

Pannon Egyetem
Műszaki Informatikai Kar
Alkalmazott Informatika Tanszék (Nagykanizsa)
Mérnökinformatikus MSc

DIPLOMADOLGOZAT

Kiterjesztett valóság integrálása ipari folyamatokba

Mihalics Bálint

Témavezető: Dr. Jaskó Szilárd

2023



PANNON EGYETEM

MŰSZAKI INFORMATIKAI KAR

Mérnökinformatikus MSc szak

Veszprém, 2023. március 29.

DIPLOMADOLGOZAT TÉMAKIÍRÁS

Mihalics Bálint

Mérnökinformatikus MSc szakos hallgató részére

Kiterjesztett valóság integrálása ipari folyamatokba

Témavezető: Dr. Jaskó Szilárd, tanszékvezető egyetemi docens

A feladat leírása:

Cél egy olyan kiterjesztett valóság (AR) alapú szoftverrendszer kifejlesztése, amely integrálásra kerül egy tetszőlegesen kiválasztott demó vagy valós ipari folyamatba. A szoftverrendszernek képesnek kell lennie az ipari folyamathoz kapcsolódó digitalizált adatok valós idejű szemléletes megjelenítésére.

Feladatkiírás:

- Tervezze meg a rendszer működését!
- Tervezze meg és specifikálja a rendszert és komponenseit!
- Megvalósítás, prototípus, és teszt környezet elkészítése.
- Tesztelje a rendszer komponensek működését!
- Tesztelje az elkészült rendszert!

Dr. Magyar Attila
egyetemi docens
szakfelelős

Dr. Jaskó Szilárd
egyetemi docens
témavezető

Hallgatói nyilatkozat

Alulírott Mihalics Bálint hallgató kijelentem, hogy a dolgozatot a Pannon Egyetem Alkalmazott Informatika Tanszékén (Nagykanizsa) készítettem a okleveles mérnökinformatikus végzettség megszerzése érdekében.

Kijelentem, hogy a dolgozatban lévő érdemi rész saját munkám eredménye, az érdemi részen kívül csak a hivatkozott forrásokat (szakirodalom, eszközök stb.) használtam fel.

Tudomásul veszem, hogy a dolgozatban foglalt eredményeket a Pannon Egyetem, valamint a feladatot kiíró szervezeti egység saját céljaira szabadon felhasználhatja.

Dátum: Nagykanizsa, 2023.05.06.




Mihalics Bálint

Témavezetői nyilatkozat

Alulírott Dr. Jaskó Szilárd témavezető kijelentem, hogy a dolgozatot Mihalics Bálint a Pannon Egyetem Alkalmazott Informatikai Tanszék (Nagykanizsa) készítette okleveles mérnökinformatikus végzettség megszerzése érdekében.

Kijelentem, hogy a dolgozat védelemre bocsátását engedélyezem.

Dátum: Nagykanizsa, 2023.05.06



Dr. Jaskó Szilárd

Köszönetnyilvánítás

Szeretném megköszönni kedves, szerető és türelmes feleségemnek Szilvinek, drága anyukámnak és édesapámnak a lelki és anyagi támogatást, hogy ez a diplomadolgozat létrejöhessen. Dr. Jaskó Szilárdnak pedig a bátorítást, hogy elkezdjem az MSc-t és a tudományos és szakmai segítséget, hogy be is tudjam fejezni. Kisfiam az egyetem elkezdésekor született, nem volt könnyű egy kisbabával karöltve végezni, munka és család mellett, de a szeretete és csillogó szeme mindig erőt adott, hogy csináljam tovább és büszke lehessen rám. Ezért ezt a dolgozatot Ágoston kisfiamnak ajánlom.

Tartalmi összefoglaló

Feladatom volt egy ipari folyamatokat támogató kiterjesztett valóságos alkalmazást készítése, amit a tényleges gyártásban vagy az oktatásban és marketingben lehet felhasználni.

Létrehoztam egy adatbázist, az azt kezelő webszervert és egy adat szervert, ami API-ként kommunikál a kiterjesztett valóság Android alkalmazással. Ez program képes PLC-ket vezérelni és adatokat megjeleníteni róluk, illetve egy virtuális gyárterületet és az azokon lévő robotokat és azokról információkat megjeleníteni valós időben.

A feladat megoldásához Unity játékfejlesztő motort és Visual Studio IDE-et használtam, C# programnyelven programoztam az Andorid alkalmazást és a szervereket is. A PLC és robot eléréshez kiegészítő könyvtárakat használtam, akár csak az alkalmazás és a szerver adatkapcsolathoz is.

Sikerült lekérnem és módosítanom is az alkalmazásba kiterjesztett valósággal kivetített felületről a PLC adatait és a robotokat megjeleníteni egy virtuális síkon a valós térben. Mind ehhez az adatokat az adatbázis és a köztes szoftveres felületek biztosítják.

Kulcsszavak: Kiterjesztett valóság, ABB, Siemens, PLC, Robot, Ipar 4.0, Unity

Abstract

My task was to create an augmented reality application supporting industrial processes that could be used in actual manufacturing or in education and marketing.

I created a database, a web server to manage it and a data server that communicates as an API with the augmented reality Android application. This program can control PLCs and display data about them, or a virtual factory floor and the robots on it and display information about them in real time.

To solve the task, I used Unity Game Engine and Visual Studio IDE, programmed the Android application and servers in C#. I used additional libraries to access the PLC and robot, including the data connection between the application and the server.

I was able to retrieve and modify PLC data from the augmented reality projected interface in the application and display the robots on a virtual plane in real time. All this data is provided by the database and middleware interfaces.

keywords: Augmented Reality, ABB, Siemens, PLC, Robot, Industry 4.0, Unity

Tartalomjegyzék

Jelölésjegyzék	11
1. Bevezető.....	12
2. Diplomadolgozat célja és eredményei.....	13
1.1. Kiterjesztett Valóság (AR)	13
1.2. Ipar 4.0 és hasonló technológiák	14
2. Felhasznált programok és technológiák	15
2.1. Rendszerterv	15
2.2. Siemens PLC programozása TIA Portal-lal	16
2.3. Virtuális robotok futtatása ABB RobotStudio-ban.....	20
2.3.1. Robotok felépítése és programozása	20
2.4. Unity játékfejlesztő motor és ARCore.....	23
2.5. Visual Studio és .NET keretrendszer C# nyelvvel	25
2.5.1. Adatbázis	26
2.6. Kiegészítő szoftverek: Blender és Photoshop.....	27
3. Szerverek.....	29
3.1. Adatbázishoz való kapcsolódás	29
3.2. Adatokat gyűjtő és kiszolgáló szerver	29
3.2.1. Adatgyűjtés PLC-ről.....	30
3.2.2. Adatgyűjtés robotról.....	32
3.3. WEB-es felület adatbázis menedzseléshez	34
3.3.1. Telepítés és beállítás.....	34
3.3.2. Gyárterület menüpont.....	35
3.3.3. PLC-k menüpont	39
4. Kiterjesztett valóság alkalmazás.....	41
4.1. Kezelő felület.....	41
4.1.1. Funkcionalitás.....	42
4.1.2. Használati esetek és menü struktúra.....	42
4.2. Program működése	44
4.2.1. Állapotgép bemutatása	44

4.2.2.	Főmenü és beállítások	49
4.2.3.	Eszközök leolvasása és működtetése.....	51
4.2.4.	Virtuális gyárterület.....	54
5.	További célok és fejlesztési lehetőségek.....	57
6.	Összefoglaló.....	58
	Irodalomjegyzék.....	59
	Mellékletek	60
	Ábrajegyzék.....	64

Jelölésjegyzék

- AR: Augmented Reality – Kiterjesztett valóság
- PLC: Programmable Logic Controller– Programozható Logikai Vezérlő
- LINQ: Language Integrated Query – Programnyelvbe Épített Lekérdezések
- DB: DataBlock – adat terület rész PLC-n
- QR: Quick Response – kétdimenziós/mátrix vonalkód
- HMI: Human-Machine Interface - ember-gép felület, kezelőfelület
- SDK: Software Development Kit – Szoftver Fejlesztő Készlet
- API: Application Programming Interface - alkalmazásprogramozási felület
- GO: GameObject – játék objektum Unity-ben

1. Bevezető

Az ipari folyamatos fejlődésen megy keresztül, jelenleg az Ipar 4.0 , vagyis a minél több információ gyűjtése és annak leghatékonyabb felhasználása a legfőbb témakör. A Pannon Egyetem nagykanizsai kampuszán mi is ezen dolgozunk az ipari laborunkban, ahol most már 6 éve dolgozom fejlesztőként. A labor elsődleges célja az oktatási és kutatási olyan ingyenes és ipari technológiák ötvözése, amelyekkel az Ipar 4.0-nak kitűzött céljait meg tudjuk valósítani. Jelen diplomamunkám is egy ilyen apró lépés ezen célok eléréséhez. Az itt ötlött idő alatt tapasztalat és az MSc alatt megszerzett tudást felhasználva készülhetett el a dolgozatom készítése közben megvalósult szoftveres rendszer.

2. Diplomadolgozat célja és eredményei

Diploma dolgozatom célja egy olyan alkalmazás kifejlesztése volt, amelyet ipari folyamatokba vagy oktatásban lehet integrálni. Másik felhasználási területe lehet még marketing bemutatók, ahol gyár vagy oktatási intézmény képességeit lehet vizuálisan reprezentálni. Szerettem volna megvalósítani, hogy QR kód azonosítás után egy megadott PLC kimeneteit lehessen leolvasni és változtatni a programból. Másik célom pedig, hogy robotokat tudjak a virtuális térben, egy gyárterületet reprezentáló felületre kivetíteni. Ha csak szoftveresen létezik a robot akkor hologramként jelenjen meg, ha valós robot akkor pedig mintha igazi gép lenne, úgy jelenjen meg. Ezt az egész területet és a rajta lévő robotokat pedig szerettem volna, ha a valóságos méret mellett a könnyebb átláthatóság miatt kicsinyíteni is lehetne.

Elsődleges felhasználóknak olyan szakemberekre gondoltam, akiknek programozható vezérlőket - röviden PLC-eket - kell karbantartani, hibákat elhárítani vagy egy ezekre a vezérlőre kötött gép működését kell befolyásolnia. Ezek a feladatok természetesen megoldhatóak fizikai eszközökkel is, mint amilyen például egy HMI, ami tulajdonképpen egy kijelző és a hozzá csatlakoztatott vezérlő gombok vagy egy érintőkijelző, de például az alkalmazásom erre is megoldást nyújthat hiszen költséghatékony és sokkal rugalmasabban áttervezhető felületet tud nyújtani, mint egy merev programozású HMI. A gyárterület kivetítése, újonnan felvett szakemberek beoktatását és a gyár területén való könnyebb eligazodást segítheti.

Az oktatásban olyan oktató kollégák vehetik hasznát, akik ipari automatizálást vagy gyártás vezérlést oktatnak vagy akár közvetlenül robotikát és az általam fejlesztett alkalmazással lehetőség van nagy gyárterületek szimulációjára is.

Marketing szempontból pedig cégük profilját vagy oktatási képességeiket bemutató PR szakemberek és oktatók vehetik hasznát, például egy nyílt nap alkalmával.

1.1. Kiterjesztett Valóság (AR)

A kiterjesztett valóság témaköre most nagyon felkapott lett az utóbbi években, köszönhetően annak, hogy ma már egy közép kategóriás telefon is képes a futtatására a megfelelő szoftverekkel. Kezdetben több alternatív AR szoftvercsomagot is megnéztem, és a Vuforia ígéretes volt, ráadásul nincsen limitálva, hogy milyen eszközön képes

futni, de mivel felhő alapú AR technológia, vagyis a számítások egy nagy központi szerveren futnak, folyamatos internetkapcsolatot igényelt volna. Ami még nem is lenne feltétlenül baj, de az ingyenes felhasználáshoz limitálva van az adatforgalom, ami viszont nekem nem tetszett benne.

Az ARCore-t, ami mellett döntöttem a Google fejleszti és az általam elérhető eszközök támogatják is. A teszteket egy Xiaomi Redmi Note 8 Pro-n végeztem, ami megfelel a Google követelményeinek. Ahhoz, hogy működjön a Google által akkreditálnak kell lennie a telefonnak, amit a gyártók a kamera paramétereinek pontos kalibrálásával és egy minimum hardveres követelménnyel tudnak megtenni. Ezt egész pontosan így fogalmazza meg a Google: „Az egyes készülékek tanúsítása során ellenőrizzük a kamera, a mozgásérzékelők és a tervezési architektúra minőségét, hogy a készülék az elvárásoknak megfelelően működjön. Emellett az eszköznek elég erős CPU-val kell rendelkeznie, amely integrálódik a hardvertervezéssel a jó teljesítmény és a hatékony valós idejű számítások biztosítása érdekében” [1].

