alsó és felső 25%-át eltávolítja. Így a maradék értékekből el lett távolítva a zaj egy kis része. Ezt követően a középső rész átlagát számolja és ez adja az új pozíciót.

Ennek implementálásával elkészült egy alkalmazás mely képes a kiválasztott beltéri helymeghatározóból származó adatok fogadására, azon egyszerűbb műveletekkel való szűrésére és az így kapott eredmények szerver felé való továbbítására. A fejezet elején ezen felül bemutatásra került néhány további lehetséges helymeghatározó rendszer is, melyek azonban nem feleltek meg a feléjük támasztott követelményeknek. A következő alfejezetben az adatokat tároló szerver kerül bemutatásra, valamint az, hogy azok milyen struktúrát követnek.

2.5 Szerver megvalósítása

Az implementálandó rendszer második összetevője egy szerver alkalmazás. Ennek szerepe, hogy a beltéri helymeghatározó rendszerből származó adatokat egy központi rendszeren tárolja, ahonnan a kliensalkalmazások le tudják kérni. Ennek oka, hogy a kliensek a felhasználóknál foglalnak helyet, továbbá változó számú eszköz kerülhet felhasználásra, így biztosítani kell egy lehetőséget, mellyel egy bővülő eszközkészlet esetén sem kell a megoldáshoz a program módosítása. Ennek hátránya azonban, hogy ezzel a tényleges pozícióadat megjelenéséhez képest a kliens késleltetéssel fog a jelenlegi pozícióhoz jutni. Ez ASP.NET-et felhasználva C# nyelven került megvalósításra. A szerver egy REST API-t biztosít az adatok megfelelő irányú módosítására.

Első lépésként annak eldöntése volt a feladat, hogy az egyes adatok milyen formában kerülnek eltárolásra. Ehhez lehetséges felépíteni egy SQL adatbázist, mely képes a folyamatosan változó adatok tárolására, azonban ennek létrehozása és módosítása komplexebb feladat, mint amennyit a szerver komplexitása elvárna magától. Így az tárolást egy NoSQL adatbázis szolgáltatja, melynek segítségével a rendszer képes egy *dll* fájlformátumban az adatok titkosított tárolására és abból való visszaolvasására. Ez a kiválasztott adatbázis-kezelő a LiteDB [12]. Ennek egy rendes adatbázis szerver felépítésével szemben több előnye is van, melyek a következők:

- nem igényel kiszolgálószervert a működése során,
- egy kis méretű fájlban tárolja az adatokat, így ezzel a szerver mérete és komplexitása jelentősen csökkenthető,
- a megvalósítás C#-ban íródott, így támogatja a legújabb .NET verziókat is,
- LINQ támogatással rendelkezik, melynek hatására a lekérdezések és módosítások megvalósítása jelentősen megkönnyíthető,
- a NoSQL adatbázis szerkezet lehetővé teszi az adatok hierarchikus egymásba ágyazott tárolását, amellyel könnyebben átláthatóvá válik az adatbázis

Ezt követő lépésként annak eldöntése következett, hogy milyen funkciók biztosítását kell tartalmazni a rendszernek. Ezek közül 4 következett egyértelműen az előzőekben bemutatott folyamatokból, melyek az alábbiak. Első, egy új azonosító hozzáadása, mely abban az esetben történik, amikor egy még nem követett személy belép a területre, ahol a felismerés történik. Ennek ellentéte, hogy amennyiben egy felhasználó elhagyja ezt a zónát, akkor a hozzá kapcsolt pozícióadatoknak el kell tűnniük a rendszerből, hogy később a kliens amikor ezeket az értékeket lekéri, akkor ne érkezzen a számára olyan információ, amelynek hatására egy nem létező egyént is meg szeretne jeleníteni. Ezen felül felmerült, hogy a követés során ezen azonosítókhoz tartozó érzékeléseknek valamilyen módon megváltoztathatónak kellett lenniük. Ezen felül a kliens oldaláról biztosítani kellett egy interfészt, melyen keresztül ezeket az adatokat le tudja kérni a

felhasználó. Ehhez meg kellett határozni egy adatstruktúrát, amelyben ez eltárolható, valamint visszaadható és azt követően meg kellett valósítani a hozzá tartozó kéréseket, valamint a hozzájuk köthető adatbázis módosításokat.

2.5.1 Adatok struktúrája

Ehhez első lépésként meg kellett határozni, hogy a helyzetmeghatározó rendszerből az adatok milyen struktúrában kerülnek kiolvasásra és azokat milyen formában érdemes eltárolni. Meg kellett tervezni, hogy milyen további paraméterekre van szüksége a rendszernek ahhoz, hogy az egyes karakterekhez a név megjeleníthető legyen, valamint azt, hogy ezeknek az adatoknak a metszetét milyen formában célszerű visszaküldeni a kliens felé, hogy az a legkevesebb módosítással képes legyen az értékek megjelenítésére.

Marvelmind adatok

A Marvelmindból érkező adatok lekéréséhez és tárolásához meg kellett vizsgálni, hogy milyen struktúrát használ a Marvelmind és azt hogyan lehetne leképezni a szerver felé. Ennek felépítése azonban megegyezett az elvárttal, azaz minden egyes tag esetén négy érték kerül elküldésre. Ezek közül az első azonosítja a taget. Ez megfelel a tag ID-jának, mely még a rendszer kiépítése esetén megadható az egyes jeladóknak. Ezt követően három egész szám kerül továbbításra, mely megfelel az *X*, az *Y* és a *Z* koordinátának. Az adatbázis struktúrájából kifolyólag egy JSON alapú tárolást valósít meg, így a fenti adatokat az alábbi adatstruktúra tartalmazza:

```
{
    "id": 17,
    "position": {
        "x": 1452,
        "y": 1325,
        "z": 152
    }
}
```

A fenti struktúra esetén látható, hogy a pozíció nem a tényleges tag objektumban kerül eltárolásra, hanem annak egy alobjektumában. Ennek célja, hogy így amennyiben a pozíció valamilyen plusz értékkel a későbbiekben kiegészítésre kerülne, akkor az egy szinttel lejjebb található, így a fő struktúra nem változna. Másik előnye, hogy így a programkódban a fenti megvalósítás könnyebben értelmezhető, mivel így egy tag rendelkezik egy ID-val és egy pozícióval, ami lehet egy 3D-s vektor, melyből egyértelmű, hogy melyik elem mit szeretne jelenteni. Ez utóbbi az adatokat mm-ben tárolja.

Kiegészítő adatok

Ezen felül, azonban kellett egy másik struktúra, mely a kapott azonosítójú elemekhez hozzárendeli, hogy mely karakterhez tartozik és azt, hogy a megjelenítéshez milyen további információk állnak még rendelkezésre. Ezek közül az első érték, hogy amennyiben a megjelenítést végezzük, akkor valamilyen módon az egyes karakterekre

hivatkozni kell. Ez több szempontból is fontos, melyek közül az első, hogy amennyiben a pozíciója változik egy karakternek, akkor az nem a Marvelmind azonosítót fogja nézni, hanem ezt az értéket. Így amennyiben a későbbiekben szeretnénk az egyes tageket újra felhasználni egy másik területen belül, akkor az a struktúrában – akár működés közben is – egy egyszerű módosítással megváltoztatható.

A második és egyben fontosabb szerepe, hogy egy személyhez nem csupán egyetlen tag köthető. A fejlesztés során előjött, hogy szeretnénk követni az egyes karakterek pozícióján felül azoknak az orientációját is. Ehhez az került felhasználásra, hogy amennyiben 2 tag lenne a megfigyelt egyénen, akkor a két elem különbségéből meg lehetne állapítani azt, hogy mely *XY* irányba néz a karakter. Ehhez azonban kell egy azonosító, mellyel egy szereplőt abban az esetben is lehet azonosítani, amennyiben több eltérő tag is tartozik hozzá. Ennek kiegészítése a „location" elem is, mely azt határozza meg, hogy az éppen elmetett tag az a pozíció értelmezéséhez (0 esetén), vagy pedig az orientáció meghatározásához (1 esetén) van felhasználva. A további adatok minden esetben a korábbiból kerülnek kiírásra, mivel olyan eset előfordulhat, hogy orientáció nincs, de lokációval abban az esetben is rendelkeznünk kell.

Ezen felül további 4 eltérő paraméterrel rendelkezünk. Ezek közül az első a típus, mely jelen pillanatban minden esetben 0. Ez egy enumeráció, mely jelzi, hogy a tag milyen objektumot követ. Jelenleg a rendszer csak személyek vizsgálatára alkalmas, melyet a 0 érték jelöl, viszont a későbbiekben az alkalmazás továbbfejleszthető, hogy akár targoncákat, továbbá akár drónokat is képes legyen megfigyelni. Ennek az értéknek a megváltoztatásával módosítható, hogy az elmentett karakter milyen formában kerüljön a kliens alkalmazásban megjelenítésre. Ezen felül tartozik a személyhez még egy név is, mely a személy felett megjelenő azonosítót jelenti. Ez egy egyén esetén a neve is lehet, azonban egy drón esetén egy mindenki számára érthető leírás kell, hogy kiválasztásra kerüljön. Ahhoz, hogy ez pontosan mindig a karakter felett jelenjen meg bizonyos magassággal, így meg kell adni, hogy a személy pontosan milyen magas. A megjelenítést végző alkalmazás ehhez ad hozzá még egy bizonyos távolságot. Ezen felül még egy háromdimenziós vektorral is rendelkezik ez a leírás. Erről részletesebben a További funkciók fejezetben lesz szó.

```
{
    "id": 17,
    "userId": "idstring",
    "type": 0,
    "location": 1,
    "name": "Name string",
    "boundingBox": {
        "x": 150,
        "y": 150,
        "z": 0
    },
    "height": 1700
}
```

A fenti struktúra hasonlóan a pozícióhoz JSON formátummal rendelkezik. Ebben az esetben minden típusú eszközt egy azonos felépítéssel definiálunk. A későbbiekben, amennyiben előfordulna, hogy az egyes járművekhez további adatokra is szükség lenne, akkor lehetséges, hogy célszerűbb lenne az egyes típusokhoz külön struktúrákat definiálni, szemben a jelenlegi megoldással. Ezzel azonban a lekérés és a tárolás is bonyolódna annyival, hogy a jelenlegi megoldás esetén ennek implementálása nem volt megtérülendő. Látható, hogy a befoglaló téglatest azonos struktúrával épül fel mint az érkező adatok esetén a pozíció, ennek oka, hogy itt is a fent már részletezett 3D-s vektor került felhasználásra.

Leküldött adatok

A harmadik struktúra, ami létrehozásra került az alkalmazásban, az nem más, mint a kliens irányába elküldött adatstruktúra. Ez a fenti két elemnek az összetételéből jött létre. Az alapvető adatokat, mint a felhasználó azonosítója, neve, magassága, típusa és a befoglaló téglatest oldalainak mérete, az előző adatstruktúra tartalmazta, melyek egy az egyben átmásolásra kerültek ebbe a struktúrába. Ezen kívül még két további kétdimenziós vektor kerül elküldésre, melyek közül az első (front tag) határozza meg a pozícióját a megfigyelt objektumnak, a második pedig az orientáció kiszámítása során kerül használatra (back tag). Amennyiben a korábbi null értéket vesz fel, abban az esetben nincs az adott azonosítóra észlelés, mely esetben nem fog megjelenni a megfigyelt szereplő. Amennyiben az utolsóra nincs érték, azaz a back tag vesz fel null értéket, akkor a személyt jelző virtuális karakter meg lesz jelenítve, de nem fog hozzá forgásérték tartozni. Ebben az esetben a virtuális karakter a megfigyelő személy felé fog nézni.

```
{
    "userId": "idstring",
    "name": "Name string",
    "height": 1700,
    "type": 0,
    "boundingBox": {
        "x": 150,
        "y": 150,
        "z": 0
    },
    "frontTag": {
        "x": 150,
        "y": 160
    },
    "backTag": {
        "x": 170,
        "y": 180
    }
}
```

A szerver a fenti 3 struktúrát használja, melyek közül csupán a második kerül elmentésre az adatbázisba. Ennek oka, hogy amennyiben a rendszer újraindulna, akkor az egyes személyekhez tartozó adatokat így nem kell, hogy újra megadják. Azonban a Marvelmind adatokat nem szükséges elmenteni, mivel ez feleslegesen terhelné az

adatbázist, valamint amennyiben újra indulna a rendszer, akkor az azonnal elküldi az érzékelt objektumokat, szóval nincs olyan érték, melyre az indulás pillanatában szükség lenne.

2.5.2 Elérés módja

Annak érdekében, hogy ezeket az értékeket a rendszer módosítani tudja több elérési pontot is biztosít a rendszer. Ehhez a szabványos http megoldások kerülnek felhasználásra, azaz egy-egy végpont esetén a GET, PUT, POST és DELETE parancsok. A megoldás két eltérő részre bontható, melyek közül az első a helymeghatározó rendszerből érkező adatok frissítésére és a kliensben való lekérésre biztosít megoldásokat, míg utóbbi az egyes tagekhez tartozó információk eltárolásához és módosításához.

Tag információk elérése

A kettő közül a taghez rendelt információk elérése a komplexebb folyamat, bár ez is az alapvető megoldásokat tartalmazza. Ezt foglalja össze a lenti 2.4. táblázat. Ahogyan a táblázat is mutatja két eltérő lehetőségünk van lekérni az információkat. Ezek közül az első, amennyiben minden elemre szükségünk van, a második pedig, amennyiben arra vagyunk kíváncsiak, hogy egy adott Marvelmind taghez létezik-e már hozzáadott leírás. Ezek amennyiben tartalmaznak adatokat, akkor a korábban részletezett struktúrát adja vissza, vagy egy üres elemet, amennyiben még nincs ilyen információ.

2.4. táblázat: A tag információk végpontjai

Végpont	Típus	Válasz
api/taginfo	GET	200, 400
apit/taginfo/<id>	GET	200, 400
api/taginfo/<id>	DELETE	200, 400
api/taginfo	PUT	200, 400

Ezen felül két további elérhetőség van biztosítva ezen információk módosításához. Ezek közül a korábbi a törlés, melynek segítségével egy Marvelmind azonosítóhoz lehetőségünk van törölni minden adatot. Ez amennyiben sikeresen törlésre került bármi, akkor a http válaszok közül a 200-ast, azaz a rendben választ küldi vissza. Abban az esetben is ez történik, amennyiben már nincs milyen értéket törölni, hiszen annak eredménye azonos, hogy az adatbázisba ahhoz a taghez nem fognak további adatok tartozni. A másik elem pedig egy PUT. Ennek lényege, hogy a fenti struktúra itt a szervernek kerül elküldésre és amennyiben az még nem található meg az adatbázisban, akkor hozzáadásra kerül, amennyiben viszont már adott egy elem az elküldött azonosító mellett, akkor ahelyett, hogy hibát dobna egyszerűen lefrissíti az ott található adatokat az elküldöttel. Ez az oka annak, hogy az üzenetküldés típusa nem POST, hanem PUT, még abban az esetben is, amennyiben az elején csak hozzáadunk.