1.2. Ipar 4.0 és hasonló technológiák

Az intenzív digitalizáció és AR technológiák egyre nagyobb hangsúlyt kapnak az ipar jövőjének számító Ipar 4.0-ban. Minél többféle módon tudunk információt megosztani az azt felhasználó szakemberekkel, annál minőségibb lehet a gyártás. A másik oldal pedig, hogy nagy a forgás a munkaerőpiacon, egyre nehezebb a megfelelő tudással rendelkező embert találni, így a betanítás sebességének növelése kulcsfontosságú lehet egy ipari folyamatban. Ehhez nyújthat segítséget az általam elképzelt alkalmazás, természetesen nem teljesen a jelenlegi állapotában, hanem a benne rejlő potenciállal.

Vannak a piacon ebbe az irányba mutató fejlesztések, de ezek inkább csak a hardverre helyezik a hangsúlyt, ami AR esetén a szemüvegbe vagy sisak vizorjába épített eszközöket fedik le. Szoftverek megvalósítását a megvásárló cégre bízzák és ráadásul ezek a rendszerek zömében zártak.

Az én megoldásom továbbfejlesztéssel szintén használható lenne sztereoszkópikus kijelzéssel, de ez most nem képezte részét a projektnek, bár a fejlesztést elkezdtem ebbe az irányba is.

2. Felhasznált programok és technológiák

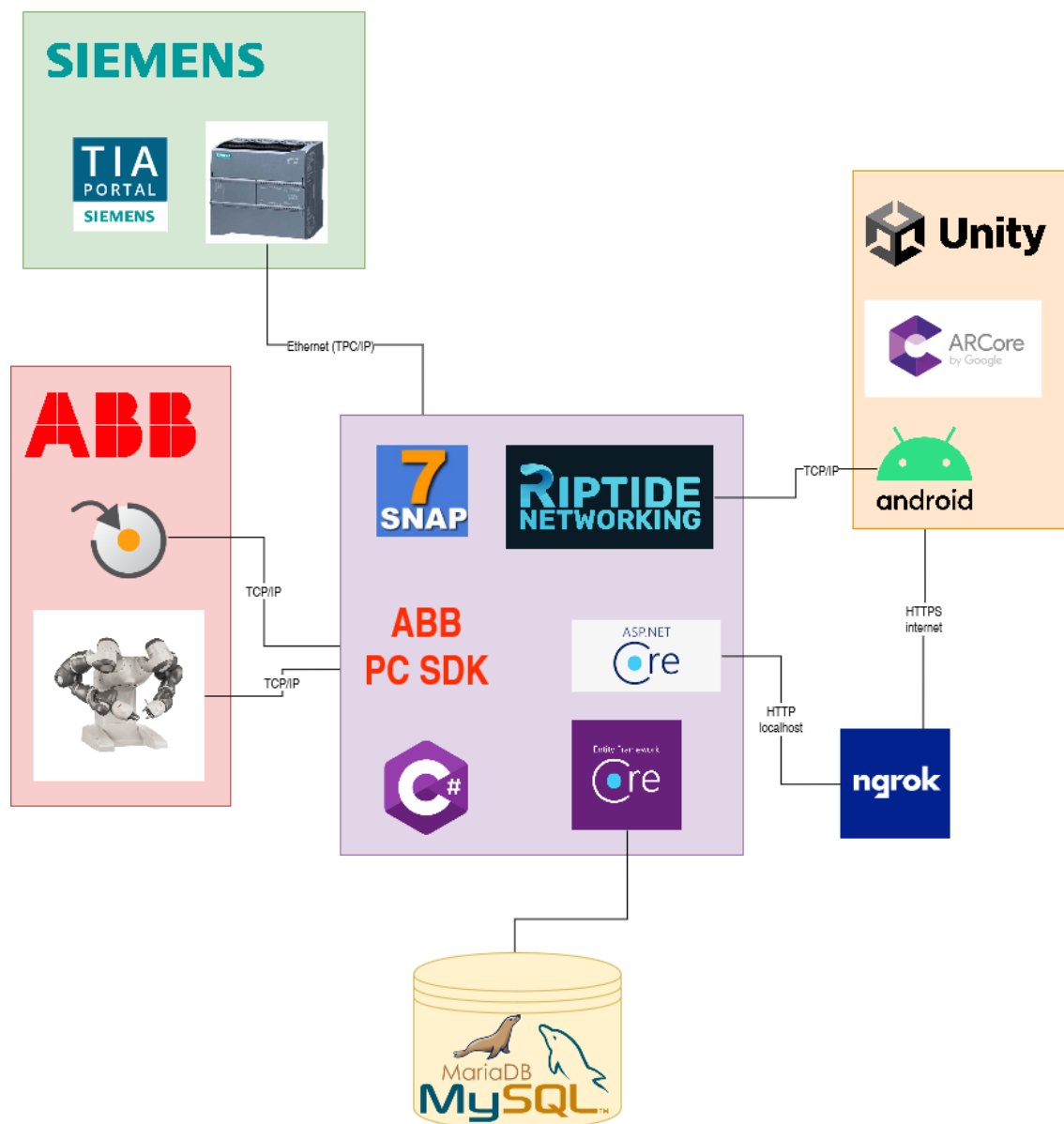
Ehhez a projekthez több különböző ipari szoftvernek és eszköznek kell együtt működnie olyan technológiákkal, amiket alkotóik tervezésükkor nem pont erre a célra találtak fel. Ez adja a szépségét ennek a rendszernek, hiszen jobbára ingyenes vagy az egyetem által oktatási célra beszerzett, de az iparban nagy nevű cégek által biztosított gépek lesznek egy hálózatba kötve, amik az Ipar 4.0 jövőbemutató elképzeléseit már most a jelenben is próbálják, még ha csak kis lépésekkel is de megvalósítani.

Amost következő alfejezetekben, a felhasznált technológiákról esik szó, ahol szükséges a teljes megértéshez ott, rövid elméleti áttekintőt adok, illetve bemutatom, hogy az adott technológiához milyen szoftverek kellenek, azokat, hogyan kell beállítani, hogy a projektben egy egységként üzemelhessenek.

2.1. Rendszerterv

Mielőtt a későbbi fejezetekben részletesen kifejteném, hogy milyen programok és technológiák, hogyan is működnek, ebben részben szeretném bemutatni a rendszer komponenseinek egymáshoz kapcsolódását egy egyszerűen átlátható rendszerterven.

Az alábbi képen a színes dobozokba eltérő technológiák vannak, amik egy része gyártóspecifikus, mint amilyen a Siemens vagy ABB eszközök, míg van, ahol azért került külön egységbe mert az egy specifikus fejlesztőeszközben készült, mint a Visual Studio vagy a Unity. Ezeket a komponenseket mind központi szerverhez kapcsolódnak, ami .Net Core keretrendszerben fut és a kiegészítő könyvtárak segítségével kapcsolódik a többi egységhez. Ez a kapcsolat minden esetben hálózati, különböző szintű protokollokkal megvalósítva a mi a blokkokat összekötő kapcsolatokra került.



2.1 Rendszerterv

2.2. Siemens PLC programozása TIA Portal-lal

A projektem egyik célja, hogy iparban is használt PLC-khez lehessen hozzáilleszteni a mobilos alkalmazást, ami egy QR kód leolvasásával virtuális HMI-ként funkcionál. Jelen projektben egy *Siemens Simatic S7-1200*-as PLC-t használtam fel, mivel az egyetem laborjában ilyen típusú eszközökhöz volt hozzáférésem. „A CPU egy mikroprocesszort, egy integrált tápegységet, valamint bemeneti és kimeneti áramköröket egyesít, beépített PROFINET, nagysebességű mozgásvezérlő I/O és beépített analóg bemenetek egy kompakt méretű házban egy nagy teljesítményű vezérlő létrehozásához. A CPU felügyeli a bemeneteket és módosítja a kimeneteket az Ön felhasználói programjának logikája

szerint, amely a Boole logikát, számolást, időzítést, összetett matematikai műveleteket és kommunikációt tartalmazhat a következő programokkal más intelligens eszközökkel.”

[2] Technikailag bármilyen program futtatna a PLC-n, ami képes az eszköz Q azonosítójú kimeneteit vezérleni M azonosítójú belső memória változókkal. Ahhoz, hogy egy külső eszköz vagy program TCP/IP alapú hálózaton keresztül vezérelni vagy olvasni tudja ezeket a Q és M memória területeket a PLC-nek ki kell szolgáltatni ezeket, amit adatblokkok (DataBlock – rövidítve DB) segítségével lehet megtenni. Ezek a DB-k számozva vannak, így lehet rájuk hivatkozni (pl.: DB1).

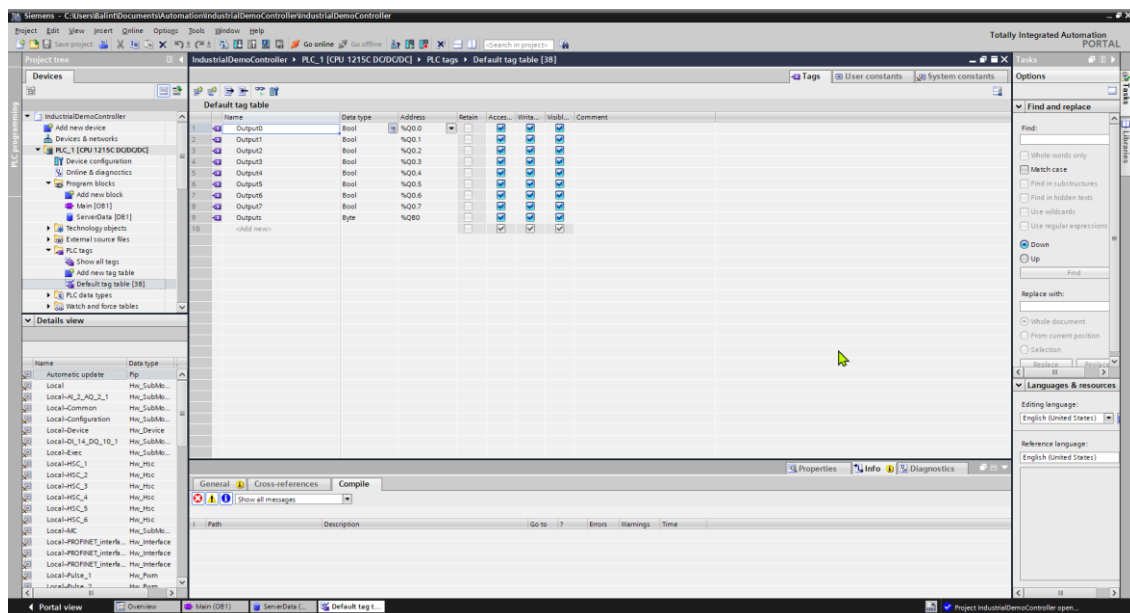


2.2. ábra: S7-1200 PLC, kimenetei

Ebben a projektben egy olyan egyszerű programot írtam a PLC-re, ami különösebb belső logikák nélkül közvetlenül állítja át a Q kimenetet a hozzá kapcsolt M memória regiszter értékére. Minden regiszternek van egy száma, ami 0-tól indul és 1 bájt adatot tárol. Lenne lehetőség 2 és 4 bájt hosszúságú adatok tárolására és módosítására, de jelenleg 1 bájtnyi adatot módosítunk, pontosabban a biteket a bájtban belül.

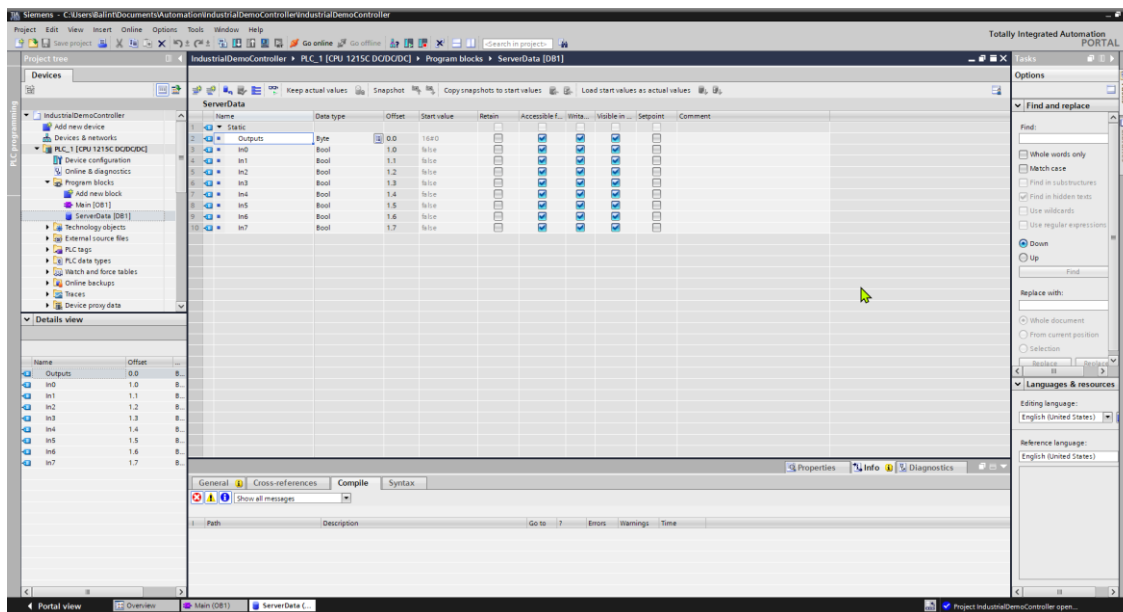
A PLC program elkészítéséhez a Siemens TIA Portal-t használtam, amiben a fizikai eszközre való felöltéshez előbb meg kell adni a PLC típusát, majd a TIA projektben meg kell adni a PLC hálózati elérhetőségét. A Simatic PLC-ken lehetőség van egy TCP/IP-re építkező Ethernet alapú adatcsere konfigurálásra, amihez a PUT/GET kommunikációt kell engedélyezni a PLC beállításainál. Ekkor nem szükséges olyan PLC programrészt írni, ami a TCP/IP kommunikációért felel. Ezzel a beállítással a PLC az adatblokkokat kiszolgáltatja a hálózaton írásra és olvasásra egyaránt, amit majd az adatgyűjtő szerverbe integrált C# könyvtár képes lesz használni.

Ha használni akarunk valamilyen ki vagy bemenetet vagy egyéb adattípust, azt először deklarálni kell, ami jelen esetben azt jelenti, hogy felvesszük a PLC „tag”-ek (címkék) közé, ahol megadjuk a címke nevét, adat típusát és a regiszter címét. Az S7-1200-as PLC-nek 8 kimenete van, ami Q0.0-tól Q0.7-ig tart. A Q jelenti, hogy kimenet regiszterről van szó, 0 azt jelent, hogy ez a PLC-be épített kimenet, vagy is az „nulladik” itt az első jelenti a Q regiszterek között. Ha lenne még kiegészítő kimenet modul a PLC-hez kötve, akkor az lenne a Q1, Q2, stb. A pontutáni szám pedig a regiszter bitjét adja meg, szintén nullától számozva hétig és így lesz meg az egy bájt, ami 8 bit. Fontos még megjegyezni, hogy a bájtrend (endianness) ezen a PLC-n big-endian, vagyis a hetes bit a bájton belül, balról olvasva a biteket a bájt sor végére kerül.



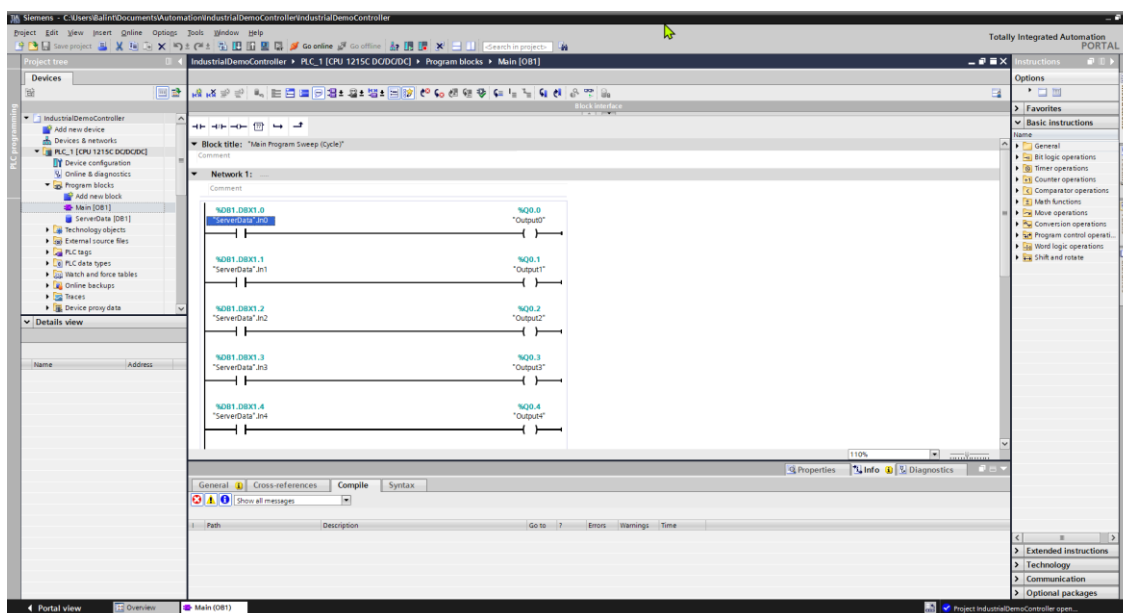
3. ábra: PLC memória címkék

Az adatblokk, ami majd a valós kimenetet és a fizikai bemenet nélküli M regisztereket tárolja majd egymás után vannak felfűzve. Ez a sorrend szabadon módosítható, én most a kimenetet mint egy bájt raktam az elejére és külön-külön bitekét adtam meg - nevezzük egyszerűen bemeneteknek-, az M bit regisztereket. Itt fontos a sorrend, mert mikor majd kiolvassuk az értékeket, vagy éppen módosítani szeretnénk valamelyiket, akkor az itt megadott sorrend és bitek eltolása (offset) a kezdetektől lesz a fontos. A kezdőérték az az adatblokk nulladik bitje, mivel itt az adat tulajdon képen egy nagy összefűzött érték lesz. Azt, hogy mennyi az eltolás mértéke az adat típusa határozza meg, ami bit esetében egy, byte esetében nyolc bit, vagyis egy teljes regiszter léptetés. A kimenet (Outputs) 0.0-val kezdődik, mert egy bájt, míg az In0 1.0-val mert az már a következő bájt 0. bitje.



4. ábra: Adatblokk tartalma

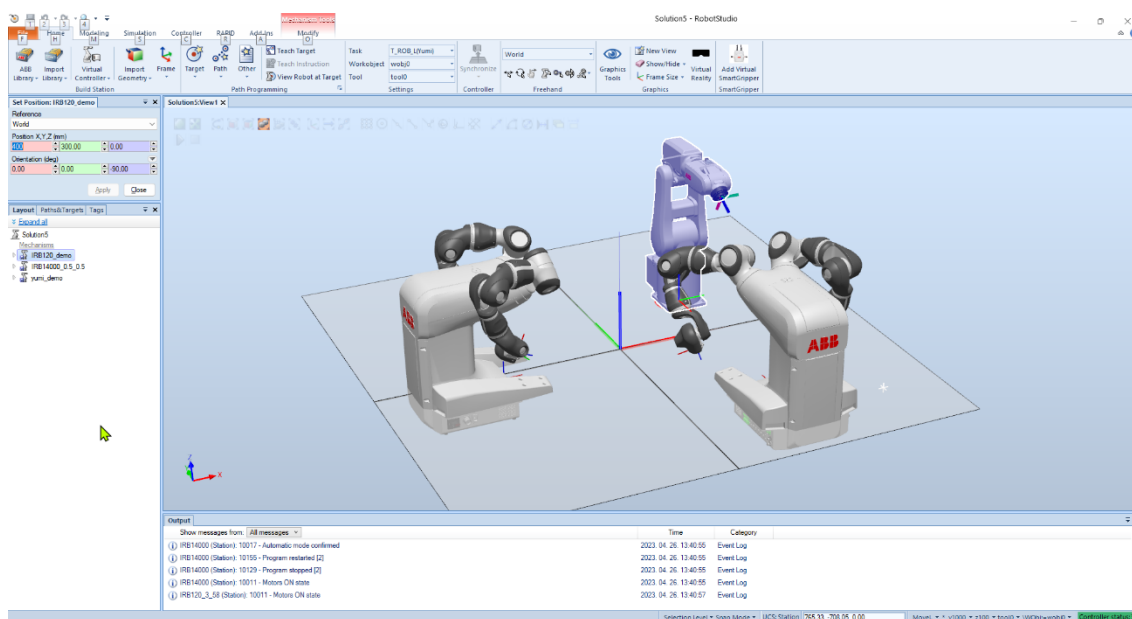
Ennyi PLC alapozás után nézzük a programot, ami a DB In0-7 értékét áttolja közvetlenül a kimenet hozzá társított bitjébe. Ezt a Program blocks-on belül a Main [OB1] program blokk végzi el. PLC-k esetében is érvényes, mint minden más programnyelvénél, hogy kell egy fő belépő ág. Ezt a célt ez a program blokk szolgáltatja. Annyi különbség van például egy konzolos számítógép alkalmazással szemben, hogy a PLC-k végtelenítve futtatják a programot (akár egy mikrokontroller), alap esetben nincs vége a programnak. A PLC-k esetében több fajta programozási módszer is van, én grafikus létra programozást választottam, ami az alapértelmezett módja a TIA Portalnak.



5. ábra: PLC-n futó program

2.3. Virtuális robotok futtatása ABB RobotStudio-ban

Mivel fizikailag, csak egy robot elérhető a laborban ezért, hogy a program működését látványosabbá tegyem virtuális robotokat, is felhasználtam a projektben. Ezek a szoftver szempontjából ugyan úgy látszódnak csak az ABB által fejlesztett RobotStudio programban futnak, de kisebb dolgokat leszámítva, ugyan azt tudják, mint egy valós robot. Ez is az egyik célja magának a RobotStudio-nak, hogy még az előtt fel meg lehet írni és tesztelni egy gyártás robot programját (vagy akár más szimulált gépeket is) mielőtt azt a valós robotra feltöltenénk. Illetve másik haszna, hogy ezzel lehet a robotok firmware-jét frissíteni, ami technikailag az operációs rendszer, ami a robot vezérlőn fut, amit az ABB IRC5-nek nevez, illetve már felprogramozott robotokhoz is lehet vele kapcsolódni, azokról letölteni a programot és kényelmes módon számítógépről módosítani. A virtuális robotok mind localhost-on érhetőek el. A valós és virtuális vezérlők is az alapértelmezett felhasználónévvel és jelszóval futnak, ami Default User és robotics.



2.6. ábra: Robotok a RobotStudio-ban

2.3.1. Robotok felépítése és programozása

A projektem megértéshez nem kell elmélyülni a robotika világában, de azért nem árt néhány alapfogalmat áttekinteni, a teljesebb megértéshez. Kulcsár Béla Robottechnika könyve így fogalmazza meg a robotot: „Az ipari robotok definíciójából következik a mechanikus beavatkozás nélküli átprogramozhatóság. Az átprogramozhatóság

(programmódosítás vagy programbefolyásolás) többféle módon végrehajtható.” [3] A robotok mechanikus egységekből, azokat hajtó szerkezetekből (manapság már nagyrészt szervo- vagy léptetőmotorokból) állnak, illetve az energiával ellátó vezérlőkből állnak. Számunkra most kifejezetten a vezérlés a fontos, hiszen a programom ehhez kapcsolódik és ebből nyeri ki a szükséges információkat.

Az ABB robotkarok vezérlőjét IRC5-nek nevezik és rajta futó operációs rendszert, ami többek között az ABB cég RAPID programnyelvén futtatni tudja a programokat RobotWare OS-nek. „A RobotWare az ABB Robotics szoftvertermékcsaládja. A termékeket úgy tervezték, hogy növeljék a termelékenységet, és csökkentsék a robotok birtoklásának és üzemeltetésének költségeit. Az ABB Robotics sok évet fektetett a termékek fejlesztésébe, és ezek a termékek több ezer robot telepítésén alapuló tudást és tapasztalatot képviselnek.” [4]

Ebben a projektben nagyon egyszerű program fut a robotokon, általában csak pontok közötti mozgások vannak felprogramozva. A programot betanítással vettem fel a labor YuMi robotján és az van elindítva rajta, míg a virtuális robotokon üres feladatok futnak.

2.3.1.1. Ipari Labor YuMi robotja

Demonstrációs céllal, hogy ne csak szoftveresen létrehozott virtuális vezérlőkhöz, illetve azokon keresztül robotokhoz kapcsolódjunk, a projektben felhasználtam a Pannon Egyetem nagykanizsai kampuszán található Alkalmazott Informatika Tanszékhez tartozó Ipari laborban található ABB márkájú IRB 14000-es robotot, amit a cég is leginkább YuMi-ként emleget, aminek jelentése „You and Me”, ami arra utal, hogy ez a kollaboratív robot képes emberekkel együtt dolgozni védőcellák használata nélkül. Az alábbi [oldalon](#) lehet hozzáférni a robot technikai információihoz:

About YuMi®



Human - robot collaboration

Innovative human - friendly dual arm robot with breakthrough functionality designed to unlock vast global additional automation potential in industry. YuMi® is designed for a new era of automation, for example in small parts assembly, where people and robots work side-by-side on the same tasks. Safety is built into the functionality of the robot itself. YuMi® removes the barriers to collaboration by making fencing and cages a thing of the past.

ABB's growing family of YuMi robots is part of a suite of exciting collaborative automation solutions that help people and robots safely work closer together than ever before possible.