A fenti táblázatban látható, hogy amennyiben valamilyen hiba keletkezik, akkor egy összefoglaló hiba kerül visszaküldésre. Ez a 400-as http kód, mely a hibás kérést jelöli. Ennek oka, hogy az esetek többségében nincs specifikus hiba, hiszen nem kerül vizsgálatra, hogy az adatbázis tartalmazza-e már az adott értéket, hanem amennyiben igen, akkor az felülírásra kerül. Emiatt az egyetlen hibalehetőség az, amennyiben az adatbázis elérése során keletkezik valami hiba.

Tag értékek elérése

A másik csoport a tag tényleges értékeinek megváltoztatására jött létre. Ennek két szerepe van, melyek közül az első, hogy ezeket a tag értékeket le lehessen kérni. Itt az előzőekhez hasonlóan két eltérő típus került megírásra a lekérésből is, melyek közül az első minden érték megszerzése, az utóbbi pedig egy bizonyos felhasználói azonosítóhoz tartozó információk lekérése. Ezeknek a struktúrája az utolsó részletezett felépítéssel egyezik meg. Amennyiben egy taghez nincs információ elmentve, akkor egy üres lista kerül visszaküldésre. Ezek a struktúrák nincsenek elmentve, minden esetben a jelenlegi adatokat felhasználva a kérés pillanatában generálódik.

2.5. táblázat: A tag módosításának végpontjai

Végpont	Típus	Válasz
api/tag	GET	200, 400
api/tag/<id>	GET	200, 400
api/tag	PUT	200, 400

Az utolsó lehetséges megoldás a tagek pozícióit frissíti le. Ebben az esetben egy minden értéket tartalmazó lista kerül felküldésre. Minden lépés elején minden jelenlegi azonosítót és értéket törlünk, majd amennyiben az új struktúrában ez szerepel, akkor az új módosított értékkel hozzáadjuk. Ennek hátránya, hogy ameddig ez a folyamat tart, addig a szerverről való lekérés is szünetel, mivel ellenkező esetben inkonzisztens adatok kerülnének leküldésre.

Az elmúlt fejezetben kiválasztásra került a beltéri helymeghatározó, melynek első lépésként bemutatásra kerültek a lehetséges opciók, majd azokból igény szerint eldöntésre került a jelenlegi helyzetben legmegfelelőbb eszközkészlet. Ezt követően a kiválasztott rendszer részletesebb vizsgálat alá került. Ezt követte a hozzá tartozó alkalmazás és szerver elkészítése, melynek segítségével az adatok továbbküldhetőek az egyes kliensekre. A következő fejezet témája pedig a kliens alkalmazás megvalósítása, melynek kiindulási lépése, hogy ki kell választani egy olyan platformot, melyen ez az alkalmazás megvalósítható.

3 Kliens alkalmazás

A teljes szoftver utolsó komponense a kliens alkalmazás, amely a tényleges megjelenítést végzi a felhasználók felé. Ennek segítségével lesz a helymeghatározó rendszer területén belül dolgozó alkalmazottaknak lehetősége hogy a folyamatosan mért pozíciót valamilyen módon láthassák és felhasználhassák. Ennek megjelenítése a kiterjesztett valóság segítségével történik.

Ez az a komponens, ahol a korábbiakban már bemutatott részek – a valós pozíció adatokat feldolgozó szoftver és a hozzá tartozó szerver – által létrehozott adatok az azon elvégzett módosításokkal együtt megjelennek. Azonban ehhez szükség van valamilyen eszközre mellyel a kiterjesztett valóság alkalmazás ténylegesen rálát a fizikai környezetre és azt a megfelelő módon kiegészíti.

Az alkalmazás egy rendes ipari környezetben egy okos szemüveget használna, mint például a Microsoft HoloLens 2-t, azonban ennek hiányában a fejlesztés és tesztelés időszakára egy olyan eszközt kellett választanom, mellyel rendelkeztem, viszont mégis képes alapvető módon AR-re. Így végezetül a tesztelési eszköz a saját mobiltelefonom lett, mely a következő kamerákkal rendelkezett:

- Alap hátsó kamera: 48 MP
- Széles látószögű kamera 12 MP
- Makró kamera: 5 MP
- Monokróm: 2 MP

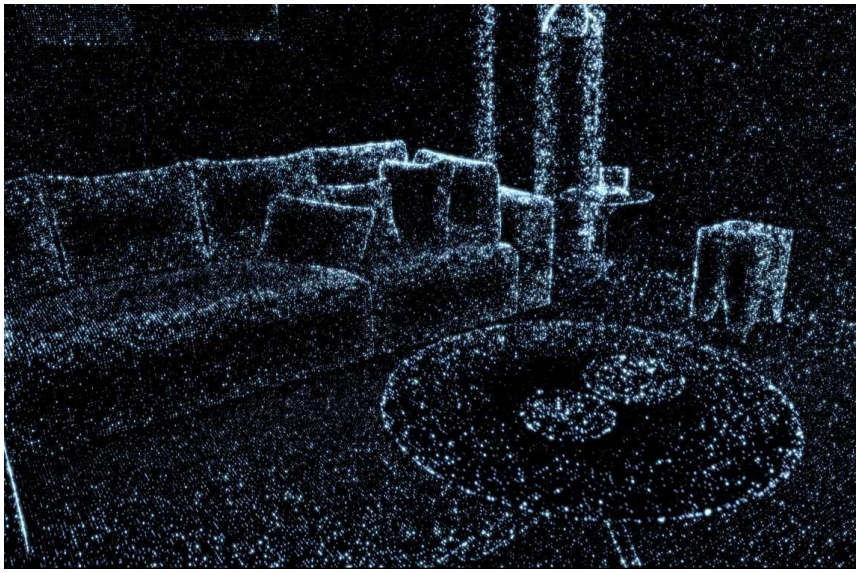
Ezek közül az AR alkalmazás esetén fontosabb szerepet az alap hátsó kamera játszott. Ennek célja egy videó folyamatos felvétele, melyen az alkalmazás különböző műveleteket el tud majd végezni és ezzel lehetősége nyílik arra, hogy a valós környezetben szimulált 3D-s objektumokat jelenítsen meg. Így hogy a fejlesztési időre egy eszköz már rendelkezésre állt, nem volt más feladat hátra, mint annak kiválasztása, hogy mely platform kerüljön felhasználásra az alkalmazás elkészítése során.

3.1 Lehetséges platformok

Az AR alkalmazások fejlesztésére több eltérő platform is elérhető [13], melyek többsége rendelkezik alapvető képességekkel, mint a kamera kép kirajzolása a mobiltelefon képernyőjére, síkdetektálás, azonban a platformok között vannak eltérések, vannak olyan megoldások, melyeket míg az egyik teljes egészében támogat, addig egy másik esetén nem rendelkezünk ezzel a képességgel. Emiatt 4 eltérő platform került megvizsgálásra, hogy melyik lenne-e alkalmas az alkalmazás elkészítésére.

3.1.1 ARKit

A kiterjesztett valóság keretrendszerek közül az első az ARKit [14], melyet az Apple fejlesztett ki és frissít. Működését tekintve kamera kép alapú odometriát használ, kiegészítve az eszköz gyorsulásmérőjéből és giroszkópjából származó adatokkal, hogy azzal pontosabbá tegye a jelenlegi lokáció meghatározását. Az újabb Apple eszközök esetén a beépített LiDAR szenzor segítségégével pontosabb képet kaphatunk a tárgyak kamerától vett távolságáról. Egy ilyen képet mutat a 3.1. ábra.



3.1. ábra: iOS LiDAR által detektált kép

Ezen felül támogatja a platform a kéz követését is, valamint támogatva vannak az alapvetőbb eszközök is, mint a síkdetektálás, valamint a szobában tapasztalható fények kiszámítása, mellyel a virtuális tárgyak megvilágítását lehetséges feljavítani. Az egyik előnye és egyben legnagyobb hátránya a megoldásnak, hogy csupán az Apple termékein képes futni, mert így azokat teljes mértékben ki tudja használni, azonban egy szélesebb körben is használni kívánt megoldást nem lehetséges vele megoldani.

3.1.2 ArCore

Egy másik megvalósítás a Google által fejlesztett ARCore [15] kiterjesztett valóság platform. Ebben az esetben is kamera kép alapú odometriát használ a rendszer, mely működése során a kamera képen jellegzetes pontokat detektál és azok elmozdulásából képes a kamera elmozdulására következtetni. Ezen felül szintén a korábbihoz hasonlóan itt is kiegészítésre kerül az eszközben található szenzorokból származó adatokkal a pozíció. A Google keretrendszere is képes a szobában tapasztalható fényviszonyok becslésére, valamint a megfigyelt területen a síkfelületek meghatározására.

Ezen felül itt is lehetséges a kamerától való távolság meghatározására, azonban itt nem egy LiDAR került felhasználásra, hanem a telefonba beépített fő kamera alapján kerül meghatározásra egy mélységtérkép. A technológia különbsége miatt ez utóbbi

kevésbé hatékony mint a korábban részletezett LiDAR-os megoldás, cserébe viszont nincs szükség egy további szenzor beépítésére. Az ArCore-ban megvalósított alkalmazás minden az elmúlt néhány évben kiadott Androidos telefonon elfut, valamint képes iOS-t futtató eszközökön is működni.

3.1.3 Vuforia

Harmadik megoldás a Vuforia [16], melyet a PTC vállalat fejleszt. Hasonlóan az előzőekhez ez is képes az saját pozíció, a síkok és a megvilágítás meghatározásra. Ezt kiegészítve a Vuforia segítségével lehetséges például egy képhez kötni a 3D-s objektumot, így amikor ránézünk egy újság egy oldalára, akkor az ott látható kép felett megjelenne 3D-s társa. Erre mutat egy példát a 3.2. ábra. Jelenleg a fenti két megoldáson kívül az egyik legelterjedtebb. Előnye, hogy mobileszközökön kívül képes a Windows-os eszközökön való futásra is, így használható vele a HoloLens.

3.2. ábra: 3D-s objektum megjelenítése egy 2D-s képen a Vuforia használatával

3.1.4 AR Foundation

Negyedik, egyben az utolsó vizsgált platform az AR Foundation [17], mely a Unity keretein belül fut és itt lehet alkalmazást készíteni a felhasználásával. Ez az eszközök támogatottságában azonos a Vuforiaval, azonban ahelyett, hogy újra megvalósítaná a korábbiakban már többször részletezett megoldásokat, a már elkészített eszköz specifikus platformját használja. Azaz amennyiben az eszköz, melyen az alkalmazás futna, egy Androidos készülék, úgy az AR Foundation az ARCore megoldásait fogja használni, iOS esetén pedig az ArKitet. Azonban ahelyett, hogy az alkalmazást kétszer kéne elkészíteni, egy egységes felületen keresztül mind a két másik platform elérhető, melynek segítségével az fejlesztési idő csökkenthető.

Ezt követően a platform eldöntése következett, melynek kiválasztása esetén az alábbi pontok kerültek figyelembevételre:

- A tesztelési és fejlesztési eszköz egy Androidos készülék

- Később célszerűen az alkalmazás egy okosszemüveg eszközön is kell hogy fusson, így több platform támogatása szükséges

Emiatt végezetül az Ar Foundation mellett esett a döntés, melynek segítségével így Unity-n belül lehetett elkészíteni az alkalmazást, miközben a támogatott eszközök között megjelenik mind az Androidos telefonok, mind a HoloLens 2, mind pedig a Magic Leap eszközök. Ezt követően megkezdődhetett az alkalmazás tényleges elkészítése.

3.2 Alkalmazás implementálása a kiválasztott platformon

Az alkalmazás elkészítése, ahogyan a korábbiakban már említésre került, Unityben történt az AR Foundation és az ARCore felhasználásának segítségével. Ennek első lépése egy olyan projekt létrehozása volt, mely támogatja a kiterjesztett valóságot. Ennek megléte utáni első feladat, hogy meg kellett jeleníteni a teszteléshez, majd a későbbiekben a kitakarás esetén egy karaktert, mely azt ábrázolja, hogy az ember hol lenne. Ezt a karaktert mutatja a 3.3. ábra.

3.3. ábra: A virtuálisan kirajzolt karakter a területen belül

Ehhez meg kellett határozni, hogy a területen belül hol is helyezkedik el az origó. Mivel AR esetén a kamera kiindulási pontja az indítás pillanatában a kamera helye, így ezt valamilyen módon egységesíteni kellett a Marvelmindból érkező X és Y koordinátákkal. Erre azért volt szükség mivel a legtöbb esetben nem pontosan a beltéri

helymeghatározó origójában állunk az indulás pillanatában, hanem a területen belül valahol. Ekkor mind a kamerának van egy koordináta-rendszere, melynek kiindulópontját az AR eszköz pozíciója és orientációja határozza meg, mind pedig a beltéri helymeghatározónak, mely a kiépítés során egy előre meghatározott pontnak és szögnek felel meg.

3.2.1 Közös origó megadása

Ezt kiküszöbölendő a megvalósítandó program azzal egészült ki, hogy a megfigyelt területen elhelyezünk olyan képeket, vagy markereket, melyeknek a beltéri helymeghatározó által megadott pozícióját ismerjük. Egy ilyen lokáció esetén mind a három koordinátára – mellyel az eltolást végezzük – valamint az elhelyezését leíró forgásokra is szükség van. Annak érdekében, hogy ennek a 6 értéknek a megadása ne legyen nehéz, így ezeket a fix pontokat egy valós környezetben célszerű a falra helyezni, ami miatt a három forgást reprezentáló szög közül egy mindig azonos lesz.