ABB is adding a new member to the YuMi family, 7 axis YuMi, the smallest and most agile collaborative robot yet, making it easier than ever to put into production. The new 7 axis also opens up a world of flexible possibilities – for example a single-arm and a dual-arm YuMi could be combined to add a parts feed or inspection station to an assembly cell. Additionally, customer might want to have three or five YuMi arms in an application to improve the cycle time.

The entire YuMi family is designed to be easy to setup and use thanks to its intuitive lead through programming. That means that even people without specialized training or prior experience can successfully use robots. Read more about [Single-arm YuMi](#).

How we help our customers



ABB robots help accelerate COVID-19 vaccine development in Thailand



ABB robots aid rapid automated testing for COVID-19 virus



YuMi takes over THT assembly

2.7 ábra: ABB IRB 14000 weboldala

„Az IRB 14000 robot egy kétkarú ipari robot integrált vezérlővel. Mindkét kar hét tengellyel rendelkezik, ami egy plusz szabadságfokot biztosít egy hagyományos 6 tengelyes robotokhoz képest.” [5] A robot fel van szerelve megfogókkal, amiket 3D nyomtatással készítettem el az ipari labor nyomtatóján. Ezeknek az univerzális prizmatikus megfogó ujjaknak a 3D modellje a gyártó honlapjáról tölthető le.

A labor YuMi robotját a 10.61.10.202-es IP címen érhető el és amint az alábbi képen látható egy asztalra van téve, előtte egy szállító szalag. Mivel ennek a fizikai pozícióját nem volt módom kimérni ezért a mobil applikációban egy általam választott helyre kerül a virtuális gépek közé.



2.8 ábra: Ipari labor ABB IRB14000 robotja

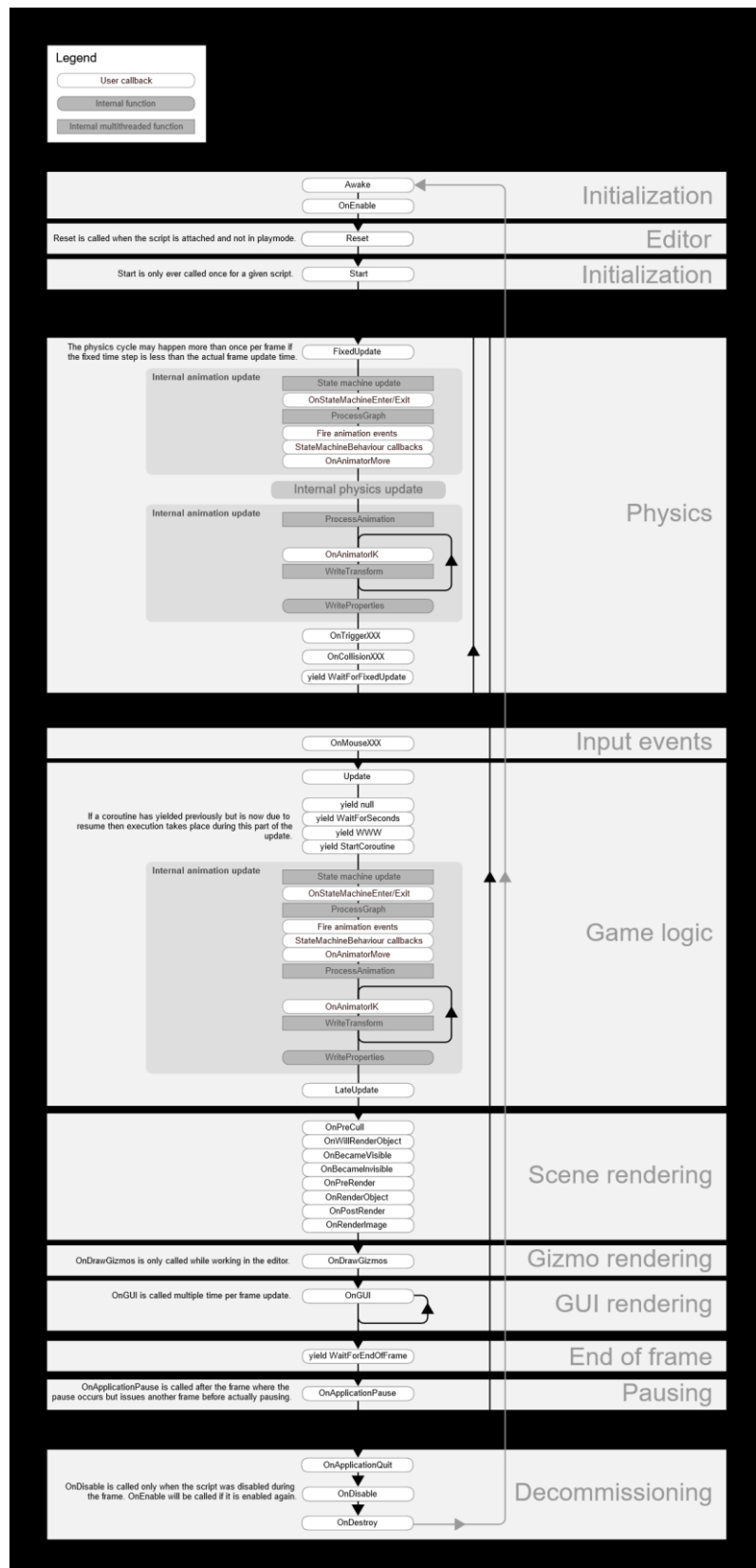
2.4. Unity játékfejlesztő motor és ARCore

Az alkalmazás, ami a fő produktuma a diplomadolgozatomnak, Unity nevű főként játékok, de a hivatalos oldala szerint is bemutatók, mérnöki és építészeti 3D látványtervek fejlesztésére lehet használni. Ez egy komplett fejlesztő eszköz vagy ahogy a játékfejlesztő szakmában mondják játék motor, ami a 2D és 3D-s munkáktól, az animáláson át a szkriptelhető programozásig, valamint a fizikai és 3D renderelésig mindent egy programon belül megold. Funkciói úgynevezett csomagokkal (package) bővíthető.

Ezekre a csomagokra nagy szükségünk is lesz, hiszen így tudjuk majd használni a Google által fejlesztett ARCore SDK-t vagy éppen a ShaderGraph nevű eszközt, ami megkímél minket a látványos grafika programozásától.

A program fő eleme a GameObject (röviden GO) vagy magyarul játék objektum. Ezeken végezhetünk mozgásokat és ehhez lehet komponenseket és C# szkripteket illeszteni. Az előbb említett programnyelv nem azt jelenti, hogy egy Unity-val készült program .NET alapú. Ezt csak belső szkript nyelvnek használja és ebből következik, hogy

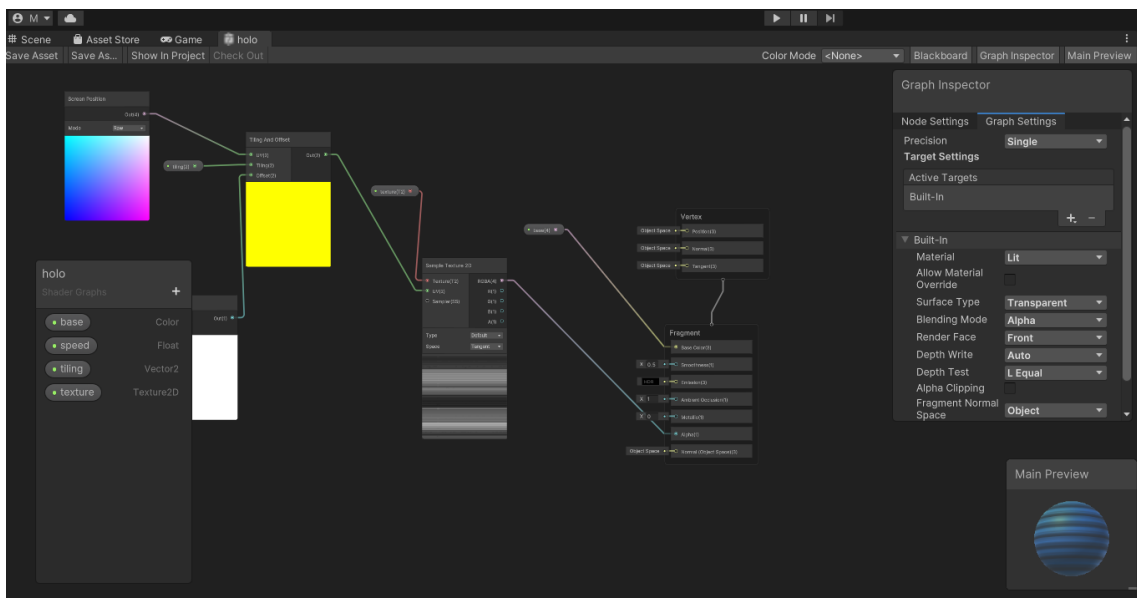
kicsit másként is kell értelmezni az így létrejött szkriptek futását, hiszen nincs különösebb ráhatásunk a lefutások sorrendjét illetően. Illetve már egy szkript is végtelenítve fut, ha van benne Update() metódus. Egy Unity szkript életciklusát a következő kép mutatja:



2.9. ábra: Szkript életciklusa (forrás: <https://docs.unity3d.com/Manual/ExecutionOrder.html>)

Mivel a szkriptek folyamatosan futnak ezért állapotgépet kell használni. „Az állapotgépek nagyszerűek, ha olyan rendszerünk van, ahol abszolút garanciát kell nyújtani arra, hogy egyszerre csak egy állapot létezhet. Ha az állapotok rögzített halmaza van - például egy karakter, amely mozogni, ugrani vagy kúszni tud -, soha nem akarjuk, hogy a karakter egyszerre ugorjon és kússzon.” [6]

Az alkalmazás nem sok grafikus elemet tartalmaz, de érdekességgént és mert jól mutatja a valóságos és a virtuális robot közötti különbséget készítettem egy holografikus shadert is. Ehhez a Unity ShaderGraph nevű kiegészítője nagy segítséget nyújtott, hiszen így nem kellett tényleges programkódban megírnom, hanem intuitív, összehúzogatható módon tudtam lekészíteni, amit az alábbi ábra mutat:



2.10. ábra: ShaderGraph működés közben

Mi is a shader? „A számítógépes grafikában a shader egy olyan program, amely lehetővé teszi megjelenítési effektusok számítását a grafikus kártyán. A shaderek-et a GPU futtatja és a grafikus csővezeték egyes részei programozhatók segítségével.” [7]

2.5. Visual Studio és .NET keretrendszer C# nyelvvel

A szerverek, amik majd az alkalmazást kiszolgálják Visual Studio 2022 Community Edition-ben (röviden VS) lettek programozva, .NET keretrendszer használatával C# (ejtsd: szí sharp) programozási nyelven.

A webszerverhez és az adatszerverhez is kellett kiegészítő függvénykönyvtárakat telepítenem, amiket a NuGet csomagkezelővel csináltam VS-en belül. Az Entity Framework (röviden EF) alapvető Microsoft-os csomagja mellett, még le kellett töltenem a Pomelo MySQL csomagját. „A Pomelo.EntityFrameworkCore.MySql a legnépszerűbb Entity Framework Core szolgáltató MySQL kompatibilis adatbázisokhoz. Támogatja az EF Core-t a legújabb verzióig, és a MySqlConnection-t használja a nagy teljesítményű adatbázis-kiszolgáló kommunikációhoz.” [8] - írja magáról a Github oldalán a Pomelo fejlesztője.

A robotok eléréséhez telepíteni kell az ABB PC SDK-t és onnan kell beimportálni a szükséges DLL-t. Az API dokumentumában sok már elavult megoldás van, de azért sikerült működésre bírnom. Megjegyzem, hogy ez az fejlesztőeszköz sima .NET Framework-höz van és Core estén még be kell importálni néhány Microsoft-os DLL, hogy működjön. Azt, hogy mik is ezek a forráskódban megtalálhatóak komment formájában. Az SDK innen érhető el: <https://developercenter.robotstudio.com/pc-sdk>

A Siemens PLC-hez a Snap7 könyvtárat kell telepíteni, aminek a C#-os implementációja a Sharp7. Ezt kifejezetten az S300, S1200 és S1500-as szériájú PLC-khez fejlesztették ki.

Az adatcserét a szerver és az alkalmazás között pedig a Riptide Networking könyvár oldja meg, amit fejlesztője kifejezetten Unity-s többszereplős játékokhoz talált ki, mivel a hálózati kapcsolat megvalósítása nem egyszerű feladat Unity-ben (ez a dolgot megkezdése elején még igaz volt, azóta sokat fejlődött a natív Unity hálózat kezelés). Ennek az eszköznek a letöltése a következő linken lehetséges: <https://github.com/RiptideNetworking/Riptide> .

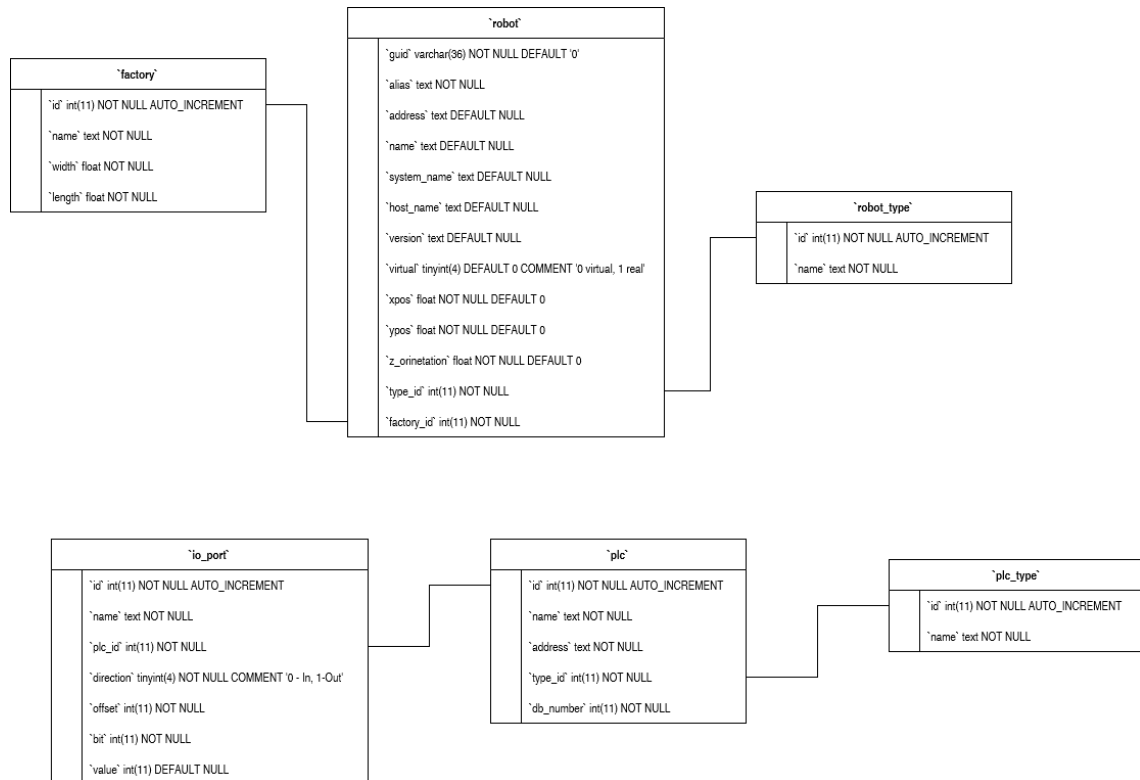
Végül pedig a QR kód generáláshoz szintén egy nagyon jó ingyenes csomagot használtam fel amit, a <https://github.com/codebude/QRCoder> oldalról lehet letölteni.

2.5.1. Adatbázis

Ahhoz, hogy a mobil alkalmazás rugalmasan tudjon működni, a programkódjának különösebb újraindítása nélkül ezért amennyi információt le lehet kérni a PLC-kről vagy a robotokról, illetve olyan adatokat tárolni, amelyek a program loggiákájához kellenek azt célszerű adatbázisban tárolni. Az utóbbira példa a gyárterület mérete és kitalált megnevezése. Ezt az információt mi adhatjuk meg, nem a gépekből származik

közvetlenül és lehetőség van több felvitelére is, bár a jelen projektben egyelőre egy gyárterületet használunk.

Az adatbázist SQL alapú MariaDB-ben készítettem, ami a MySQL-nek a fejlesztői által létrehozott nyílt forráskódú adatbázis rendszer, amelynek szintakszisa szinte teljesen megegyezik zárt forrású „rokonával”.



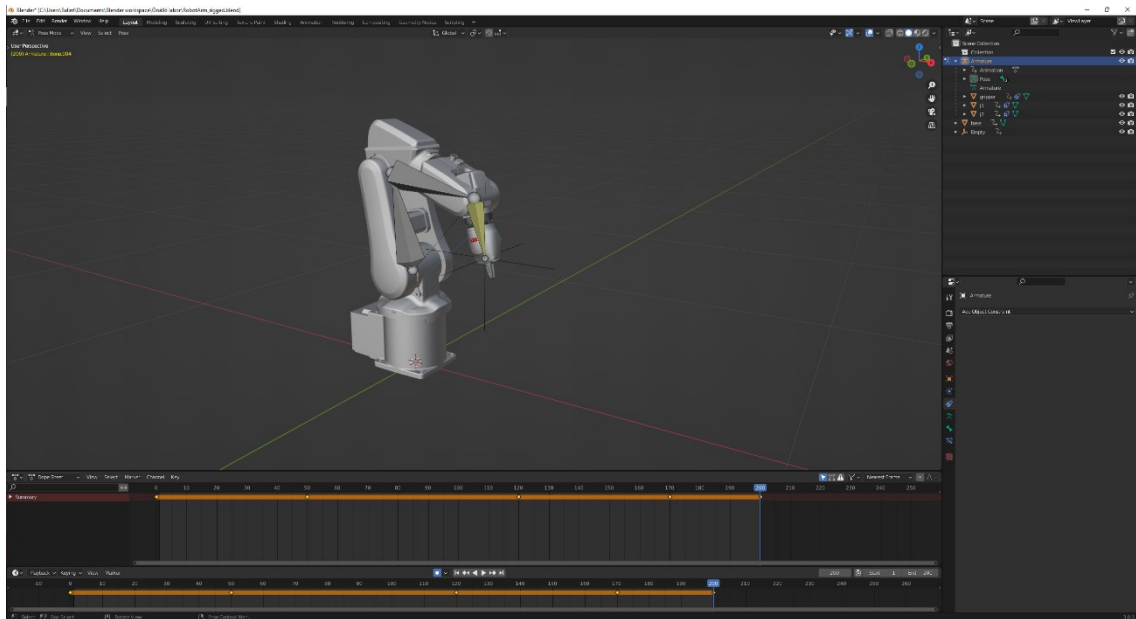
2.11. ábra: Adatbázis táblák kapcsolata

Maga az adatbázis a XAMPP nevű programnak a MySQL szerverén fut és az általam kedvelt Heidi SQL-ben állítottam össze. Alapvetően nem az adatbázis tervezés a szakterületem, de rengeteg módosítás után a célnak megfelelő tudtam létrehozni, ami jól kiszolgálja a projektet.

2.6. Kiegészítő szoftverek: Blender és Photoshop

Ezek a szoftverek csak a céljaim támogatását segítették, közvetlenül nem érintik a projektfeladatomat. Blender-t az ABB RobotStudio-ból kimentett virtuális robot modellek Unity számára is jó fájlformátumra konvertáláshoz használtam. Ez mellett még ahol szükséges volt, kisebb módosításokat hajtottam végre a modelleken, illetve az inverz kinematikához egy teszt erejéig megcsináltam a mozgatható vázat az egyik robotkaron (IRB120). Ez a funkció nem lett felhasználva a jelen projektben, de a modell, ami

belekerült az alkalmazásba tartalmazz a „rigged” vázat. A Photoshop-ot pedig pár textúra átszínezéséhez használtam fel.



2.12. ábra: Robotkar riggel-ve

3. Szerverek

Ahhoz, hogy az alkalmazás megfelelően tudjon működni különböző szerverek kellenek a háttérbe. Ebben a projektben kettő szervert fejlesztettem ezek közül a fontosabb az adatszerver, ami API-ként működik és elérhetőséget nyújt az alkalmazás számára a különböző ipari eszközökhöz. A másik pedig egy webszerver, ami az eszközök konfigurálást és adatbázisba való felvitelét, valamint módosítását segíti. Következő fejezetekben ezeket mutatom be részletesen.

3.1. Adatbázishoz való kapcsolódás

Az adatbázisról már esett szó a [2.5.1](#)-es fejezetben, itt most azt mutatom be, hogyan a projektben hogyan valósítottam meg a szerverekkel való kapcsolatát. Mind a webszerver és adatszerver esetében EF-et használtam az adatbázis eléréséhez. „Az EF Core olyan O/RM-ként használható, amely lekérzést végez a relációs adatbázis és a .NET világába található osztályokat és szoftver kódok között.” [9] Az Entity Framework nagyon kényelmes használatot tesz lehetővé adatbázis kezeléshez, hiszen nem szükséges szöveges módon megírnia az SQL lekérdezéseket, hanem minden tábla a neki megfelelő objektumba kerül, megmaradnak a táblák közötti kapcsolatok és minden adatot LINQ-hoz hasonló módon tudunk elérni. Azért mondom, hogy hasonló, mivel nem lenne szükség a LINQ könyvtárak behúzására a projektbe mert a EF tartalmazza alap szinten, de ha szükség lenne rá, akkor használható az is és akkor többlet képességekhez jutunk. Erre a összekapcsolt táblák kezelésekor van szükség.

3.2. Adatokat gyűjtő és kiszolgáló szerver

A mobil alkalmazás mellett ez a másik fontos szoftveres komponens, ami a fejlesztés alatt elkészült. Ez tartja a kapcsolatot a PLC-kel és a robot vezérlőkkel, valamint az ezekből származó adatokat beleírja az adatbázisba, ebből pedig kiszolgáltatja a mobil alkalmazásnak. Illetve akár többnek is párhuzamosan, mivel úgy van megírva, hogy amíg

egy mobil eszköz is érzékel, hogy kapcsolódva van rá, addig szórással mindegyiknek kiküldi az aktuális adatokat.

A szerver a mobillal a korábban már említett *Riptide networking* könyvtár segítségével kommunikál. A szerver egy üzenet azonosítót figyel, ami az UPDATE_PLC. Amikor ez az üzenet érkezik be, akkor az üzenetben kapott PLC adatbázisban lévő azonosítója és a bemenet azonosítója alapján, a kapott értéket átírja az adott bemenethez az adatbázisban. Ez az érték fog aztán majd átkerülni a PLC-re, ami minden egyes frissítéskor megtörténik, de ezt a következő fejezet tárgyalja.

```
[MessageHandler((ushort)MessageId.POST_PLC_DATA)]
private static void PostPlcData(ushort fromClientId, Message message)
{
    int plcId = message.GetInt();
    int ioId = message.GetInt();
    bool ioValue = message.GetBool();
    ModifyIOValue(plcId, ioId, ioValue);
}
```

A fenti programkódban látható, hogy egy C# attribútum kezeli az üzenetek számához tartozó metódusok lefutását. Minden metódus előtt egy vagy akár több ilyen attribútum is megadható - bár ennek itt most nem sok értelme lenne – és ezek a fordítóprogramnak metaadatként funkcionálnak. Ezek az attribútumok egy üzenet azonosítót várnak és ezzel fogják tudni, melyik metódus fusson le. Az üzenet tartalma pedig sorosítva van, vagyis amit egyszer kivettünk belőle az eltűnik és jön a következő. Erre érdemes figyelni és kiolvasott adatot változóba tárolni, ha többször is szeretnénk felhasználni. Fontos az is, hogy a soron következő adat típusát jól határozzuk meg, mert nagy felfordulás lehet, belőle, ha például *integer* helyett *float*-ot akarunk kiolvasni.

A tényleges módosítás a *ModifyIOValue* metódusban történik meg, amiből az érdemi rész a következő:

```
var io = plc.IoPorts.Where(i => i.Id == ioId).FirstOrDefault();
dbBuffer.SetBitAt(io.Offset, io.Bit, value);
result = s7client.DBWrite(plc.DbNumber, 0, bufferSize, dbBuffer);
```

Itt az látható, hogy egy LINQ lekéréssel a lekérjük azt az IO-t amelyiknek az adatbázis azonosítóját megkapuk a mobil alkalmazásból, majd ezt kiírjuk a PLC-re.

3.2.1. Adatgyűjtés PLC-ről

A PLC-ről csak azt olvashatjuk ki, amit az eszköz kiszolgáltat magáról és ez már korábban tárgyalva volt. Most nézzük meg, mit is kell tenni, ha tudjuk mennyi és milyen hosszú adatok, milyen sorrendben követik egymást és ezeket szeretnénk a megfelelő adatblokkból kiolvasni. Ahhoz, hogy a program rugalmasan tudja ezt kezelni ezeket előre

megadjuk az adatbázis PLC és a hozzá kapcsolt IOport tábláiban. Az I/O-k esetében fontos, hogy ki vagy bemenetről beszélünk, tehát az irány, valamint az eltolás mértéke, ami az elküldött adat elejétől értendő bájt sorszáma az adatsorban. Mivel bájtokról beszélünk ezért mindegyik érték a következőhöz képest eggyel lesz eltolva és azon belül pedig a bit mező adja meg, hogy melyik értékét állítjuk vagy kérjük le az adatbázisban. Az érték oszlopba pedig mindig az aktuálisan kiolvasott érték kerül. A robotok és PLC-k is egy közös Update metódusból futnak le, amikor is frissül az adatbázis a PLC-ről kiolvasott értékekkel, majd, hogy az alkalmazást és a hálózatot se terheljük feleslegesen jön egy kis kényszerített várakozás, ami jelenleg kettő másodperc. Ezt a következő kódrészlet mutatja be:

```
...
stringjsonPLCs=UpdatePlc();
...
riptideServer?.SendToAll(Message.Create(MessageSendMode.Reliable,MessageId.UPDATE_PLC).
AddBytes(Compress(jsonPLCs)));
Thread.Sleep(2000);
...
```

Innen jó látható, hogy az *UpdatePlc* metódus hívódik meg ami majd egy JSON-ba szerializált objektumok sztringjével tér vissza, amiben a PLC adatai találhatóak összeállítva. Ezt a sztringet küldi majd szét a rácsatlakozott alkalmazásoknak, majd várakozik egy kicsit és kezdi előről.