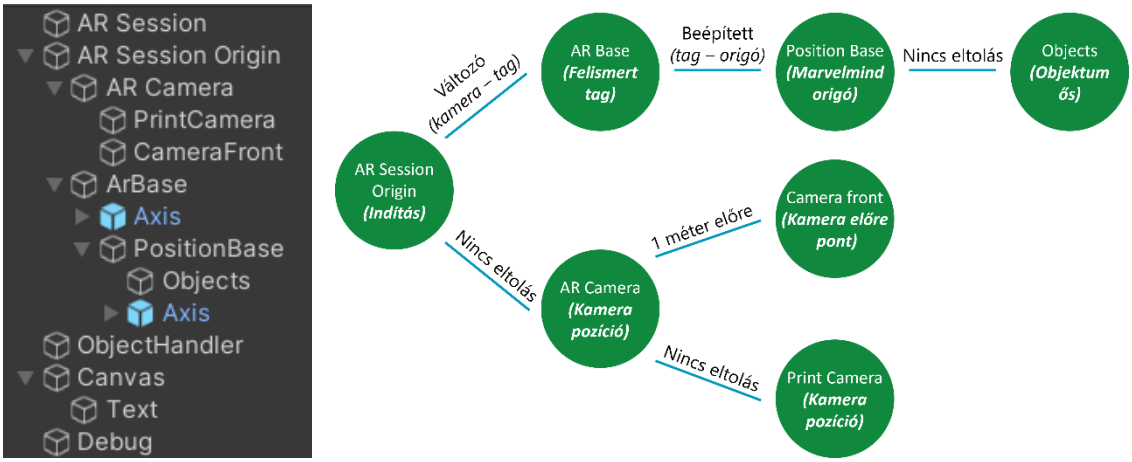
Azonban ahhoz, hogy ezzel korrigáljuk a jelenlegi elhelyezkedést, több eltérő komponens is szükséges, melyek közül az első egy olyan, amely képes felismerni a képen egy adott mintát – legyen az kép, vagy marker – majd képes annak pozícióját a kamera koordináta-rendszerében megadni. Ehhez szerencsére az AR Foundation tartalmaz egy már megvalósított komponenst, mely pontosan ezt a célt szolgálja. Ennek neve „AR tracked image manager”, mely működését tekintve a felismerendő kameraképen és az előre elmentett képek között keres azonosságokat jellemző pontok segítségével.

Ehhez azonban szükség volt legalább egy markert jelölő képre, melyet felismerve be tudja állítani a origót. Ehhez a Unity-n belül meg kellett adni egy vagy több képet és azokhoz egy-egy azonosítót. A tesztelés időtartamára egy könyvszéria két eltérő könyvének borítója került kiválasztásra ilyen ábrának, mivel azok könnyen elhelyezhetőek voltak a környezetben. Ezt követően ezeket a képeket az alkalmazás folyamatosan követi, még abban az esetben is, amennyiben azok már régen elhagyták a kameraképet. Ekkor csak az állapotuk változik meg, hogy már nem láthatóak, de a kamerától vett relatív koordinátájuk a kamera mozgatásával a megfelelő módon változik.

Annak érdekében, hogy ez a pozíció kicserélhető legyen, azaz amennyiben egy új fix pontot talál a kamera, abban az esetben a koordináta-rendszerben található idő közbeni elcsúszásokat ki tudja javítani, egy objektum hierarchia létrehozása vált szükségessé. Emiatt több olyan üres szülőobjektum – a 3D-s környezetben nem látható objektum – került létrehozásra, mely alá az egyes személyekhez tartozó elemeket behelyezve azok stabilan ott maradnak a képen, ahol a beltéri helymeghatározó szerinti koordináta-rendszerben el kell helyezkedniük.

A hierarchia 4 eltérő szintből állt (3.4. ábra), melyek közül az első az AR Session Origin. Ez határozza meg, hogy a telefon kamerája éppen hol található meg a bekapcsolástól számított koordináta-rendszeren belül. Azaz amennyiben mozgunk a

kamerával, akkor ez biztosan változni fog valamilyen irányba. Ez az irány viszont attól függ, hogy hogyan tartottuk az eszközt az indítás közben. Tehát amennyiben ebbe a koordináta-rendszerbe helyeznénk el egy a Marvelmind által adott koordinátát, abban az esetben úgy tűnne, mintha véletlenszerűen jelenne meg a karakter valahol a képen.



3.4. ábra: Hierarchia a Unity szerkesztőben (bal) és gráf formában (jobb) jelölve az átmeneteken az eltolás mértékét

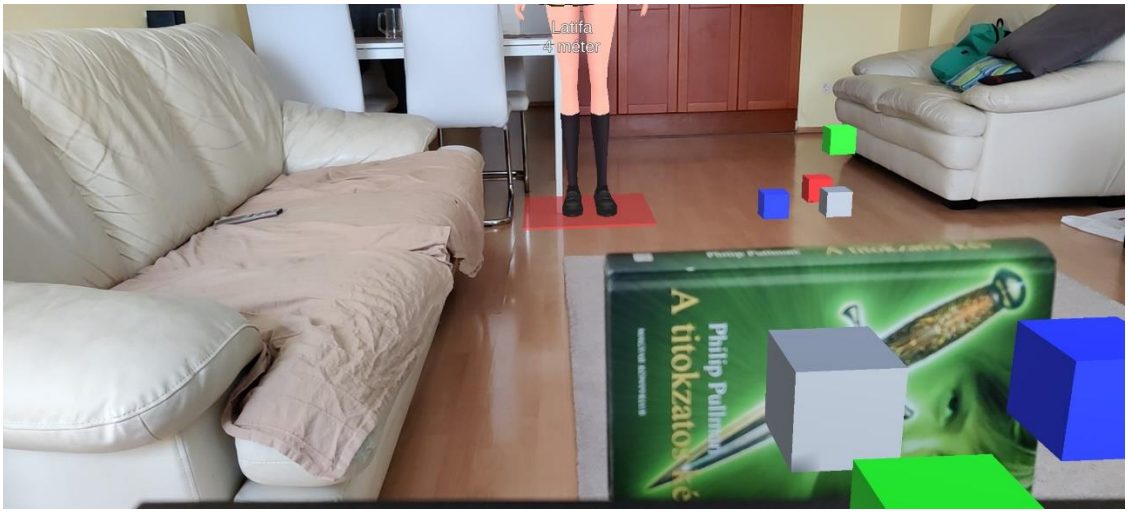
A hierarchiában ez alá két elem tartozik, melyek közül az első maga a kamera, amely a kép megjelenítéséért felelős. Ennek helye a logikában azért itt van, mivel ahogy a korábbiakban már említésre került, a Session Origin adja meg, hogy éppen hol vagyunk, a kamerának viszont a virtuális objektumok miatt pontosan ott kell lenni. Ez úgy oldható meg, hogy a hierarchiában egy szinttel lejjebb helyezzük – mivel ekkor az alsó objektum követi a szülőjének mozgását – és koordinátának csupa nullát állítunk be.

A kamerával egy szinten helyezkedik el az objektumokat megjelenítő elemek közül a második ArBase néven. Ennek pozíciója meg fog egyezni a kamerával felismert kép helyzetével a kamera koordináta-rendszerében. Azaz amennyiben a kamera a (0,1,0) pontban helyezkedik el és előre néz és ezt érzékeljük, hogy tőlünk egy méterrel előre helyezkedik el, abban az esetben annak pozíciója a (0, 1, 1) -ben lesz, és ez a pozíció lesz a globális koordinátája annak az objektumnak. Ezzel van egy olyan elemünk, mely alá az objektumokat behelyezve minden elem megegyezik a felismert képnek az elhelyezkedésével.

Azonban egy ilyen felismert objektum nem pontosan az origóban helyezkedik el, azaz amennyiben több ilyet is használunk, akkor lehet, hogy egy ilyen több tíz méterrel a tényleges kiindulási ponttól található. Ennek kiküszöbölésére egy újabb szint került bevezetésre, ahol a nullpont és a jelenlegi helyzet közötti eltéréssel lokálisan eltoljuk és elforgatjuk a megfelelő irányba a rendszert és ezzel lehetőségünk nyílik a tényleges kiindulási pontban mozgatni az elemet. Ehhez minden felismert pozícióhoz el kell mentenünk egy eltolást, ami egy 3D-s vektor, valamint egy elforgást, mely azt mondja meg, hogy mely tengely körül mennyivel kell elforgatni a koordináta-rendszert, hogy az

megegyezzen a beltéri helymeghatározónak az egyes irányaival. Hogy ezt a kettőt egymáshoz lehessen párosítani, az egyes felismert képek által visszaadott azonosítókhoz kell kötni ezeket az értékeket és így azok könnyen kiolvashatóak futás közben is.

Végezetül utolsó szintnek egy új üres objektum van beállítva. Ennek fontosabb forgatási és eltolási feladata már nincs, egyetlen célja, hogy egy koordinátában sosem változó objektumot hozzon létre, mely összefogja a beltéri helymeghatározó által küldött adatokból létrehozott szereplőket. Ez alá jön létre minden objektum, azok összes alkomponensével egyetemben. A hierarchiában ezen felül kétszer is megjelenik egy Axis elem. Annak egyedüli célja, hogy segítséget nyújtson, hogy az egyes hierarchia szintek pontosan hol is helyezkednek el a valóságban és azok merre néznek. Egy ilyen komponens egy szürke négyzet középpontjával jelöli a nulla pozíciót és további 3 eltérő színű négyzettel a tengelyeket, ahol a színek megegyeznek a Unityben használt koordináta-rendszerbeli színekkel.



3.5. ábra: Az egyes koordináták pontjai és vektorai.
Pozíció (szürke), Jobbra (piros), Fel (zöld), Előre (kék)

A 3.5. ábra mutatja a hierarchiában található elemek működését. A közelben található könyv tekinthető az egyik kezdőpontot jelölő képnek. Ahogyan az ábrán látható, annak középpontja adja az ArBase pozíciója, elforgását pedig a többi színnel jelölt négyzetek. A kép esetén a felfelé irány – amely Unity esetén az *Y* – egyezik meg a képnek a síkjára merőleges összetevővel, míg az előre a kép teteje felé, a jobbra irány pedig a kép jobb oldala felé mutat. Ez kerül a következő lépésben korrigálásra az elmentett adatokkal.

Látható, hogy megjelenik egy másik koordináta-rendszer is, ez pedig a tényleges nulla pont. Ehhez első lépésként el kellett tolni a kiindulópontot az alap koordináták szerint, azaz a fenti kép esetén (0.65, -3, 1) vektorral, ahol minden komponens méterben kerül megadásra. Ezt követően történik a következő lépés, ahol a forgatást szükséges elvégezni, mely során a koordináta-rendszert az *Y* tengely körül 180°-ot, míg a *Z* tengely körül -90°-ot forgatjuk. Így kapjuk meg a Marvelmind helymeghatározóval azonos irányú és

koordinátájú pontokat. A fenti ábrán a karakter pozíciója a (0, 1) pontban helyezkedik el, amely azt jelenti, hogy a kék négyzet irányában 1 métert el kell tolni, azonban a másik (piros) irányban nem szükséges mozgatni.

3.2.2 Medián szűrés

Ezt követő lépésként meg kellett oldani bizonyos problémákat, melyek a lehelyezésből és az ahhoz szükséges lépésekből következtek. Ezek közül az első és egyben legfontosabb, hogy amennyiben folyamatosan a képen volt a felismerendő tag, mely meghatározza a nullpontot, akkor bizonyos időközönként az érzékelt koordináta változott és ebből kifolyólag minden 3D-s elem, valamint a teljes környezet ugrált. Hogy ez a jelenség ne jöjjön létre, egy változás csak abban az esetben kell hogy megtörténjen, amennyiben ténylegesen egy bizonyos időn keresztül folyamatosan a jelenlegi pozíciótól eltérő koordinátát ad vissza a felismerés. Ennek megvalósítására szükségessé vált egy szűrő.

Itt a szerepéből következően, amely a zaj eltüntetése volt, egy Medián szűrőre esett a választás, melynek célja, hogy egy bizonyos mennyiségű elemnek a mediánját válassza ki, azaz sorberendezett módon a középső elemét. Ennek előnye, hogy így amennyiben egy ugrás történik, azaz néhány képkockán keresztül eltérő értéket érzékelünk, akkor az a sorba rendezés egyik szélén fog megjelenni, így a középső elem nem módosul. A fenti feladatban azonban nem egy koordinátára, hanem egy teljes koordináta-rendszere kellett ezt megvalósítani.

Ennek alapja nem különbözik sokban az egy koordinátára megírt megoldástól. Ennek lényege, hogy a koordináta-rendszert 4 eltérő 3D-s vektorral reprezentáltam. Ezek közül az első adja meg a tényleges pozíciót, azaz a fenti ábrán ez felel meg a szürke négyzetnek. A másik három vektor pedig az *X* az *Y* és a *Z* irányokba mutató egységvektorok. Mivel itt minden vektor is 3 elemből áll, ezért ezeket is szét kellett szedni összetevőkre és mindegyiken külön-külön egy medián szűrést elvégezni.

Ennek folyamata, hogy az eddigi elemeket egy fix hosszúságú listában tároljuk, majd amennyiben új elem érkezik, akkor azt beletesszük az eddigi elemek közé. Amennyiben ezzel meghaladjuk a kapacitást, akkor a legrégebbi elemet viszont ki kell venni a többi közül. Ezt követő lépésként az így kapott elemek közül a középső kiválasztható egy sorba rendezést követően. Itt viszont figyelni kell, hogy ez a módosítás ne az eredeti elemeken keresztül történjen, mivel úgy a következő esetben nem a legidősebb, hanem egy szélső értéket távolítanánk el az adatok közül, amivel a szűrés lényege veszne el.

Ezt követően 4 módosított vektorral rendelkezünk, melyek közül a pozíció egyértelműen meg van határozva, azonban a koordináta tengelyek irányvektorai a medián szűrést követően nem biztos, hogy ortonormált rendszert alkotnak. Ennek oka, hogy a mediánszűrés hatására létrejöhetnek olyan vektorhármasok, amelyek nem merőlegesek egymásra. Ennek kiküszöböléséhez véghez kell vinni egy ortogonalizációt [18], melynek

során az egyes vektorokat vesszük és kiválasztjuk belőlük azokat a komponenseket, melyek egy olyan rendszert alkotnak, ahol a 3 vektor egymásra minden esetben merőleges. Erre a Unity rendszeren belül van beépített megoldás, melynek a 3 vektort átadva azokat oly módon módosítja, hogy ilyen bázist hozzon létre. Ennél az átadásnál azonban számít a sorrend, mivel az első vektor lesz az, amely azonos lesz a kiszámított mediánszűrt elemekkel. Az alkalmazásban az *Y*, azaz a felfelé mutató vektor került kiválasztásra, mivel ez vektor határozza meg a talaj síkját, így ennek hibás kirajzolása lenne a legfeltűnőbb.