A tényleges adatlekérés és adatbázisba rakás, tehát az *UpdatePlc*-ben található, aminek a lényegi része így néz ki:

```
byte[] dbBuffer = new byte[bufferSize];
int result = s7client.DBRead(plc.DbNumber, 0, bufferSize, dbBuffer);
...
foreach (var ioPort in plc.IoPorts)
{
    ioPort.Value = Convert.ToInt32(S7.GetBitAt(dbBuffer, ioPort.Offset, ioPort.Bit));
    _context.IoPorts.Update(ioPort);
    _context.SaveChanges();
}
```

Itt az látható, hogy az *s7client*, ami már egy előre megnyitott kapcsolatot tartalmaz a PLC felé, annak a *DBRead* metódusával megadjuk, hogy mennyi adatot és melyik sorszámú adatblokkból szeretnénk beolvasni. Ezek az adatok az adatbázisból származnak. Majd annak a PLC-nek, amit az adatbázis szerint megnyitottunk, lekérjük a hozzá tartozó I/O portokat és az eltolás és bithely szerint kiolvassuk az értékeket, amik mennek az adatbázisba.

Ez után már a PLC-re nincs szükség és bontjuk a kapcsolatot és létrehozunk egy olyan adatstruktúrát, amiben minden szükséges adat benne van a kiválasztott PLC-ről. Bár a mobil alkalmazást egy későbbi fejezet tárgyalja, de annyit megemlítenék, hogy egy fizikailag létező gépet beazonosítunk és annak az azonosítójával kérjük majd ebben a

metódusban a soron következőleg bemutatott adatokat. Programozásban, amikor egy strukturált adatókból(objektumkból) állítunk elő egy másikat, azt „*mapping*”-nak vagy magyarul leképezésnek nevezi a szaknyelv. A következő programrészlet azt mutatja meg, hogy az adatbázisból lekért és C# objektumként, illetve azok listájaként tárolt PLC-ken egy LINQ kifejezés (*expression*) alapú lekérdezést hajtunk végre és a lekérdezés eredményét egy új általunk meghatározott felépítésű objektumba illesztjük bele. Itt még érdekes, hogy az IO-k listáját még külön egy ugyan ilyen módon leképzett lekérés tölti fel adatokkal, amit egymásba ágyazott leképezésnek nevezünk (*nested mapping*).

```
var mappedPlcs = from plc in plcs
                  select new
                  {
                      plc.Id,
                      plc.Address,
                      Type = plc.Type.Name,
                      plc.DbNumber,
                      plc.Name,
                      IOs = from io in plc.IoPorts
                           select new
                           {
                               io.Id,
                               io.Name,
                               io.Offset,
                               io.Bit,
                               io.Direction,
                               io.Value
                           }
                      };
jsonResult = JsonConvert.SerializeObject(mappedPlcs, new JsonSerializerSettings {
    Formatting = Formatting.Indented, ReferenceLoopHandling = ReferenceLoopHandling.Ignore
});
```

Miután összeállt az adat, ebből az objektumból JSON sztringet szerializálunk, ahol azt fontos megemlíteni, hogy ebben ciklikus egymásra utalások is lehetnek, az EF sajátosságaiból fakadóan, vagyis ahogy létrehozza a táblák közötti kereszthivatkozásokat, ezért tudatni kell a JsonConvert-rel, hogy ezeket a kereszthívásokat hagyja figyelmen kívül, amit a ReferenceLoopHandling beállítás ignorálásával érünk el.

3.2.2. Adatgyűjtés robotról

A robotokról való adatlekérés és tárolás nagyon hasonlóan működik a PLC-khez, de van egy fontos eltérés is. Bár lehetett volna az is, hogy minden robot vezérlőnek az IP címét tároljuk és aztán közvetlen azt lekérve az adatbázisból csatlakozunk rá, de ez sokkal nehezebb megoldás és talán kevésbé rugalmas, mint ami mellett döntöttem, vagyis, hogy hálózati felderítést végzünk és az elérhető robot vezérlőket lekérjük, majd azokon egyesével végig lépkedve begyűjtjük az adatokat. Illetve csak azokat, melyek GUID-ja vagyis egyedi azonosítója révén már fel voltak véve az adatbázisba. Ezt a módszert javasolja az ABB PC SDK API referenciás oldala is példafájlok segítségével. Ezek egy

része már elavult volt, szóval változtatnom kellett rajta, amiről sajnos semmit nem említettek a leírások, valamint .NET Core esetén még telepíteni kell, néhány extra könyvtárat, de végül sikerült működésre bírni.

A PLC-nél említett Update metódusban van egy UpdateRobot metódus hívás, ami a tényleges munkát végzi. Ennek a rövid és részleges programkódja látható alább, ahol is először aktiváljuk a hálózat szkennert és a vezérlők információt tároló üres tömböt.

```
NetworkScanner scanner = new NetworkScanner();
ControllerInfo[] controllerInfos;
```

Ez után külön lekérjük a virtuális és valós robotokat(részleges kód látható itt, a feltételeket és a ciklust nem másoltam ide):

```
controllerInfos = scanner.GetControllers(NetworkScannerSearchCriteria.Real);
//VAGY
controllerInfos = scanner.GetControllers(NetworkScannerSearchCriteria.Virtual);
```

A megtalált vezérlőkön végiglépked majd ezeken belül az adatbázisba felvitt robotokon és ha egyezést talál akkor lekéri az információkat.

```
if (controllerInfos.Length > 0)
{
    foreach (var ctrlInfo in controllerInfos)
    {
        foreach (var r in robots)
        {
            if (ctrlInfo.SystemId.Equals(Guid.Parse(r.Guid)))
            {
                r.Address = ctrlInfo.IPAddress.ToString();
                r.Name = ctrlInfo.Name;
                r.SystemName = ctrlInfo.SystemName;
                r.HostName = ctrlInfo.HostName;
                r.Version = ctrlInfo.Version.ToString();
                r.Virtual = (byte)i;
                Controller controller = Controller.Connect(ctrlInfo, ConnectionType.Standalone);
```

Itt pedig a robot vezérlőn futó aktuális feladatot (programot) és abból is az aktuális robot szerszám koordinátákat:

```
RobTask[] tasks = controller.Rapid.GetTasks();
r.RobotTasks.Clear();
foreach (RobTask t in tasks)
{
    r.RobotTasks.Add(
        new Robot.RobotTask
        {
            Name = t.Name,
            Position =
                new Robot.RobotTask.Vector3
                {
                    X = t.GetRobTarget().Trans.X,
                    Y = t.GetRobTarget().Trans.Y,
                    Z = t.GetRobTarget().Trans.Z
                }
        });
}
_context!.Update(r);
_context!.SaveChanges();
}
}
```

A következő adat leképzés rész, már részletesen volt tárgyalva a PLC-nél, ez is nagyon hason hozzá. Érdekes, hogy ha nem adunk meg külön nevet, akkor az eredeti objektum nevét kapja meg az új objektum tulajdonsága.

```
var mappedRobots = from robot in robots
                    select new
                    {
                        robot.Guid,
                        robot.Address,
                        robot.Alias,
                        robot.Name,
                        robot.SystemName,
                        robot.HostName,
                        robot.Version,
                        robot.Xpos,
                        robot.Ypos,
                        robot.ZOrinetation,
                        robot.Virtual,
                        FactoryName = robot.Factory.Name,
                        FactoryWidth = robot.Factory.Width,
                        FactoryLength = robot.Factory.Length,
                        Type = robot.Type.Name,
                        robot.RobotTasks
                    };
```

3.3. WEB-es felület adatbázis menedzseléshez

Bár nem lenne nehéz egy erre specializált szoftver segítségével (pl.: HeidiSQL) az adatbázisba felvinni az adatokat, felhasználó ergonómia szempontjából még is csak elegánsabb, ha erre egy dedikáltan erre a célra fejlesztett felületből van lehetőség. Ez lehetett volna akár egy ablakkezelős program is, de platform függetlenség és a webes technológiák előretörése miatt inkább én is egy ilyen mellett döntöttem.

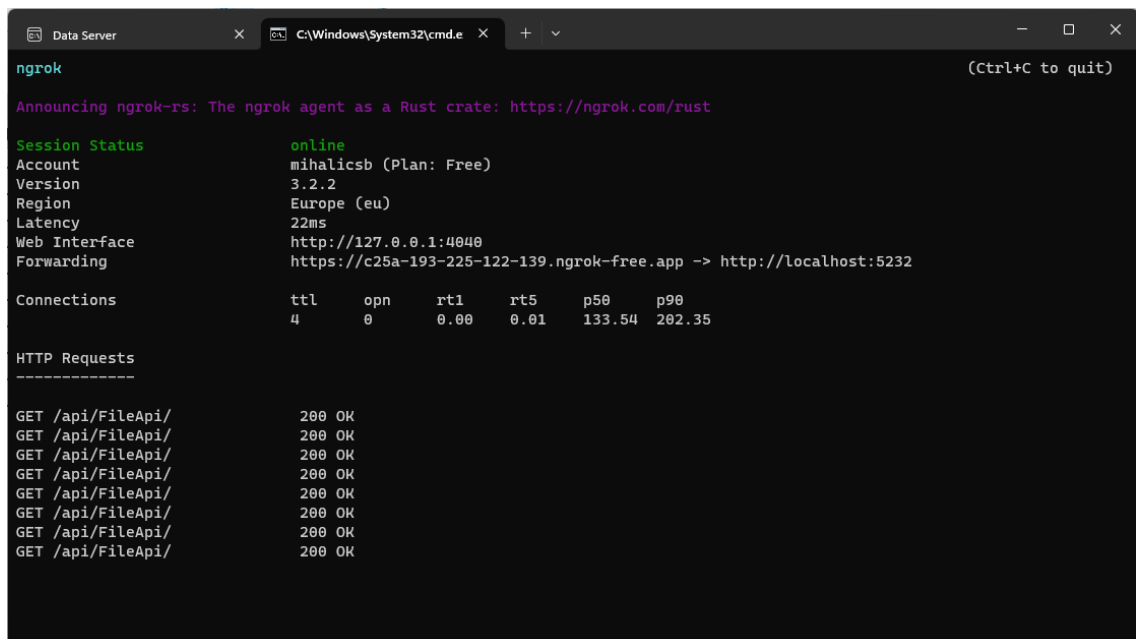
Mivel már volt tapasztalatom benne, ezért a Microsoft ASP.NET Core webes keretrendszerét választottam, hiszen ezt is az általam jól ismert C# programnyelvvvel használhattam. „Az ASP.NET Core a Microsoft népszerű ASP.NET webes keretrendszerének legújabb fejlesztése, amelyet a Microsoft 2016 júniusában jelent meg. Az ASP.NET korábbi verziói számos inkrementális frissítésen estek át, amelyek a következőkre összpontosítottak a magas fejlesztői termelékenységre és a visszafelé kompatibilitás előtérbe helyezésére. Az ASP.NET Core ezzel szemben jelentős architektúrális változtatásokkal állt elő, amelyek újra gondolják a webes keretrendszer tervezését és felépítését.” [10]

3.3.1. Telepítés és beállítás

Visual Studio-ban egy új ASP.NET Core Web App projektet kell indítani, majd érdemes a nagyobb támogatottságú .NET verziót választani és nem a legújabbat, ami az én

esetemben a .NET 6.0-át jelentette. Jelenleg nem használunk hitelesítést, sem HTTPS-t, ami termék esetén nagy hiba lenne, de ebben a projektben nem jelent különösebb problémát. Ha elnevezzük a projektet akkor *Create* gombbal elkészült a webes projektünk, amit még be kell állítani, hogy használni tudja az Entity Framework függvény könyvtárát (*library*, .dll kiterjesztéssel) valamint elérhető legyen a *Scaffolding* művelet.