A fenti megoldással volt azonban még két nagyobb probléma. Ezek közül az első, hogy amennyiben a kép eltűnt a kameráról, abban az esetben nem bővültek újabb elemekkel a szűrők és amennyiben ez éppen mozgás közben történt, úgy egy hibás állapot maradt érvényben. Emiatt meg kellett oldani, hogy amennyiben már nem jönnek új adatok, akkor az utolsó érzékelt megoldások felé mozduljon el az érzékelés. Ennek megvalósítása úgy történt, hogy amennyiben egy bizonyos időn belül (~200 ms) nem érkezik új adat, akkor a már meglévőkből a régebbi elemek folyamatosan törlésre kerülnek egészen addig, míg már csak az utolsó 5 elem található a listában. A folyamat során minden törlést követően egy medián szűrést végzünk és így a végeredmény, az utolsó 5 érzékelés mediánjához fog tartani, mely pontosabb, mint a teljes lista mediánja.

A másik probléma a szűréshez szükséges elemek számából következett. Első megoldásként egy fix elemszám megadására volt lehetőség, azonban hamar kiderült, hogy ez a jövőben nem lesz egy helyes megoldás. Ennek oka, hogy míg ezt könnyű megadni, addig eltérő eszközök esetében a szűrés más és más eredményeket adhat, annak függvényében, hogy mekkora sebességgel kerülnek beolvasásra az új képkockák. Azaz amennyiben egy régebbi telefonról használjuk az alkalmazást, melynek kamerája csak 30 FPS-re képes, akkor egy 120 elemű mediánszűrő összesen 4 másodpercnyi adatot tárol el, addig egy modernebb eszköz esetén 60 FPS-es kamerával ez már csak 2 másodperc.

Ennek kiküszöbölésére több megoldás is lehetséges, melyek alapja azonban azonos. Lényegük, hogy nem képkocka alapon kerülnek ki az listából az elemek, hanem idő alapon. Így a kamera bármekkora képkocka per másodperc rátával is rendelkezik, a medián szűrő ugyanúgy fog működni, maximum a benne található adatok mennyisége lesz eltérő. Végezetül két megoldás jöhet szóba, melyek közül a korábbi a pontosabb, azonban komplexebb, míg utóbbi az egyszerűbb módon implementálható, azonban kicsit pontatlanabb megoldás.

Az első lényege, hogy azon felül, hogy elmentjük a tömbbe a kapott vektorokat, mellé mentésre kerülne még egy időbélyeg is, melyek azt határozzák meg, hogy mikor kerültek elmentésre az adott értékek. Ennek hatására a medián szűrőből nem fix méret után töröljük az elemeket, hanem amennyiben ez az időbélyeg egy bizonyos időpontnál régebben történt. A másik megvalósítása nem módosítja a listát, hanem a lista fix méreténél figyelembe veszi a kamera FPS számát, mely a Unity rendszer segítségével

lekérhető. Ezt követően a teljes működés megegyezik az előzőekben bemutatott megvalósítással.

Az alkalmazásban ez utóbbi megoldás került felhasználásra, mivel bár kicsit pontatlanabb, amikor egy-egy képkocka kimarad, azonban ez a tesztelés alatt olyan kevésszer történt meg, hogy egy ilyen mértékű pontatlanság még megengedhető. Ennek a szűrésnek a megvalósítását követően már ténylegesen rendelkezett az alkalmazás egy stabil, fix kiindulási ponttal és egy koordináta-rendszerrel, mely megegyezett a beltéri helymeghatározó által visszaadott értékekkel.

3.2.3 Karakter megjelenítése

A folyamat következő lépéseként a karakter megjelenítése volt szükséges. Ez mind a tesztelés mind a fejlesztés során is segítséget nyújtott, mivel így látható lett a tényleges pozíciója és orientációja az egyes megjelenítendő objektumoknak. A program kezdetleges tesztelési fázisa alatt a karakter minden esetben látszik, azonban a valós környezetben ez csak akkor kellene hogy megtörténjen, amennyiben az egy bizonyos távolságnál közelebb helyezkedik el és valamilyen objektum által ki van takarva. Azaz amennyiben egy személy a 10 méteres sugarunkban van és valamilyen objektum van köztünk és a megfigyelt személy között, akkor annak helyén egy sziluett jelenik meg. Ezzel amennyiben egy fal választja el a két embert, akkor az alkalmazás segítségével lehetőségük van látni, hogy a fal másik oldalán az egyén pontosan hol helyezkedik el.

Ennek megvalósításához újból több eltérő komponensre volt szükség, melyek együttes működése segítségével történik a kitakarás. Az első lépés annak eldöntése volt, hogy az egyén, akinek a megjelenítése még kérdéses, a felhasználó által látható helyen helyezkedik-e el. Ez egy sugár (ray) elindításával történt, ami a kamera pozíciójából a másik személy irányába mozgott, azonban ahhoz, hogy el lehessen dönteni, hogy a sugár ténylegesen el is éri a vizsgált egyént, ahhoz annak pozíciójában szükség volt egy ideiglenes 3D-s alakzat létrehozására.

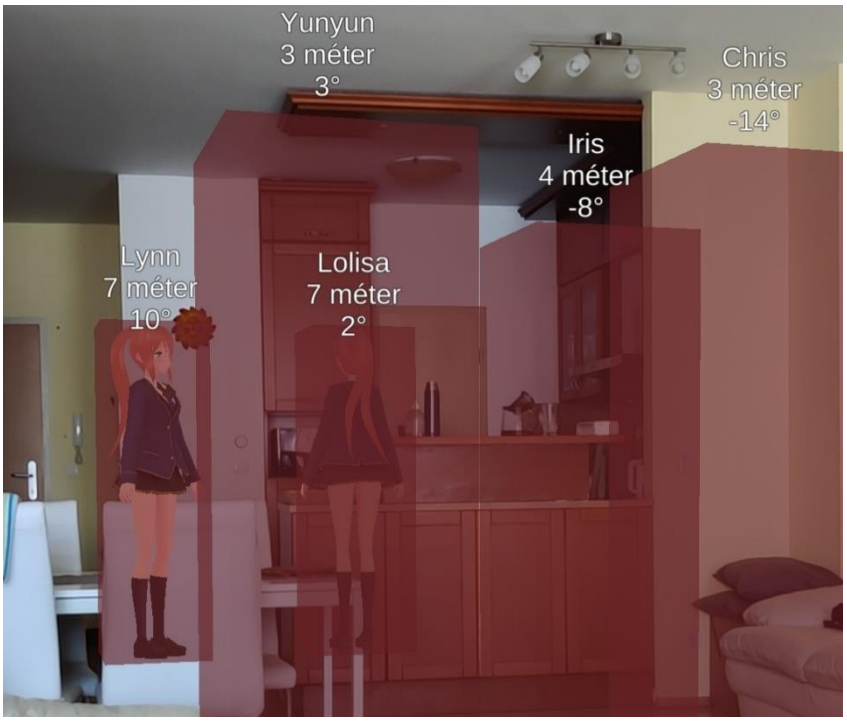
3.6. ábra: A karakter kitakarása láthatatlan elemekkel

Tehát a sugárkövetés rész 3 eltérő feladatból állt. Először fel kellett ismerni a területen minden olyan síkot, melyek képesek kitakarni egy személyt. Erre az AR Foundation keretein belül létezik egy megoldás. Ennek egy használat során készült képét mutatja a 3.6. ábra. Ahogy látszódik nem tökéletes a kitakarás, azonban a megvalósítás során ez nem akkora befolyásoló tényező, mivel épületekben nem a kis tárgyak, hanem falak és egyéb nagyméretű objektumok kitakarása lesz a mérvadó.

Ezt követő lépésként egy ideiglenes 3D-s objektum létrehozására volt szükség, mely rendelkezett egy „collider”-rel, melynek szerepe a jelenlegi helyzetben, hogy a sugár így neki tudott ütközni. Ezt követően két eltérő sugár kerül kiküldésre, egy a másik objektumnak fél méteres magasságába, míg a másik 1 méteres magasságába. Ennek szerepe, hogy csak abban az esetben rajzoljuk ki a karaktert, amikor annak nagyobb része nem látszódik, nem pedig csak egy pont. Amennyiben mind a két pont az adja vissza, hogy nem sikerült elérni az objektumot, azaz valamilyen másik 3D test a területen belül kitakarja, abban az esetben kerül csak megjelenítésre a sziluett. Tehát amennyiben legalább az egyik pont azt mondja, hogy látszódik a személy, akkor nem jelenítünk meg semmit. Ezt követően utolsó lépésként pedig ez az ideiglenes objektum törölhető és amennyiben a megjelenítés szükséges, akkor egy rendes objektum hozható létre a helyére.

A fent részletezett megoldásnak azonban van egy nagy hátránya. A kitakart objektum egy áttetsző elemmel a továbbiakban is ki van takarva, azonban az alkalmazásnak a célja pontosan az, hogy amennyiben a kitakarás fennáll, abban az esetben kellene az objektumot megjeleníteni. Erre megoldásként egy kétkamerás rendszer implementálása történt meg, melynél az első és egyben fő kamera felel minden olyan feladatért, mely az AR köré csoportosul, azaz a síkok felismeréséért és kirajzolásáért, valamint a kamerakép képernyőre való továbbküldéséért. Ezen felül rendelkezik a rendszer még egy további kamerával is, melynek egyedüli célja, hogy az így kapott objektumokat kirajzolja. Ez a másik eszköz minden rajzolását követően indul, így bár az egyes síkok a továbbiakban is ki vannak rajzolva, a 3D-s objektumok rajzolásáért felelős kamera erre rárajzol. Emiatt láthatóvá válnak a kitakarást követően is az egyes karakterek.

Ennek megvalósítása a Unityben található rétegek segítségével történik. A rendszer tartalmaz előre beépített elemeket is, mint a „water”, mely a víznek a kirajzolását segíti, valamint például a UI-hoz is tartozik egy ilyen. Ezek kerültek kiegészítésre még két elemmel, mely a síkok és a megfigyelt objektumok megjelenítéséért voltak felelősek. Ezt követően az egyes játékelemeket is a megfelelő módon be kellett állítani, azaz mind a két elem esetén a megfelelő rétegre kellett helyezni őket. Így amennyiben az első kamerát vizsgáljuk látható, hogy a megfigyelt objektumokon kívül minden renderelést ő végez, azonban a rákövetkező kamera végzi a felismert figurák rajzolását. Így minden esetben a már kirajzolt síkok hátrébb fognak elhelyezkedni, mint a személyekhez tartozó 3D-s szereplők, ezzel ellehetetlenítve azt, hogy a kitakarás megtörténjen.



3.7. ábra: A megvalósított kitakarás

Ennek végleges megvalósítását mutatja a 3.7. ábra. Ebben látható, hogy a közeli karakterek nem látszódnak, míg a távolban elhelyezett szereplők igen, mivel azok esetén a szék és a pult teljes egészében kitakarja a személyeket. Amennyiben viszont egy karakter több mint 10 méterre helyezkedne el, akkor abban az esetben sem jelenne meg 3D-s karakter a helyén, amennyiben az teljes egészében ki lenne takarva.

3.2.4 Karakter elforgatása

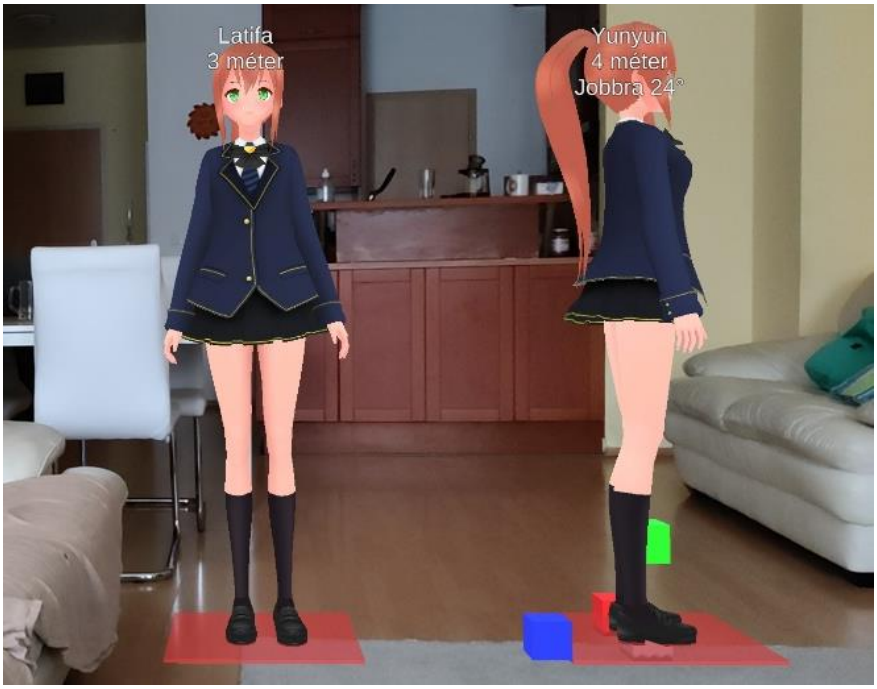
Annak érdekében, hogy ne csak azt tudjuk ábrázolni, hogy az egyén hol helyezkedik el a területen belül, hanem azt is, hogy az merre néz, szükségessé vált egy második tag alkalmazása is. Ez a szerver résznél már bemutatott módosításhoz vezetett, melyet felhasználva lehetségessé vált, hogy a karakter el tudjon forogni. Ennek megvalósítása során azonban több probléma is fellépett.

Ezek közül az első, hogy a két tag *X* és *Y* koordinátájának különbségével kapott vektor nem húzható rá egy az egyben a 3D-s megvalósításra, melynek oka, hogy a forgatást a 3D-s koordináták felhasználásával lehetséges kiszámolni. Így első lépésként azt kellet meghatározni, hogy a kapott 2D-s pozíciók a térben hol helyezkedtek el. Amennyiben ezek meghatározása sikeres volt, akkor már rendelkeztünk egy előre irányba mutató vektorral. A forgás meghatározásához második elemként a területen belül a felfelé mutató irányra is szükség volt, azonban ez nem függ a kapott koordinátától, így ez egyértelműen lekérhető.