Mivel ez csak egy bemutató program és nem került implementálásra egy éles környezetben, amihez Windows tűzfalat és egyéb hálózati beállításokat kellett volna módosítani, illetve mivel az egyetemen nem férhetek hozzá a hálózati beállításokhoz, célszerű volt egy olyan programot használni, ami kivezeti a webszerver az internetre bármilyen hálózatban is legyen. Erre az ngrok nevű alkalmazást használtam, aminél megadhattam a webszerver portját és, hogy http protokollt szeretném kiengedni localhost-ról ez program nyitott egy alagutat egy véletlenszerűen generált internetcímre.



```
ngrok (Ctrl+C to quit)
Announcing ngrok-rs: The ngrok agent as a Rust crate: https://ngrok.com/rust

Session Status      online
Account             mihalicsb (Plan: Free)
Version             3.2.2
Region              Europe (eu)
Latency              22ms
Web Interface        http://127.0.0.1:4040
Forwarding           https://c25a-193-225-122-139.ngrok-free.app -> http://localhost:5232

Connections
  ttl   opn   rt1   rt5   p50   p90
    4    0    0.00  0.01  133.54 202.35

HTTP Requests
-----
GET /api/FileApi/      200 OK
GET /api/FileApi/      200 OK
GET /api/FileApi/      200 OK
GET /api/FileApi/      200 OK
GET /api/FileApi/      200 OK
GET /api/FileApi/      200 OK
GET /api/FileApi/      200 OK
GET /api/FileApi/      200 OK
```

3.1. ábra: ngrok webszerver alagút

A fenti képen az látható, hogy a *Forwarding* URL-lel lehet elérni a szervert és GET lekérésekkel a QR kódot szolgáltató API-val letöltötte az alkalmazás a képeket.

3.3.2. Gyárterület menüpont

A webfelület ezen menüpontja lehetőséget ad új gyárterület, robotok és robot típusok felvitelére, valamint itt listázza ki kezdésként a robotokat, amik már fel lettek véve az adatbázisba.

Adat kezelő szerver

Robotok

PLC-k

Új hozzáadása

Gyárak

Robot típusok

Álnév	Direkt keresési cím	Robot neve	Rendszernév	HOST név	Verzió	Virtuális	Robot térbeli X pozíciója	Robot térbeli Y pozíciója	Robot térbeli Z irányultsága	Gyár neve	Robot típusa		
Virtuális IRB120 1	127.0.0.1	IRB120_3_58	IRB120_3_58		6.13.0.1		0.5	0.3	-90	Gyár_1	IRB120	Módosít	Töröl
Labor Yumi	10.61.10.202	14000-501288	14000-501288		6.6.0.1		0	0.3	-90	Gyár_1	IRB14000	Módosít	Töröl
Virtuális Yumi 2	127.0.0.1	IRB14000	IRB14000		6.13.0.1		-0.5	-0.035	0	Gyár_1	IRB14000	Módosít	Töröl
Virtuális Yumi 1	127.0.0.1	Yumi	Yumi		6.13.0.1		0.4	-0.5	90	Gyár_1	IRB14000	Módosít	Töröl

Diplomamunka 2023 - Mihálics Bálint

Diplomamunka 2023 - Mihálics Bálint

3.2. ábra: Robotok listája

Innen lehet új robotot felvinni vagy átlépni a gyárak és robottípusok listázását végző oldalalakra. Új robot esetén azok a beviteli adatok jelennek meg, amiket nem az adat szerver fog majd kitölteni, hanem a felhasználónak kell megadnia. Ebből a legfontosabb a GUID, ami egyben a rekord egyedi kulcsa is. Ezt az egyedi elsődleges kulcsot általában automatikusan növekményes formában szokták kezelni az adatbázisokban, itt viszont mivel már eleve rendelkezünk eggyel a robotból ezt használjuk, hiszen ez fogja majd az adatszerverben a hálózati keresés eredményeképpen lekért robotot ehhez az adatbázis bejegyzéshez kötni. A térbeli pozíciók és a forgás pedig ahhoz kell, hogy a robotokat rá tudjuk rakni a gyárterületre. Ezek az értékek a virtuális robotok esetén a Robotstudio-ban lévő pozíció adatai ez egyes robotoknak. A forgatási érték pedig a függőlege Z tengely értéke fokban. A Z koordináta és X,Y forgatási érték most nem volt fontos. Itt még meg kell adni, egy legördülő listából a robot típusát és, hogy melyik gyárterületre rakjuk rá. Megemlítendő, hogy mikor elmentjük a PLC-t akkor megtörténik a QR kód generálás is a háttérben, ami a webszerver nyilvános gyökér könyvtárában a qr mappába kerül.

Adat kezelő szerver Robotok PLC-k

Új robot

GUID

Álnév

Robot térbeli X pozíciója

Robot térbeli Y pozíciója

Robot térbeli Z irányultsága

Robot típusa

IRB120

Gyár neve

Gyár_1

Mentés

Mégsem

Diplomamunka 2023 - Miholics Bálint

3.3. ábra: Új robot felvitel

A gyárakat is ki lehet listázni és innen lehet megnyitni az új gyár felvételénke az oldalát. Jelenleg több gyár most nincs használatban, de ide lehetne gyártelepek, vagy akár gyáraknak a szintjeit is külön felvinni.

Új hozzádaása

Gyár neve	Gyár szélessége	Gyár hosszúsága	
Gyár_1	5	5	Módosít Töröl

Diplomamunka 2023 - Miholics Bálint

3.4. ábra Gyárterület lista

Új gyárterületnél a neve és a fizikai mérete a felvihető értékek. A méret méterben értendő és a mobil alkalmazásban akár 1:1 méretben is megjeleníthető lenne, de ott le van méretezve.

Adat kezelő szerver Robotok PLC-k

Gyár felvétele

Gyár neve

Gyár szélessége

Gyár hosszúsága

Mentés

Mégsem

Diplomamunka 2023 - Mihalics Bálint

3.5. ábra Gyárterület felvitele

Az eddig felvitt robot típusok listája látható a következő képen. Az adatszerver nem tudja lekérni a vezérlőkről egyértelmű adatként, hogy milyen robottal is van dolga. A mobil alkalmazásban viszont tudnunk kell, hogy melyik robot modelljét töltsük be. Ehhez kell az itt felvitt adat. Az alkalmazásban a modell neve megegyezik ezzel az értékkel és annak megfelelően töltődik majd be.

Adat kezelő szerver Robotok PLC-k

Új hozzáadása

Robot típusa

IRB120

Módosít

Töröl

IRB14000

Módosít

Töröl

Diplomamunka 2023 - Mihalics Bálint

3.6. ábra Robottípusok listája

Ahogy alább látható itt csak egy szöveget kell megadni. Jelen esetben tudni kell, hogy alkalmazásban mi lesz a 3D modell neve és gyártót sem lehet választani, ami egy későbbi fejlesztési tervem. Illetve, ha későbbi fejlesztésként megoldom, hogy a 3D modelleket is a webszerver szolgáltatssa ki, akkor az e kényszer, hogy ügyelni kelljen a 3D modell nevére és az itt felvitt értékre megszűnik.

Robot típus felvitele

Robot típusa

Mentés

Mégsem

3.3.3. PLC-k menüpont

Ez a menüpont kilistázza a felvitt PLC-ket, illetve ez alól az oldal alól érhető el az új PLC, I/O port és PLC típus felvitelének a lehetősége is. Itt még nagyon hasznos funkció, hogy a felvitt PLC-nél legenerált QR kódot innen lehet letölteni, hogy például kinyomtassuk és ráragasszuk a gépre.

Új PLC			I/O portok			PLC típusok		
PLC neve	Cím/URL	Adatblokk száma	PLC típusa			QR kód		
PLC_1	192.168.100.100	1	S7-1200			Letöltés	Módosít	Töröl

3.7. ábra: PLC-k listája

Új PLC esetén azokat az adatokat lehet felvinni, amik nem az adatszerver futása közben íródnak az adott PLC rekordba. A PLC neve fog majd megjelenni a mobil alkalmazásban virtuális kivetített HMI-n, a cím fontos mert itt nem úgy mint a robotok esetében, közvetlenül ezzel a címmel kell csatlakozni a PLC-hez. A típus itt annyira nem fontos, de ez is egy későbbi fejlesztési lehetőségnek van fenntartva, mivel még az adat szerveren használt Snap7 könyvtár esetében is lehetséges, hogy bizonyos funkciók nem vagy nem teljesen azonos módon érhetőek el. Ezt tovább gondolva, pedig új gyártókat is fel lehetne vinni ahol aztán tényleg eltérő a PLC-k kezelése.

Új PLC

PLC neve

Cím/URL

PLC típusa

Adatblokk száma

Mentés

Mégsem

3.8. ábra: PLC felvitele

A felvitt I/O portok listája látható az alábbi képen. Innen lehet megnyitni az új portok felvitelének oldalát.

Új hozzáadása							
IO port neve	Irány	Eltolás	Bit hely (Big Endian)	Érték	Plc		
lámpa relé	Kimenet	0	0	0	PLC_1	Módosít	Töröl
lámpa gomb	Bemenet	1	0	0	PLC_1	Módosít	Töröl
kettes kimenet	Kimenet	0	1	0	PLC_1	Módosít	Töröl

3.9. ábra: I/O portok listája

A PLC önmagában semmit sem érne, ha nem tudjuk használni a ki és bemeneteit. Egy új port felvitelénél megadhatunk egy nevet, ami majd megjelenik az alkalmazásban, az irányt, hogy ki vagy bemenetről van szó és a bájtt adatokat, amik korábban már voltak tárgyalva. Az érték alapból a PLC-ből jön ,de itt akár egy kezdő értéket is megadhatunk indulásnak.

Új I/O

IO port neve

PLC neve

Irány

Eltolás

Bit hely (Big Endian)

Érték

Mentés

Mégsem

3.10. ábra: I/O port felvitele

4. Kiterjesztett valóság alkalmazás

A diplomamunkám fő terméke egy Android operációs rendszerre fejlesztett mobil alkalmazás, ami kiterjesztett valóság segítségével képes ipari folyamatokat vezérleni és azok állapotát megjeleníteni.

4.1. Kezelő felület

A következő fejezetekben bemutatom az alkalmazás felületét és hogy a program két menüpontja milyen funkciókat lát el.

4.1.1. Funkcionalitás

Az alkalmazás két kiterjesztett valóság módszert használ. Az egyik esetén képet nyomkövetünk, míg a másik esetben síkokat azonosítunk be és arra rakjuk a modelleket. Ezt az utóbbit környezet megértésének nevezzük, amiről a Google ARCore oldala következőt mondja:

„Az ARCore folyamatosan javítja a valós világ környezetének megértését a jellemző pontok és síkok felismerésével. Olyan jellemzőpontok csoportjait keresi, amelyek látszólag közös vízszintes vagy függőleges felületeken, például asztalokon vagy falakon helyezkednek el, és ezeket a felületeket geometriai síkokként teszi elérhetővé az alkalmazás számára. Az ARCore képes meghatározni az egyes geometriai síkok határait is, és ezt az információt elérhetővé teszi az alkalmazás számára. Ezt az információt felhasználhatja a sík felületeken nyugvó virtuális objektumok elhelyezésére.”

Bár ugyan azt a QR kódot használjuk az első eljárásban, vagyis a képazonosításban mint amiből a QR kód által tartalmazott információ kinyerjük, a kettőnek nincs köze egymáshoz, csak egy képpel két felhasználási kört is letudunk. A QR kód olvasása alaphoz nem része sem a Unity-nek sem az ARCore-nak, ezért ehhez egy külső csomagot kellett telepíteni, aminek a Unity-QR-Scanner a neve és a <https://github.com/nickdu088/Unity-QR-Scanner> oldalon érhető el. Mivel már sok szó esett QR kódról, röviden csak annyit róla, hogy ez egy kettő dimenziós bárkód, amit arra fejlesztettek, hogy géppel vizuálisan nagyon gyorsan leolvasható legyen és amire rárakják arról kiegészítő információt tudjon tárolni.

A robotok 3d modelljei, úgynevezett kapott és ezzel értem el a holografikus hatást, míg a Riptide Networking szintén egy külső csomag, ami pedig az adatszerver és az alkalmazás közötti adatcserét valósította meg. Az adatok JSON sztringként érkeznek és egy előre összeállított PLC és Robot osztályba kerülnek vissza serializálásra.

4.1.2. Használati esetek és menü struktúra

A program indulása után kettő menüpont közül lehet választani, illetve egy Beállítások menüpont van, amik működése a 4.2-es Program működése fejezetben lesznek részletesen bemutatva. Itt a használhatóságát fejtem ki az egyes menüeknek.

Kezdeként a Beállítások részt nézzük, ahol a Programnak meg lehet adni az adatszerver és a webszerver URL-jét. Ez a kettő lehet ugyan az is, de most korábban már kifejtett okok miatt [\(3.3.1\)](#) külön címen érhető el. Ez amúgy nem is rossz abból a szempontból, hogy egy cégnek sem feltétlenül jó, ha az adminisztrációs webszerver ugyan azon az IP címen érhető el, mint az adatokat szolgáló API szerver. Ezek lehetnek külön szerver gépeken is, amik növelik a rendszer külső behatolásával szembeni biztonságát.