Ezen felül lehetséges, hogy egy személynél nem szeretnénk elforgással számolni, mivel abban az esetben egy helyett két tagre van szükségünk. Ezért biztosítani kellett egy

olyan lehetőséget is, amennyiben egy egyén csupán egyetlen taggel van azonosítva. Ebben az esetben a kitakarás során létrehozott 3D-s figura minden pillanatban a felhasználó felé fog nézni. Ehhez azonban meg kellett határozni, hogy a kamera a 2D-s síkra vetítve mely pontban található és ezen pontot felhasználva lehetséges meghatározni az előre irányt.

Ehhez a kamera pozíciójában lehelyezésre került egy áttetsző objektum, melyet azt követően a lokális koordináta-rendszerben – azaz a legmélyebb hierarchiában, ahol a szög és a pozíció megegyezik a Marvelmind rendszerben található adatokkal – a talaj síkjába került elhelyezésre. Ezt követően ennek a pontnak a globális koordinátáját felhasználva, lehetséges volt meghatározni a Marvelmind adatok 3D-s pontja és ezen pont között egy vektort, amelyet felhasználva már folytatható volt a számítás a fent részletezett folyamat alapján.



3.8. ábra: A karakter 1 tages rendszerben (Latifa) és 2 tages rendszerben (Yunyun)

Erre mutat egy példát a 3.8. ábra, ahol jól látható, hogy a bal oldalon álló karakter a kamera irányába néz, míg a jobb oldalon álló a kamerától elfelé. Ennek oka, hogy a rendszer azt a két opciót, hogy 1, vagy 2 taggel működjön egyszerre is képes kezelni. Amennyiben hiányzik a hátsó pontot meghatározó tag, vagy az egy bizonyos tűréshatáron belül helyezkedik el, abban az esetben úgy tekinti, mintha csak 1 tag lenne rendelve az objektumhoz és azt a felhasználó felé tekintve rajzolja ki. A fenti ábrán a bal oldali karakter a kamera felé tekintve látható, mivel ebben az esetben csupán a fő koordináta volt megadva, úgy, hogy az *Y* értéke 1000 volt, az *X*-é pedig 0. Ez okozza azt is, hogy a figura a kiindulóponttól egy méterrel eltolva látható. A másik szereplő esetében a fő pozíció az kiindulási pontban volt található (ezt jelöli a szürke négyzet), a második irányt

megadó tag pedig ott helyezkedett el, ahol a bal oldali karakter pozícióját megadó elem. Ebből egy irány számolva látható, hogy így ténylegesen helyes, hogy a karakter jobbra néz, azaz háttal áll a másik karakternek.

3.2.5 Szöveg kirajzolása

Az alkalmazás célja nem csak az, hogy amennyiben egy egyén nem látható, hanem például egy fal kitakarja, akkor megjelenítsünk a helyén egy 3D-s grafikát. Ezen felül szándék, hogy a mért adatokat valamilyen módon felhasználjuk, hogy azok segítséget nyújtsanak az emberek megtalálásában. Emiatt szükség volt valamilyen megoldásra, melynek segítségével amennyiben az ember körbenéz egyértelműen meg tudja határozni, hogy a keresett személy milyen távolságra és merre helyezkedik el tőle.

Mivel minden szereplőről folyamatosan érkezik pozícióadat, így az alkalmazás ezt képes felhasználni, hogy azokból távolságot, valamint szöget számítson. Ezen felül, hogy ezek ne bármilyen kontextus nélkül jelenjenek meg szükségessé vált minden egyes személyhez egy azonosítót (például a személy nevét) is eltárolni. Ennek felépítéséről már szó esett a szerver leírása során, így itt az nem kerül részletezésre. Hogy ez a megjelenítés ténylegesen megtörténjen viszont több eltérő lépésre is szükség volt, melynek első és egyben legfontosabb eleme, hogy a képernyő egy meghatározott pontjára a rendszer képes legyen kirajzolni egy szöveget úgy, hogy az minden esetben a felhasználó felé tekintsen, függetlenül a hozzá tartozó karakter és személy helyzetétől.

Ehhez két eszközre volt szükség. A Unity rendszeren belül lehetséges a képernyő fölé rajzolni. Ezt Canvasnak hívják, melyet főként a felhasználói felületek megjelenítésére használnak, azonban az alkalmazásban ez felel az egyes karakterek felett kiírt szövegek megjelenítéséért. Ahhoz, hogy ennek pozícióját meghatározzunk, azonban szükséges, hogy a 3D-s objektum pontosan hol is helyezkedik el a képernyőn. Így azon felül, hogy létrehozunk egy szövegmezőt, mely mindig a képernyő síkjában jelenik meg, szükségessé vált egy olyan objektum létrehozása is, mely követi a karakter mozgását és ezt felhasználva a rendszer vissza tudja számolni, hogy hol kell megjeleníteni a szöveget.

Ehhez a Unity azon megoldása került használatba, mely képes pontosan visszaadni, hogy egy 3D-s objektum a képernyő mely pixelén lenne kirajzolva, akár abban az esetben is amikor az ténylegesen nincs megjelenítve. Ez a megoldás egy háromdimenziós vektort ad vissza, melyek közül az első kettő adja meg a pixelt, a 3. érték pedig, hogy milyen távolságra van tőlünk az objektum, mely negatív abban az esetben, ha a mögöttünk található a pont. Ez utóbbi érték lett felhasználva annak érdekében, hogy a szöveg amennyiben a személy nincs rajta a képen, akkor is megjelenjen. Itt a megjelenés esetén fontos volt azt elérni, hogy amennyiben a felhasználó elkezdi az eszközt abba az irányba mozgatni, mint ahol a szöveg van, akkor idővel, de eljusson a tényleges karakterhez. Egy valós környezetben azonban többen vannak egy területen belül, minthogy azt átlátható

módon meg lehetne jeleníteni, így ez a funkció, hogy a nem látható személyek neve is kiírásra kerül csak egy bizonyos távolságon belül működik.

Ezt mutatja a 3.9. ábra, ahol a bal oldali képen mind a három név látszódik, még akkor is, amennyiben a bal oldalon álló személy nincs rajta a kameraképen. Ezzel szemben áll azonban a jobb oldali kép, ahol az 5 méternél távolabb álló személyhez tartozó szöveg már nem jelenik meg, csak amennyiben ténylegesen abba az irányba tekintünk. Ennek azonban jelenleg még nincs felső korlátja, így a teljes területen meg tudjuk nézni, hogy ki hol tartózkodik, viszont ehhez körbe kell fordulnia az illetőnek.

3.9. ábra: Szövegek megjelenítése a képernyőn

Ahhoz, hogy a fenti lépéseket meg lehessen tenni, szükséges kiszámolni a karaktertől való távolságot. Ez nem lenne egy nehéz feladat, hiszen adott mind a karakterhez tartozó 3D-s koordináta, valamint a kamerához tartozó koordináta is. Azonban ez azt is figyelembe venné, amennyiben a tagek eltérő magasságokban helyezkednének el. Ezért első lépésként mind a két vektort le kell vetíteni a talaj síkjára is itt kell meghatározni a távolságát a kapott 2D-s értékek szerint. Ennek kiszámítása egy gyorsan elvégezhető feladat, hiszen a Unity támogatja, hogy két koordinátát egymásból kivonjunk, majd ezen meghívható egy olyan metódus, mely elvégzi a maradék lépéseket és egy tizedes törtet ad vissza a kiszámított távolsággal. Ez az érték méterben értetendő, de mivel ez a felhasználó számára is érthetőbb, mintha milliméterben lenne kiírva a távolság, emiatt ez a megjelenítendő érték is. Az átláthatóság kedvéért azonban ahelyett, hogy a tizedespont utáni érték is kiírásra kerülne, a kapott érték a legközelebbi egészre kerekítődik.

Következő lépés a karakterrel bezárt szög meghatározása, melyhez a korábbiakban már kiszámított vetületek is segítséget nyújtottak. Ennek oka, hogy egy 3D-s koordináta esetén a bezárt szöget két eltérő értékkel is lehet jellemezni, azonban a felhasználó számára ezek közül a talaj síkjában való eltérés mond többet. Emiatt a fent már részletezett módon mind a kamera, mind pedig az objektum levetítésre került a talaj síkjába, melyek kiegészültek egy harmadik objektummal is, mely a kamerának volt egy aleleme. Ez egy nem látható elem, mely együtt forog és mozog a felhasználóval, azonban

tőle mindig egy méterrel előrébb van. Ennek célja, hogy amennyiben két vektor által bezárt szöget számolunk, akkor míg az előbbi vektor meghatározható az objektum és a kamera közötti különbséggel, és ennek ismerete elégséges egy távolság kiszámításához, addig itt szükség volt egy további vektorra, amely a kamera és az előre irányba mutató pont közötti különbség volt. Ezeket felhasználva már lehetett egy szöget számítani, melyet radiánból átváltva a felhasználó irányába már kiírható volt. Ezzel kapcsolatban azonban felmerült még egy probléma. Mivel a felhasználó számára a -87°-ra lévő személy nehezen értelmezhető, emiatt ezt még át kellett váltani egy 0 és 180°közötti értékre, melyet megelőzően ki kellett írni, hogy ez balra, vagy pedig jobbra keresendő.

3.10. ábra: Az elfogás megjelenítése a távolság értékek alatt

Ahogy a fenti 3.10. ábra is mutatja a távolság számítása a kamerától történik, ezért fordulhat elő, hogy míg a középső karakter közelebb helyezkedik el körülbelül 3,4 méterre, addig a szélső szereplők már túlmennek a 3,5 méteres határon, így a kerekítést követően oda már 4 méter kerül kiírásra. Ezen felül látható, hogy az elforgások is megjelennek a távolságok alatt és annak függvényében, hogy a karakter a kamerától balra, vagy pedig jobbra helyezkedik el, azt jelzi a felhasználó felé. Továbbá egy bizonyos szög alatt – ami jelenleg ±15° – nem jelenik meg az elforgás. Ezzel az egyértelműen előttünk lévő személyek esetén nem zavarják be a felhasználót a nagyon kis szögekkel.

3.2.6 További funkciók

Ezzel a képességgel együtt az alkalmazás már képes volt kirajzolni karaktereket, amennyiben azok valamilyen okból kifolyólag ki vannak takarva, ezen felül a figurák felett megjelenítésre került egy szövegmező, mely megjeleníti a személy nevét, annak távolságát a megfigyelőtől, valamint azt, hogy mekkora szöggel és mely irányba található az egyéntől. Azonban vannak olyan megoldásai is az alkalmazásnak, melyek már nem az alapvető használatnak a részét képezték, hanem valamilyen egyéb segítő megoldást

jelentettek. Ezek két csoportba sorolhatóak, melyek a fejlesztést segítő eszközök, valamint a ténylegesen a felhasználók számára is látható képességek. Ezek közül a korábbi az alkalmazás elkészítése során a fejlesztő számára nyújtott valamilyen típusú segítséget, míg utóbbi a megjelenést követően mindenki számára hasznos funkciókat biztosít. A következő alfejezetben két ilyen komponens kerül bemutatásra, melyek közül az első az implementációt segítő elemek közé tartozik, míg a második pedig a másik csoportba.

Befoglaló téglatest

Egy kibővített funkció, hogy minden egyes szereplőhöz lehetséges megadni egy befoglaló téglatestet. Ennek célja, hogy amennyiben az alkalmazást például egy olyan raktárban használnánk, ahol önvezető járművek is vannak, akkor ezzel a megoldással meg lehet jeleníteni köréjük egy területet, ahová tilos a belépés. Hasonló módon, amennyiben egy terület valamilyen okból le van zárva, akkor egy tag segítségével létrehozható köré egy ilyen objektum, mely egyértelműen jelzi a munkás számára, hogy tilos és akár veszélyes is a megközelítése.

Annak érdekében, hogy ez megfelelő módon működjön, azonban két eltérő típusú befoglalóra van szükség. Ezek közül az első, amennyiben egy tárgy, vagy objektum köré egy olyan mezőt szeretnénk létrehozni, amely a talaj szintjén utal arra, hogy a biztonságos távolságon belül értünk. Ebben az esetben a befoglaló téglatestnek nincs magassága, így nem takarja ki a területet, azonban előfordulhat, hogy ezt valaki, mikor maga elé néz nem veszi észre, így amennyiben szükség van rá, akkor lehetséges ennek a befoglalónak egy magasság megadása is. Ebben az esetben beszélünk ténylegesen egy téglatestről, mivel ekkor már egy bizonyos távolságon belül kitakarásra kerül a terület, amely egyértelműen jelzi, hogy ezen túlmenni tilos.

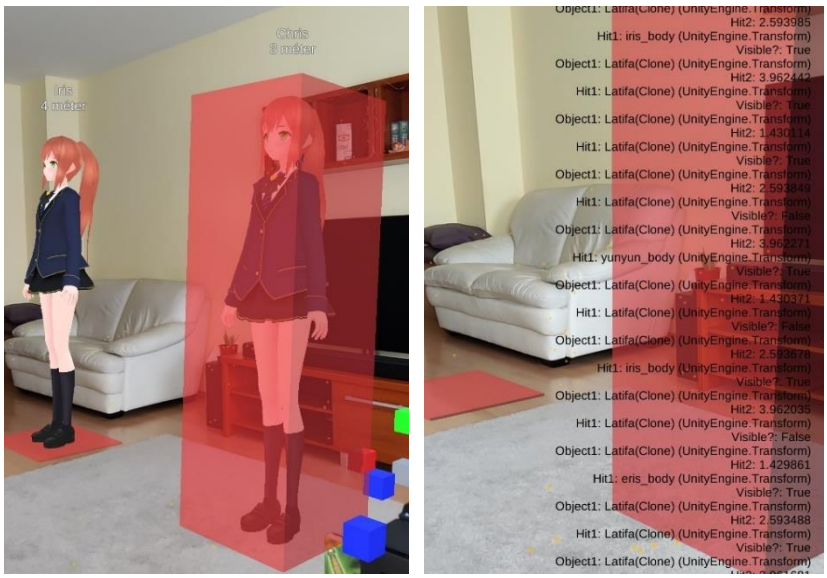
Hibák megjelenítése

Egy másik segítő része az alkalmazásnak annak érdekében jött létre, hogy az egyes hibaüzenetek kiírásra kerülhessenek. Erre egy alapvető Unity alkalmazásban használható a programban található konzol. Hasonló módon, amennyiben egy mobil telefonra készülő alkalmazást fejlesztünk, akkor lehetőségünk van az alkalmazást a Unity programon belül futtatni egy kiegészítő Android alkalmazás segítségével, melynek hatására valós időben követhető, hogy mely objektum hol helyezkedik el a játéktérben. Azonban amennyiben AR alkalmazást fejlesztünk, abban az esetben ezen opciók nem vehetőek igénybe, így a program futása során üzenetek kiírására csupán két lehetőségünk van.

Ezek közül az első opció Androidos telefonok esetén, hogy a beépített Logcat nevű eszközt használjuk, mely folyamatosan kiírja a hibakereséshez és a hibákhoz tartozó üzeneteket. Így ennek a megoldásnak elindítását követően a kapott üzenetek itt visszaolvashatóak. Ennek előnye, hogy minden kiírt üzenet itt visszaolvasható, azaz lehetőségünk van elemezni azt, hogy a folyamat alatt mi történt, miért ment tönkre az

alkalmazás. A másik opció, hogy a program berkein belül egy szövegmezőre írjuk az összes adatot, melyet a képernyőn megjelenítünk. Ennek hátránya a másikkal szemben, hogy összesen annyi sort láthatunk, amennyi éppen a képernyőre kifér, cserébe viszont az eszköznek nem kell összekapcsolt állapotban lennie egy számítógéppel, ahol a másik esetben üzenetek kiírása lenne követhető. Egy AR alkalmazás fejlesztése esetén célszerű tud lenni, hogy körbe lehessen járni a területet úgy, hogy közben bizonyos információkat lássunk, ezért az alkalmazás tartalmaz egy ilyen mezőt.

Azonban amennyiben ehhez az elemhez minden új üzenetet hozzáadunk, akkor rövid idő alatt el tudunk jutni egy olyan mennyiségű szöveghez, melynek hatására elkezd a program akadozni. Ennek kiküszöbölése végett a szöveg nem hozzáfűzésre kerül a már megjelenített adatokhoz, hanem az egész egy sorban van tárolva, amelynek megadható a mérete. Így bár minden sor hozzáadását követően a teljes szöveg újra kiírásra kerül, azonban csupán azok a sorok, melyek az elmúlt időben jöttek létre. Ez a tesztelés alatt átlagosan 40-60 sor szokott lenni, mivel így a teljes képernyő lefedhető volt.



3.11. ábra: Befoglaló téglatest és hiba üzenetek megjelenítése a képernyőn

A fenti két funkciót mutatja a 3.11. ábra, ahol a bal oldali képen látható a két eltérő típusú befoglaló téglatest benne az egyes karakterekkel, a jobb oldali ábrán pedig a kitakarás fejlesztése során egy hibaüzenet. Az ábrán látszódó szövegmennyiség 3-4 képkocka alatt keletkezett, így egy alkalmazás futása során célszerű kevés adatot kiírni egyszerre, hogy az azokban bekövetkező változás jól megfigyelhető legyen.

A jelenlegi fejezetben bemutatásra került 4 eltérő AR platform, melyen keresztül alkalmazások fejleszthetőek, majd ezek közül kiválasztásra került egy, melyen a fejlesztés történt. Ezt követően ismertetésre kerültek a fejlesztés lépései, a kiindulási pozíció meghatározásától egészen a karakter mozgásáig és forgásáig. Továbbá megvalósításra került néhány további, fejlesztést és használatot segítő komponens is. Ezt követően nem volt más hátra, mint a 3 elkészült alkalmazást egymás mellett tesztelni.

4 Alkalmazás tesztelése

Az alkalmazás a korábbi részek által leírt módon elkészült, melynek hatására nem volt más hátra, mint hogy a teljes rendszer tesztelésre kerüljön. Ez két eltérő fázisban került megvalósításra. Az első részben a helymeghatározó rendszer nem volt fizikailag jelen a helyszínen, annak megléte csupán az alkalmazás segítségével lett elhitetve a szerverrel, így lehetőség volt szabadabb módon mozgatni a karaktereket. A második fázisban való tesztelés során már a helymeghatározó rendszer küldte a tényleges adatokat, melyek az alkalmazás által feldolgozásra kerültek és a mért pozíciók jelentek meg a képernyőn.

4.1.1 Helymeghatározó nélküli tesztelés

A tesztelés első részeként annak eldöntése volt a cél, hogy a szervertől kapott információk megfelelő sebességgel frissülnek, valamint amennyiben egy új szereplő lép be a területre, vagy hagyja el, akkor az alkalmazás képes ezt a megfelelő módon kezelni. Ehhez a Marvelmind feldolgozó alkalmazásban a tesztelés üzemmód bekapcsolását követően lehetőségünk volt új azonosító értékeket és pozíciókat megadni. Azonban annak érdekében, hogy ez működhessen, első lépésként fel kellett építeni a szerver résznél már részletezett struktúrát, hogy az egyes azonosítókhoz hozzákössük a felhasználót, a nevével és a hozzá rendelt biztonsági zónával. A szerver ezen részéhez azonban nem állt rendelkezésre még alkalmazás, így ez a folyamat a Postman segítségével történt. Ebben – hogy a karakter megjelenjen – elég lett volna egyetlen tag hozzáadása, de mivel a tesztelés során a forgás működése bizonyítást kellett, hogy nyerjen, ezért két eltérő tag került felvételre karakterenként.

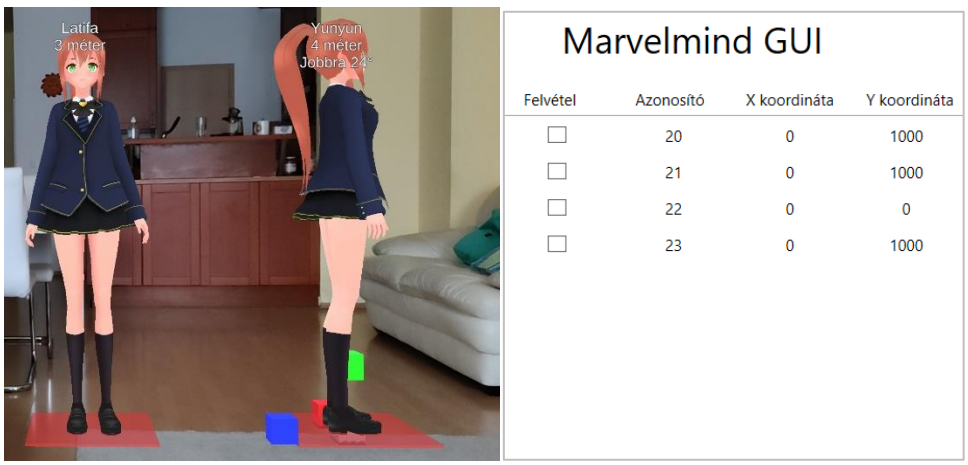
4.1. táblázat: A tesztelés során hozzáadott karakterek

Tag ID	Azonosító	Típus	Pozíció	Név	Befoglaló	Magasság
20	latifa	0 (személy)	0 (elöl)	Latifa	(500, 500, 0)	1700
21	latifa	0 (személy)	1 (hátul)	Latifa	(500, 500, 0)	1700
22	yunyun	0 (személy)	0 (elöl)	Yunyun	(500, 500, 0)	1700
23	yunyun	0 (személy)	1 (hátul)	Yunyun	(500, 500, 0)	1700
24	chris	0 (személy)	0 (elöl)	Chris	(500, 500, 0)	1700
25	chris	0 (személy)	1 (hátul)	Chris	(500, 500, 0)	1700

A 4.1. táblázat mutatja az ily módon hozzáadott szereplőket. A tesztelés során a befoglaló téglatest minden karakter esetén egy a földre kirajzolt, tehát magassággal nem rendelkező terület volt. Ezen felül mivel a fejlesztés során használt 3D-s karakter 1,7 méter magas volt, így magassághoz is minden esetben ez került beállításra, továbbá mivel

jelenleg az alkalmazás csak személyek követését támogatja, így a típus is minden esetben 0, amely az embert jelölte. A főbb eltérések, hogy kettő soronként eltér a név és az azonosító, mely így együttesen jelöl egy karaktert. Ezen a párokon belül is található különbség, méghozzá a tag azonosítójában és annak pozíciójában. Utóbbi jelöli, hogy a taget a pozíció meghatározására használjuk (elöl), vagy pedig annak eldöntésére, hogy a személy éppen merre néz (hátul). Amennyiben utóbbi nem foglalkoztatna minket, akkor azok a sorok kihagyhatóak.

Ezt követte a tényleges tesztelés, ahol a fenti karakterek megjelenítése került kipróbálásra az alkalmazásban. Első lépésként egyetlen személy került hozzáadásra, így az elvárt kimenet, hogy amennyiben a Marvelmind alkalmazásba a 20-as azonosítójú tag mellé megadásra kerül egy koordináta, akkor csak egyetlen 3D-s objektum fog megjelenni. Ez sikeresen megtörtént, így következő lépésként már több eltérő képesség egyszerre való kipróbálása következett. A három karakter közül kettő esetén megadásra került a megfelelő tagek mellé néhány koordináta. A próbálkozás célja, hogy a legtöbb eset egy teszten belül kipróbálásra kerüljön, így emiatt az egyik személyt nem került megjelenítésre, hogy bizonyítva legyen, hogy az a rész is megfelelően működik. A második szereplőt az *Y* tengelyen 1 méterrel el lett tolva, amely a lenti 4.1. ábra esetén a Latifa nevű. Hozzá tartozik a 20-as és a 21-es azonosító, melyek közül az első a pozíciót, a második pedig a nézés irányát határozta meg. Bár mind a két tag meg volt adva, mégis a karakter a kamera felé nézett. Ennek oka, hogy amennyiben a két tag közötti eltérés egy bizonyos szint alatt van, akkor azt nem veszi figyelembe, mivel, ha ezzel is számolna a rendszer, akkor előfordulhat olyan, hogy két egymáshoz közeli tagből származó zaj miatt a 3D-s figura folyamatosan ugrálna.



4.1. ábra: Pozíciók (jobb) és a hozzá tartozó megjelenített szereplők (bal)

Ezen felül hozzáadásra került még egy másik egyén is, melyet a 22-es és a 23-as tag szimbolizált. Itt a pozíciót szintén az első, míg az elforgást a második pont jellemezte. Azonban mivel ebben az esetben ezek között 1 méter eltérés volt, itt az elforgást már megjelenítette az alkalmazás, amiatt látszódik, hogy a jobb oldali karakter (Yunyun)

jobbra néz. Így a teszt során bebizonyosodott, hogy amennyiben egy szereplő mellé nem adunk meg taget, akkor az nem jelenik meg, amennyiben egy, vagy pedig két taget, de egymáshoz nagyon közel adunk meg, akkor a karakter bár megjelenik, de minden esetben a kamera irányába fog nézni, amennyiben pedig a két tag között az eltérés is elég nagy, akkor abból az irányt a megfelelő módon meghatározza és megjeleníti.

Ezen felül tesztelésre került a két rendszer közötti késleltetés is, mely során elmentésre került két időbélyeg, egy a Marvelmindból származó adat elküldést megelőzően, valamint ugyanehhez az adathoz egy másik, amikor azt a kliens alkalmazás azt az értéket megkapta. Annak érdekében, hogy a késleltetést ténylegesen le lehessen mérni, nem került felhasználásra a beltéri helymeghatározó. Ebben az esetben a pozíció értékeket a Marvelmind alkalmazás tesztelési üzemmódjával lehetett változtatni és amennyiben megjelent a módosított érték a kliens applikáció kiírásai között, akkor a két időbélyeg elmentésre került. Ezeket az elmentett elemeket mutatja a lenti 4.2. táblázat.

4.2. táblázat: Késleltetés a kliens eszköz és a Marvelmindból megjelenő adat között

Elküldés másodperc	Elküldés milliszekundum	Fogadás másodperc	Fogadás milliszekundum	Különbség
20	151	20	789	638
33	402	33	995	593
58	743	59	318	575

A fenti táblázatban látszódik, hogy az eltérés a két időpont között a legtöbb esetben 600 milliszekundum körül található. Ennek bizonyos hányada abból származik, hogy a Unity alkalmazás nem folyamatosan tölti le az adatokat a szerverről, hanem minden 200 milliszekundumban megpróbálja azokat kiolvasni. Emiatt a késleltetés, ha a szerveren azonnal megjelenne az adat, átlagosan ennek felével érne fel. Ezen felül lép fel, hogy a két végpont – mind a szerver és kliens, mind pedig a Marvelmind és szerver – között az adat a vezeték nélküli hálózaton keresztül kerül továbbításra, így ez további időtartamot visz a rendszerbe.

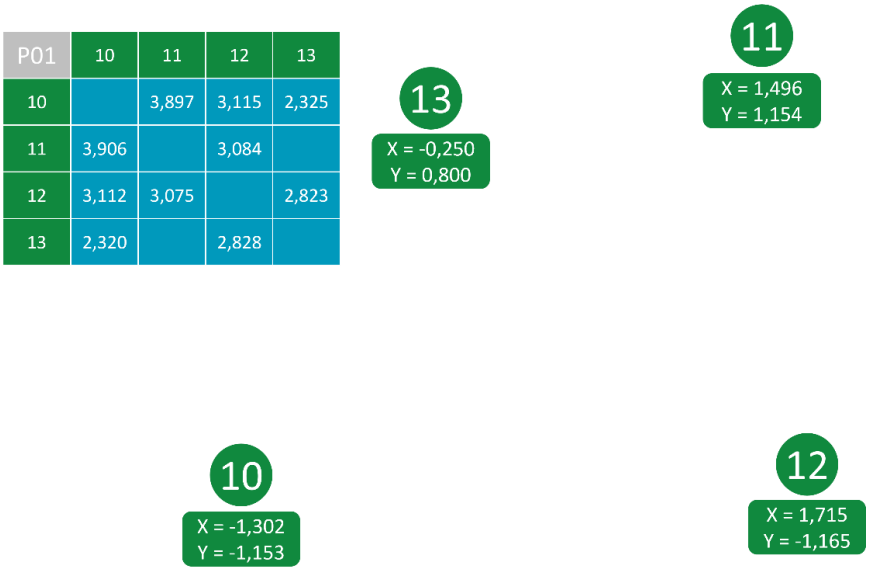
Bár a fenti érték nem egy elhanyagolható mértékű késleltetés, a megoldás során az algoritmusok és metódusok optimalizálása még nem történt meg, így ezzel egy bizonyos fokig csökkenthető ez az időtartam. A fenti probléma redukálására egy másik megvalósítás lehet, amennyiben a Marvelmind feldolgozását végző programot és a szerver futtatását azonos eszközre helyezzük, vagy amennyiben ez nem megvalósítható, úgy a két eltérő komponens vezetékes hálózattal kötjük össze, hogy az adat a lehető leggyorsabban feltöltésre kerülhessen. Amennyiben ez sem csökkentené az átfutási időt, akkor lehetséges, hogy az elmúlt két pozíció közötti értékkel egy sebességet számítunk

és ezzel az értékkel az eltelt idő függvényében interpolálunk. Ezzel a késleltetést le lehet csökkenteni a kliens alkalmazásból fellépő komponensre. Ennek hátránya, hogy amennyiben a becslés hibás, akkor lehetséges, hogy nem megfelelő adat kerül kirajzolásra a fent meghatározott 600 milliszekundumos időközben. Ez viszont idővel korrigálásra kerül, amikor is egy új adat érkezik a helymeghatározó rendszerből, mivel ekkor minden esetben felülírnánk a becsült értéket a ténylegesen mért adatokkal.

4.1.2 Tesztelés beltéri helymeghatározóval

Ezt követő lépésként a beltéri helymeghatározóval való tesztelés következett. Ehhez első lépésként ki kellett alakítani egy megfelelő környezetet, ahol a rendszer megfelelően tudott működni. Ennek alapja, hogy le kellett helyezni fix pozíciójú tageket. Ezek azok a tagek melyek a működés során nem mozognak, hanem a kialakítás alatt felkerülnek a falra, állványzatra és a teljes folyamat közben ottmaradnak. Hogy a beltéri helymeghatározás működhessen minden ilyen jeladó között meg kellett határozni a távolságokat. Ezt megtenni azonban elég egyszer, mivel ez a későbbiekben már nem változik.

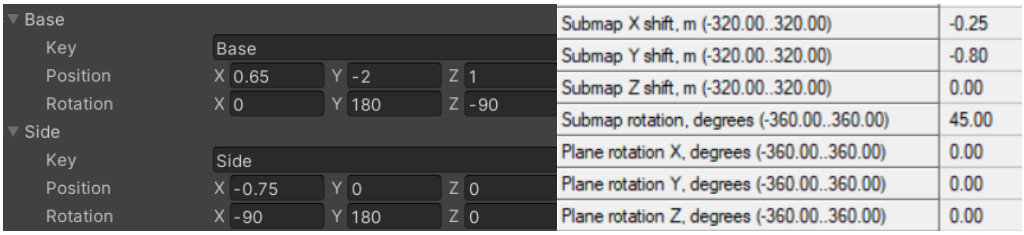
Ennek azonban előfeltétele, hogy a programból pontosan ugyanazt a verziót használjuk, mint amilyen verziójú a jeladó és a modem. Amennyiben ez nem állna fent, akkor vagy a programból kell régebbi változatot letölteni, vagy pedig frissíteni kell az egyes beaconök verzióját. A tesztelés előtti napokban jelent meg a Marvelmind esetén a legújabb 7.0-ás verzió, így a folyamat itt is az összes eszköz új verzióra való frissítésével kezdődött. Amennyiben ez sikerült, akkor kezdődhetett a távolságok meghatározása. Ehhez a Marvelmind által kiadott program használható, melyben amennyiben bekapcsoljuk az érzékelőket, akkor automata módon megkezdődik a köztük lévő távolság felderítése.



4.2. ábra: Térkép és a hozzá tartozó távolság mátrix

Ezt a lépést mutatja a 4.2. ábra, ahol látható, hogy a térképen található pontok között elkezdi meghatározni a távolságot. Amennyiben az összes tag a talaj szintjén helyezkedik el, nem is kell módosítani a folyamatot, de amennyiben ez nem valósul meg, úgy minden fixen lehelyezett taghez be kell írni a magasságot, ahová le lett helyezve. Ehhez az alkalmazás nem biztosít semmit, viszont a feladat egy mérőszalag segítségével könnyedén kivitelezhető. A távolság meghatározása során mind a két irányt kiszámítja a rendszer, azaz külön meghatározza, hogy milyen távol helyezkedik el a 10-as a 11-es tagtől és viszont. Ennek szerepe, hogy ez biztonsági lépést ad a rendszernek, hogy amennyiben ezek között az eltérés túl nagy, akkor a megfigyelés nem elég pontos, célszerű az elemeket úgy átrendezni, hogy azok egymásra jobban rálássanak.

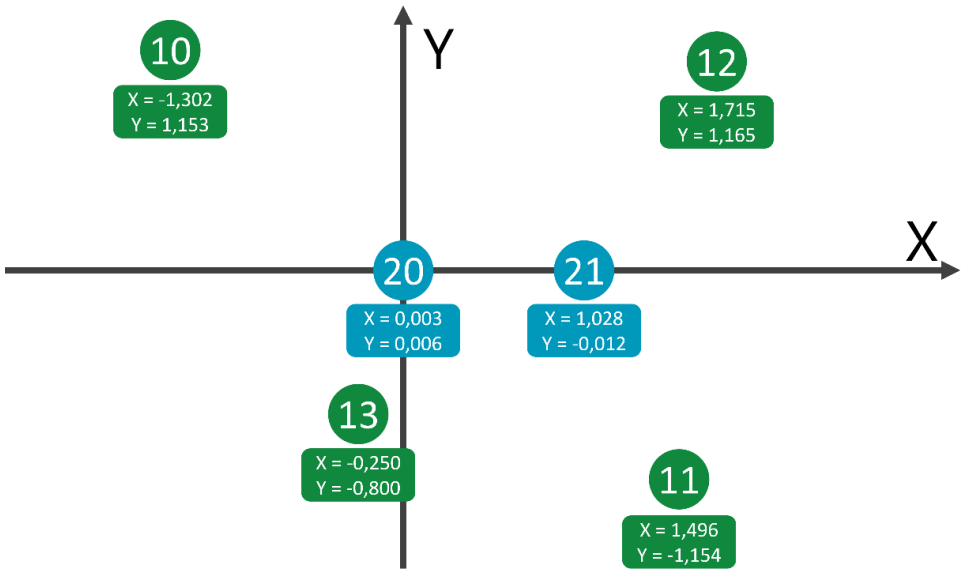
Ezen felül amennyiben ez nem megvalósítható, úgy lehetőség van arra is, hogy egy hibás távolságmérést is elfogadjunk, amennyiben valamilyen másik megoldással megbizonyosodtunk arról, hogy mekkora a helyes távolság. Ilyenkor ki lehet választani az egyik mért értéket, vagy pedig megadhatunk egy azoktól eltérő távolságot is. Ezt követően ezt az elfogadott párt már nem módosítja az alkalmazás. Amennyiben minden tagnél elfogadtunk egy értéket, vagy pedig a két mért érték megegyezik, akkor lehetőségünk van a térképet lefagyasztani, azaz elérni azt, hogy onnantól ne próbálja a fix pontú tagek között a távolságot meghatározni, a mátrixot frissíteni, mivel azok egyszer már sikeresen meg lettek állapítva, innentől elég csak a mozgó tageket figyelembe venni.



4.3. ábra: A nulla pozíció megadása Unityben (balra), Marvelmind Dashboardban (jobb)

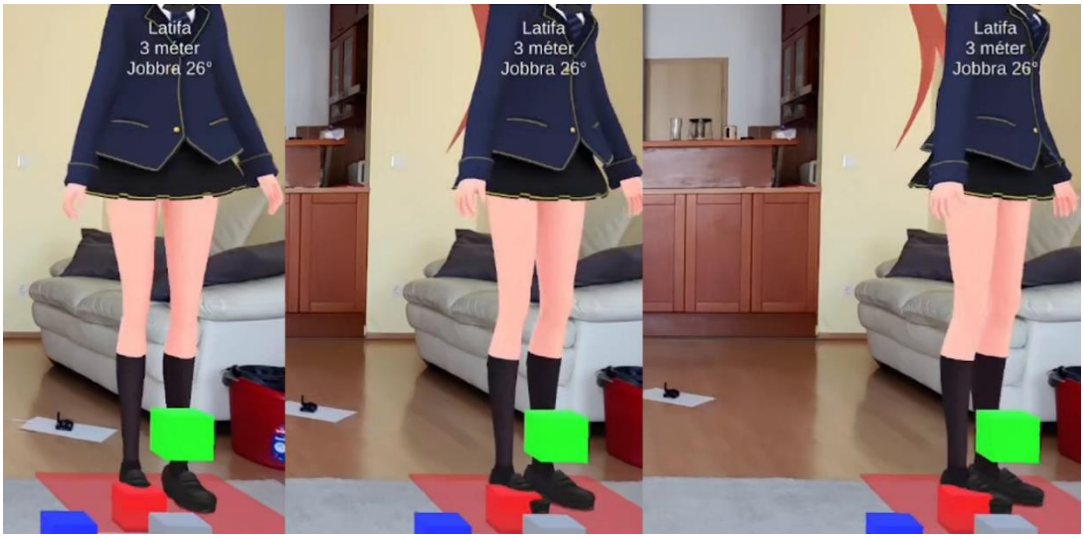
Ezt követő lépésként a két koordináta-rendszert kellett egymással megegyeztetni. Ehhez két opciónk is van, melyek közül az első, hogy megadjuk az AR alkalmazáson belül, hogy kiindulópontot mennyivel kell elmozgatni ahhoz, hogy az megegyezzen a beltéri helymeghatározóval. Ennek megoldása megegyezik a „3.2.1 Közös origó megadása” fejezetben bemutatott elvvel. Ezen felül egy másik opció, hogy a Marvelmind pozíciókat, azaz a hozzáadott alkalmazásban a teljes térképet mozgatjuk. Ennek ezen felül más szerepe is van. A rendszer alapvetően egy térképen egyszerre csak 4 fix pontú markert képes elmenteni. Minden ilyen négyeshez egy úgynevezett altérképet kell megadni és hogy ezeknek az orientációja és pozíciója megegyezzenek egymással, lehetséges a térképet a fenti megoldással korrigálni. A tesztelés során ennek keveréke került használatra, azaz meghatározásra került egy pont a szobában, ami a kiindulási pozíciót jelölte, oda lehelyezésre került egy tag és a megfelelő forgatásokkal és

elmozgásokkal Marvelmind koordináta-rendszere eltolásra került ebbe a pozícióba (4.3. ábra).



4.4. ábra: A kialakított mérési rendszer

Ezt követően a tagek már megfelelő módon követhetőek voltak. A folyamat alatt csak 6 darab tag állt rendelkezésre, ezért a tesztelés során egyetlen karakter követése volt a cél, valamint annak eldöntése, hogy mennyire jelentkezik a beltéri helymeghatározóból származó zaj a rendszerben. Így a felépített tesztelési rendszerben 4 darab fix pozíciójú tag került elhelyezésre, mely meghatározta a felismerendő területet. Ezen felül pedig két mozgó koordináta is lehelyezésre került. Ezek a 20-as és a 21-es azonosítókkal rendelkeztek, így az 4.1. táblázat első két sora került itt is felhasználásra. A cél így Latifa megjelenítése volt, amennyiben két taggel rendelkeztünk. Az így kialakított mérési rendszert a 4.4. ábra mutatja.



4.5. ábra: A forgás tesztelése. Kiindulás (bal), félúton (közép), végezetül (jobb)

Ezt követően következett a tényleges tesztelés. Itt a tagek számából következően két dolog került kipróbálásra, melyek közül az első, hogy amennyiben a pozíciót meghatározó tag fixen egy helyen van, csak a forgást meghatározó mozog, akkor mennyire képes azt a forgást valós időben lekövetni és mennyire pontos. Ehhez a 20-as marker lehelyezésre került egy fix pontba. A másik jeladónak az egyenletes sebességgel történő mozgatását mutatja az 4.5. ábra. Látszódik, hogy a karakter képes lekövetni a forgást és mindig olyan irányba néz, hogy az megegyezzen a várttal.

4.6. ábra: A mozgás tesztelése. Kiindulás (bal), végső állapot (jobb) és köztes állapotok (középső két ábra)

Ezt követő lépésként a mozgás kipróbálása következett. A két teszt közül ez volt az érzékenyebb, mivel a beltéri helymeghatározóból származó zaj jobban befolyásolja a mozgást, mivel itt előfordulhat olyan, hogy a karakter nem a jó helyen áll, ami a tagnek köszönhetően rendkívül feltűnő. Ebben az esetben a fix tag a forgásért felelős volt, míg a pozícióért felelős került mozgatásra, de ennek a fentitől eltérő útvonal történt a megvalósítása. Ennek a módosításnak célja, hogy egymás mellett is legyen a két tag, mivel az lett megfigyelve, hogy amennyiben két beacon egymáshoz nagyon közel kerül, akkor egymás jelének bezavarhatnak és ez a karakter ugrálásához vezethet. Ennek a folyamatnak a megvalósítását mutatja a 4.6. ábra, ahol a figura mozgását követhetjük. Látható, hogy a karakter mind a mozgást, mind pedig a forgást a megfelelő módon követi és amennyiben a középső taghez közel helyezkedünk el, úgy sem zavarja be egymást a kettő.

5 Eredmények értékelése

Az alkalmazás a fent részletezett módon elkészült, mely során megvalósult egy olyan applikáció mely képes volt valós időben egy beltéri helymeghatározóból származó pozícióadatot felhasználva egy területen belül a személyek megjelenítésére, kiegészítve további információkkal, melyekkel a rendszer használata nélkül a felhasználó nem rendelkezne. Ilyen adatnak tekinthető a megfigyelt személyek felhasználótól való távolsága és a vele bezárt szöge, valamint annak eldöntése, hogy a felhasználó ki van-e takarva. Továbbá ide tartozhat a befoglaló téglatest, mely egy biztonsági zónát jelöl, valamint a személy magassága és neve is. Azonban ennek az alkalmazásnak a megvalósítása egy több lépcsős folyamat volt, melynek minden eleme kihagyhatatlan, annak érdekében, hogy a megvalósított applikáció a megfelelő módon és elvárt pontossággal működjön.

Ehhez első lépésként bemutatásra került a tényleges cél, mely összefoglalta és definiálta, hogy a diplomaterv folyamán milyen feladatok elvégzésére volt szükség. Ezt követően rövid áttekintés következett, melyben bemutatásra került néhány már napjainkban is működő AR alkalmazás. Ezek két eltérő csoportban kerültek megjelenítésre, melyek közül az első halmaz elemei az átlagfelhasználó számára is elérhető megvalósítások, míg a másik csoport tartalmazza az iparban használt rendszereket. Ezek eredményeképpen kiderült, hogy a legtöbb rendszer használ valami féle nyomkövető technikát, így a jelenlegi megoldásban is szükség lesz erre.

Ebből kifolyólag a rákövetkező fejezet a beltéri helymeghatározóban elérhető lehetőségekkel és az abból származó adatok feldolgozásával és tárolásával foglalkozott. Ehhez először röviden át lettek tekintve a jelenlegi megoldások, melyeknek az előnyeit és a hátrányit is figyelembe véve kiválasztásra került egy olyan rendszer, mellyel lehetséges volt egy zárt helyiségben megfelelő pontossággal és sebességgel objektumok követése. Ehhez elkészült egy segítő megoldás, melynek használatával lehetőség volt a helymeghatározó rendszerben található adatok kiolvasására, azok megfelelő formátumba alakítására, ennek a struktúrának a szerver felé való továbbküldésére, valamint szükség esetén fájlba írására is.

Ehhez elkészült a szerver komponens is, mely az ily módon előállított adatokat képes volt fogadni és azokat eltárolni további felhasználásra. Itt a pozíciókon kívül azonban szükségessé vált további információkat is eltárolni a rendszerhez, tehát a szerver kiegészült egy adatbázis komponenssel is, mely alkalmas volt egyszerűbb adatok NoSQL formában való tárolására. Itt került elmentése, hogy mely tagek mely megfigyelt személyekhez, objektumokhoz tartozik, valamint az egyes karakterek milyen azonosítóval jelenjenek meg a kliens alkalmazásban.

Ezt követő fejezet során bemutatásra került, hogy a kliens alkalmazás megvalósítására milyen platformon van lehetőség és hogy részben a tesztelési eszköz miatt melyik került

kiválasztásra a felsorolt opciók közül. Ezt követően elkezdődött a kliens alkalmazás implementálása annak minden komponensével együtt. Itt megírásra került, hogy:

- a két eltérő eszközrendszernek a koordináta-rendszere mily módon tudott azonos értékeket felvenni,
- a beltéri helymeghatározóból származó adatok zajossága miatt milyen további lépésekre volt szükség, hogy a kapott adatokat fel lehessen használni,
- milyen lehetőségek voltak a karakter megjelenítésére és hogyan jött létre a fent látható szereplő,
- milyen megoldásokkal és hogyan kellett kiegészíteni a már épülő alkalmazást, hogy támogassa, hogy a megfigyelt szereplő nem csak pozícióval, hanem orientációval is rendelkezik,
- miként került megvalósításra egy olyan szöveg kiírása a felhasználó felé, hogy az minden esetben látható legyen a számára, azonban mégis kövesse a helymeghatározó rendszerrel követendő objektum pozícióját,
- a tesztelés és a fejlesztés során milyen további megoldások biztosítottak lehetőséget, hogy az alkalmazást valós időben tesztelni lehessen és a folyamat során megjelenő hibák átlátható módon megjelenítésre kerüljenek,
- milyen további megoldásokat tartalmaz a rendszer annak érdekében, hogy a felhasználók számára könnyebbé tegye a munkát

A fent részletezett megoldások megvalósítását követően egy tesztelési szakasz következett, melyben a rendszer egészében került kipróbálásra. Ez a folyamat két fázisban történ, melyben először a beltéri helymeghatározó nélkül került kipróbálásra a három applikáció. Ennek célja az egyszerűbb mozgások és elhelyezkedések kipróbálása volt, anélkül, hogy egy külső rendszerre bíznánk a pozíciók meghatározását. Második fázisként kiépítésre került egy beltéri helymeghatározó rendszer és ezalatt bemutatásra került ennek folyamata, hogy a térkép mily módon hozható létre jeladó négyesekből.

Ezt követően a tényleges rendszerteszt következett, ahol már minden komponens a helyén volt és egymással kommunikált. Ezen vizsgálat alatt szintén a mozgás és a forgás került górcső alá, azonban azok helyes működésének kipróbálása helyett (ez már a korábbiakban bizonyítást nyert), a vizsgálat célja az volt, hogy megállapítsa, hogy amennyiben ezeket a beltéri helymeghatározóhoz tartozó jeladókat mozgatjuk, akkor a megjelenítés megfelelő sebességgel és pontossággal követi azt. A teszt eredményét követően nem volt más hátra a rendszer vizsgálatából, mint a következtetések levonása és az esetlegesen előforduló hibák javítása.

5.1 Fejlesztési javaslatok

Azonban az alkalmazásban vannak még fejlesztése lehetőségek. Ilyenek lehetnek valamilyen kisebb hiányosságok kijavítása, de akár új funkciók teljes implementálása is. A következő alfejezetben bemutatásra kerül néhány ilyen gondolat, hogy az alkalmazást milyen módon lehetne a jövőben továbbfejleszteni, hogy az kibővített lehetőségekkel működjön.

Mozgás követés és forgás

Ezek közül a módosítások közül az első a mozgás követése. Ehhez mind a szerver, mind pedig a kliens alkalmazás módosítására szükség van. Ennek célja, hogy egy-egy karakterhez nem csak a jelenlegi pozíció kerülne elmentésre, hanem azon felül tárolnánk a korábbi elemeket is, így lehetőségünk lenne a megfigyel objektum mozgásának meghatározására. Amennyiben vesszük a korábbi értékeket és azok között különbséget számítunk, akkor egy olyan vektort kapunk, melynek iránya megegyezik a követendő személynek a mozgásirányával, nagysága pedig a sebességével. Ez csak akkor igaz, amennyiben egyenletesen mintavételezünk, és a Marvelmind szerencsére így működik. Ez a megoldás kiegészíthető azzal, hogy a mozgást nem csak szimpla összeadással határozzuk, meg, hanem valamilyen szabályzást végzünk vele, például egy PID szabályzó segítségével. Így figyelembe vehető az is, hogy a személy éppen gyorsít, vagy pedig lassít és ezzel előre becsülhetjük egy bizonyos idő múlva a megfigyelt szereplő hol fog elhelyezkedni a térben.

A megoldás célja, hogy ezt megjelenítsük a talaj síkjára vetítve. Így amennyiben tudjuk egy jármű mozgásirányát, akkor a közelben lévő felhasználókat értesíthetjük, hogy feléjük valamilyen irányból egy jármű közelit. Ezen felül a mozgás becslése arra is használható, hogy a már amúgy is megjelenített orientációt pontosítsuk, úgy, hogy az emberek a legtöbb esetben arra fordulnak amerre haladnak. Így amennyiben ezek megegyeznek, akkor a két mért érték segítségével pontosítható a nézés iránya. Azonban itt figyelembe kell venni, azokat az esetteket, amikor ez nem igaz, például amennyiben a megfigyelt személy éppen áll, akkor a mozgás vektor csak a zajt fogja tartalmazni, azonban ebben az esetben ennek a vektornak a mérete csupán néhány milliméter. Ez tehát kizárható úgy, hogy csak egy bizonyos méretnél nagyobb vektorok játszanak szerepet a mozgás meghatározásában. Amennyiben azonban nem emberekről beszélünk a fenti megoldást nem alkalmazható, mivel egy jármű esetén gyakran elfordul, hogy tolat.

Útvonal ajánlása

Amennyiben a vállalat rendelkezik egy raktárral és hozzá egy vállalatirányítási rendszerrel és tudott az, hogy melyik rakomány hol helyezkedik el a raktárban, akkor lehetséges egy térkép felépítése a raktár fölé, melynek segítésével az ott dolgozó alkalmazottaknak útvonalakat lehet ajánlani, hogy hogyan tud a leghamarabb eljutni oda, figyelembe véve, hogy a raktár mely részein történik jelenleg árukiszedés, hol van

nagyobb forgalom. Hasonlóan a Google Maps megoldásához (lásd.: 1.2 Jelenlegi megoldások), a földre lehetne festeni egy irányt mutató nyilat és a felhasználót odavezetni a tárolóhoz.

Ennek kiegészítése lehet, hogy amennyiben a raktár területén belül önvezető járművek is közlekednek, akkor azoknak a már megtervezett útvonalának körülbelüli képét a felhasználóknak a talajra lehetne rajzolni. Így amennyiben jön az AGV, akkor a felhasználó tudni fogja, hogy ne álljon be annak útjába, mivel úgy felfogja a forgalmat. Ez a korábbi javaslattal együtt alkalmazva képes lehet arra, hogy a felhasználó tudja, hogy mikor kell félreállni az AGV útjából és hol is van pontosan az az út, ahol ő nem lehet.

Szintek és régiók

Az alkalmazás jelen pillanatba csupán egy szint kezelésére képes, azaz amennyiben a raktár rendelkezik egy galéria résszel, akkor abban az esetben az ott tartózkodó emberek is a talaj szintjén helyezkednek el. Ennek oka, hogy a Marvelmindból a pontosabb 2D-s érzékelés érdekében kikapcsolásra került a magasság figyelése is. Annak megvalósítására, hogy a felhasználókat megjeleníthessük mindenhol elég a beltéri helymeghatározó esetén bekapcsolni a 3D-s követést, ami miatt viszont a 2D-s pontosság csökkenhet. Ezután az alkalmazások minden karakterhez figyelembe veszik a magasság értéket is úgy, hogy abból a tag a talajtól vett magasságát levonjuk. Így a 3D-s térben a szereplőt bárhol követhetjük. A két tag hatására az olyan események is érzékelhetők, amely során a felhasználó lehajol.

Ez kiegészíthető, hogy az egyes területeket elmentjük egy térképen, akár címkével, például „galéria” és egy-egy ilyen régióhoz hozzárendelünk egy magasságot. Így a *Z* érték beli zaj teljes egészében megszüntethető, hiszen nem a kapott értéket, hanem az elmentett magasságot használnánk. Az egyetlen kivétel ez alól a lépcső, hiszen ott nem lehetséges meghatározni egy fix magasságot, hanem ott ténylegesen követni kell a kapott koordináta értéket. A címkéket pedig ezen felül az egyes személyek mellett meg is lehet jeleníteni, így a karaktereket pozíciók szerint bele lehet helyezni az egyes szobákba, mellyel a felhasználó pontosan tudja, hogy hol keresse a másik embert.

Irodalomjegyzék

[1] M. Ranieri, „A new sense of direction with Live View,” Google Maps, 01 október 2020. [Online]. Available: https://blog.google/products/maps/new-sense-direction-live-view/. [Hozzáférés dátuma: 18 november 2021].

[2] M. Cosimano, „Review: Pokemon Go,” Destructoid, 12 július 2016. [Online]. Available: https://www.destructoid.com/reviews/review-pokemon-go/. [Hozzáférés dátuma: 18 november 2021].

[3] M. Peckham, „Review: 'Pokémon Go’ Is an Ingenious Idea With Too Many Rough Edges,” Time, 12 június 2016. [Online]. Available: https://time.com/4401279/pokemon-go-review/. [Hozzáférés dátuma: 18 november 2021].

[4] Marxent Labs, „Augmented Reality Shopping Apps – AR Furniture,” Marxent Labs, [Online]. Available: https://www.marxentlabs.com/products/augmented-reality-furniture-apps/. [Hozzáférés dátuma: 18 november 2021].

[5] TUP, „The picking method Pick-by-Vision,” Dr. Thomas + Partner GmbH & Co., 12 szeptember 2017. [Online]. Available: https://logistikknowhow.com/en/warehouse-automation/the-picking-type-pick-by-vision/. [Hozzáférés dátuma: 18 november 2021].

[6] T. Gál, G. Dr. Grasselli, L. Dr. Nagy, E. Nyilas, Z. Tarján, L. Dr Terjék és A. Dr. Vántus, „A komissiózás,” Debreceni Egyetem, [Online]. Available: https://www.agr.unideb.hu/ebook/logisztika/a_komissizs.html. [Hozzáférés dátuma: 18 november 2021].

[7] H. a. Y. S. H. Koyuncu, „A Survey of Indoor Positioning and Object Locating Systems,” *IJCSNS,* %1. kötet10.5, 2010.

[8] Sewio, „Indoor Location Tracking and Monitoring,” Sewio, [Online]. Available: https://www.sewio.net/people-employee-indoor-location-tracking-and-monitoring/#Geofencing. [Hozzáférés dátuma: 12 május 2021].

[9] Marvelmind, „Marvelmind,” Marvelmind Robotics, [Online]. Available: https://marvelmind.com. [Hozzáférés dátuma: 18 november 2021].

[10] Marvelmind Robitics, „Architectures Comparison,” Marvelmind Robitics, 11 augusztus 2020. [Online]. Available: https://marvelmind.com/pics/architectures_comparison.pdf. [Hozzáférés dátuma: 12 május 2021].

[11] Marvelmind Robotics, „Placement Manual,” Marvelmind Robotics, 13 július 2020. [Online]. Available: https://marvelmind.com/pics/Marvelmind_Robotics_ENG_placement_manual.pdf. [Hozzáférés dátuma: 12 május 2021].

[12] LiteDB team, „Overview,” LiteDB, [Online]. Available: https://www.litedb.org/docs/. [Hozzáférés dátuma: 18 november 2021].

[13] M. Leon, „Ultimate AR Comparison Guide: ARKit vs ARCore vs Vuforia vs AR Foundation,” Circuit Stream, 3 szeptember 2021. [Online]. Available: https://circuitstream.com/blog/augmented-reality-guide/. [Hozzáférés dátuma: 18 november 2021].

[14] AppleInsider, „ARKit,” AppleInsider, 2 április 2021. [Online]. Available: https://appleinsider.com/inside/arkit. [Hozzáférés dátuma: 18 november 2021].

[15] Google Inc., „Overview of ARCore and supported development environments,” Google, 01 november 2021. [Online]. Available: https://developers.google.com/ar/develop. [Hozzáférés dátuma: 18 november 2021].

[16] Vuforia Engine Team, „Vuforia Engine Developer Portal,” PTC, Inc., [Online]. Available: https://developer.vuforia.com. [Hozzáférés dátuma: 18 november 2021].

[17] Unity Technologies, „About AR Foundation,” Unity Technologies, 12 október 2021. [Online]. Available: https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@4.2/manual/index.html. [Hozzáférés dátuma: 18 november 2021].

[18] W. Ferenc, „Ortogonalizáció,” Budapesti Műszaki és Gazdaságtudományi Egyetem, 22 március 2016. [Online]. Available: http://math.bme.hu/~wettl/okt/linalg/2016/handout_e05_16fm_ortogonalizacio.pdf. [Hozzáférés dátuma: 18 november 2021].

Ábrajegyzék