A főmenüben lévő első menüelem az Eszköz leolvasása, ami egy olyan karbantartó vagy működést figyelő mérnöknek lehet jó, akinek egy zsúfolt gépben kell egy bizonyos PLC-ről adatokat lekérnie. Jelenleg természetesen, nagyon sok kiegészítő adatthoz nem jut, hiszen a PLC-n egyértelműen látszik, hogy melyik kimenet aktív, bár megjegyzendő, hogy a QR kódot, amit le lehet olvasni, azt nem feltétlenül muszáj közvetlenül PLC-re rakni, sőt lehet, hogy az eszközre nincs is közvetlen rálátás, így, hogy aktív-e a kimenet vagy sem szoftveres megjelenítése hasznos lehet. Kis módosítással, lehetőség lenne szenzorok és egyéb belső változók adatainak is a megjelenítésér, ami viszont tényleg kiegészítő információt nyújthat ott az eszköz mellett állva. Amire viszont már most is képes a program, hogy egyből lehet változtatni egy kimenet állapotát, mintha csak egy fizikai nyomógombot nyomnánk meg. Mind a kettő eset persze megvalósítható fizikai visszajelzőkkel (LED, lámpa, nyomógomb, kapcsoló, stb.) vagy HMI-vel, de ezek extra költsége, helyigénye és a jövőben való megváltoztathatósága lényegesen nehezebb mint az általam írt programmal.

A másik menüpont pedig a Gyárterület betöltése, amikor is egy előre megadott alapterületű (szélessége és hossza van tárolva az adatbázisban) gyárterületet reprezentáló síkot kell elhelyezni a kiterjesztett valóság által beazonosított talajra. Erre töltődnek be a robotok, amik a gyárterülethez tartoznak. Jelenleg csak egy gyárterület van és mindig az töltődik be, de később majd szeretném, ha felhasználói azonosítás után és választó menüből ki lehetne választani, hogy egy gyárnak melyik szintje vagy épülete töltődjön be. Ez hasznos lehet olyan szakembereknek, akiknek karbantartó munkát kell végezni egy roboton, és ez térképként megmutatja, hogy fizikailag hol helyezkedik el a gép. Ehhez persze még hasznos lehet, ha nem csak a robotok, hanem más gépek is látszódnának a kivetítésen, illetve megjelölné a felhasználót is, de ez most nem volt része a megvalósításomnak. Másik haszna lehet, ha gyorsan szeretnénk állapot információt kapni az adott eszközről és a rajta futó programról. Ez különösen nagy gyárterületen lehet

hasznos, ahol nehéz kiigazodni a sok gép között, illetve újonnan felvett szakember beoktatásakor segíthet neki eligazodni a gépek között. Másik haszna még ennek a menüpontnak, hogy oktatásban lehet szimulálni gépeket egy osztályteremben, marketing szempontból pedig reprezentálhatja a céget vagy ahova fejlődni szeretne.

4.2. Program működése

A következő alfejezetben először a mobil alkalmazás funkcióinak technikai megvalósítását vezetem le utána pedig a programból kimentett képeken végig vezetve magyarázom el a működését.

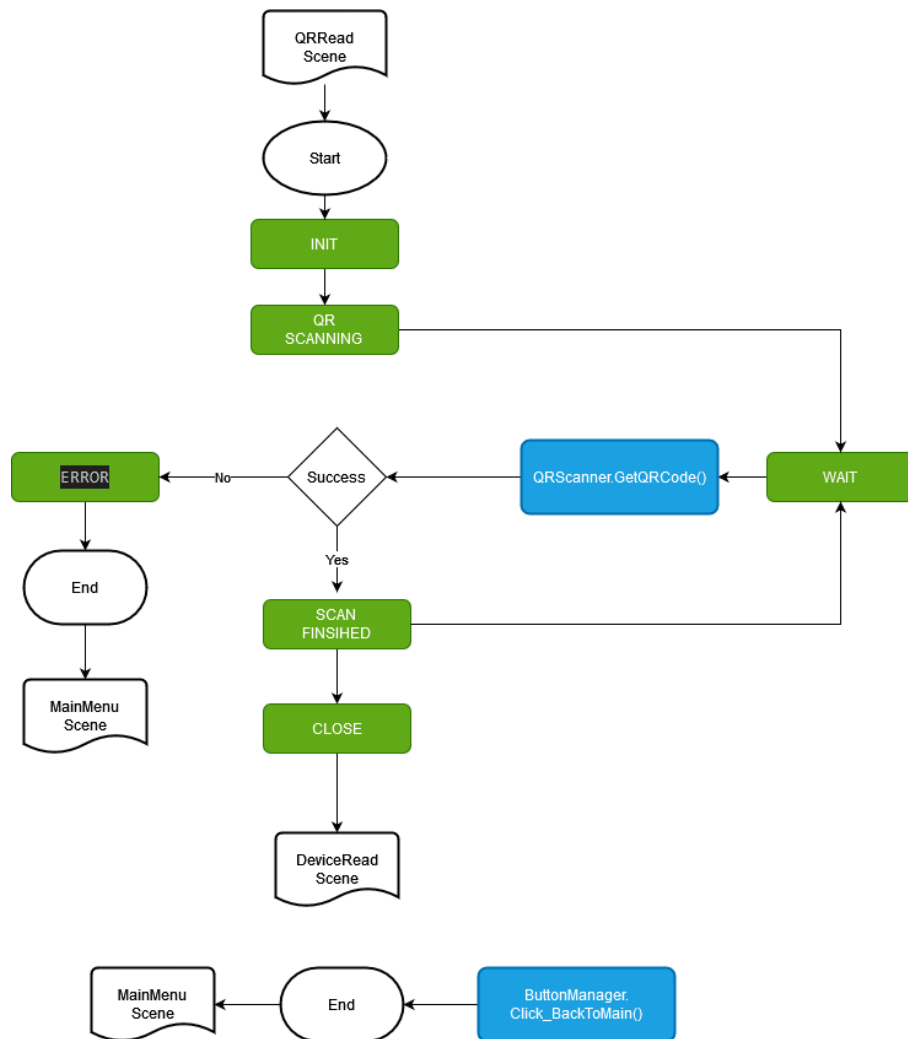
4.2.1. Állapotgép bemutatása

Ellentétben egy konzolos alkalmazással, ahol a program futása szekvenciális és egy belépő ágtól fut a metódus végéig – tipikusan egy *Main* metódus – Unity-ben az elkészült programnak nagyon sok feladatot kell párhuzamosan megoldani, amire nem sok ráhatásunk lehet. Ilyen a grafika, legyen az 2D vagy 3D, a fizika szimulációja vagy az általunk fejlesztett szkriptek is ilyenek. Minden szkript, amit egy GO-hoz kapcsolunk tartalmaz egy *Update* metódust, aminek jelentése frissítés. Nagyon leegyszerűsítve, ez annyiszor hívódik meg, ahányszor csak a program az adatott platformon/eszközön képes lefutni. Tehát nincs lehetőségünk várni a program egy bizonyos pontján, mint azt tennénk egy konzolos alkalmazásban, például a felhasználó döntésére egy menü esetében. Bár eseménykezelés van és a programban a felhasználói felületen a gombok ezt használják, de a program futását ezek sem állíthatják meg és nem is teljesen úgy működik, mint egy *Windows Form*-os ablakkezelős alkalmazásban.

Ahhoz, hogy a program bizonyos külső vagy belső hatásokra addig maradjanak egy ponton amíg kell és utána tovább léphessenek állapotgépet kell implementálni, ami a Unity-hez hasonló játékfejlesztő programokban gyakori megoldás. Ilyenkor az *Update* metódus azt figyeli, hogy a program futása közben milyen állapotban van a program és addig azt ismétli, amíg tovább nem léptetjük. Illetve, ha valamit csak egyszer kell lefuttatni és nem maradhatunk abban az állapotban akkor egy semleges, feladatot végre nem hajtó általam *Wait* vagyis várakozásnak elnevezett állapotba mozdítjuk a futást. Van olyan is amikor pedig nem léptethetjük a következő állapotba a programot, hanem egy

egyszeri beállítás elvégzésére, egy köztes állapotba lépünk és annak a lefutása billenti át a programot a tényleges feladatot végzőbe.

Gyakorlatilag a legtöbb állapot csak egy rövid beállítást és után egy külső hatásra váró várakozást lát el feladatul, egyedül a 3D játékokobjektumok és azok mozgását, kinézetét módosító *UPDATE*, vagyis frissítés állapot az, ami mindig önmagába tér vissza és jelenleg csak a program bezárása vagy a főmenübe való visszalépés szakítja meg futását.



4.1. ábra: QR kód leolvasás állapot gépe

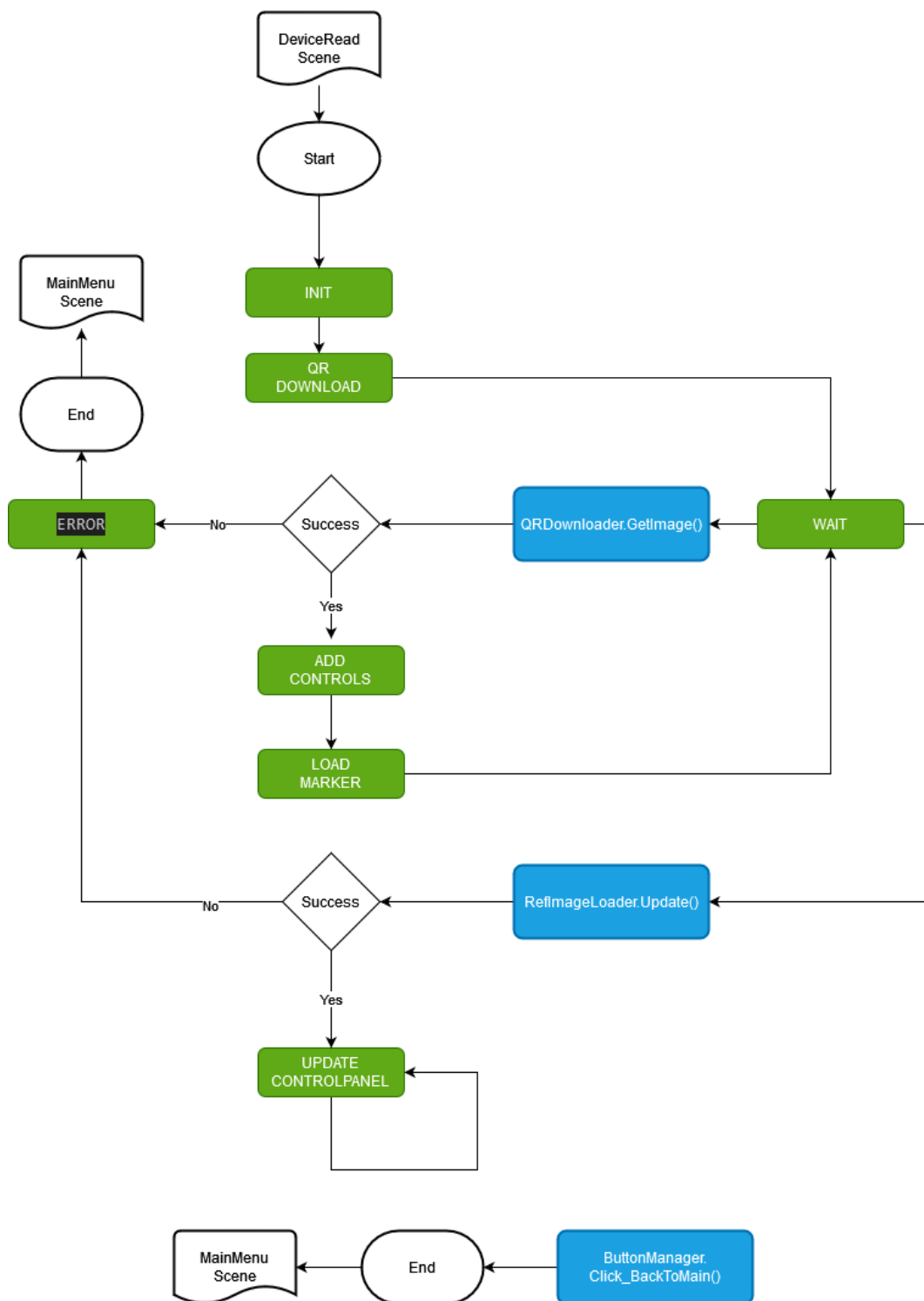
Az eszköz leolvasást kettő jelenetet igényelt, met így a QR kód leolvasását és ugyan ennek a kódnak letöltését szerverről külön lehet kezelni, amiből az első a 4.1-es ábrán látható. Először megnyitjuk a kamerát, amivel keressük a QR kódot, majd ha megvan akkor kiolvassuk a tartalmát, ami a PLC adatbázisban lévő azonosítója (Id). Ez után jön

a következő fázis, amikor le kell tölteni az ehhez az azonosítóhoz tartozó legenerált QR kódot képként.

Miért is kell letölteni, azt amit beolvastunk? Mert az ARCore-nak szüksége van egy képekre, hogy a 3D-s objektumokat arra vetítse ki. „Az Kiterjesztett Képek egy olyan funkció, amely lehetővé teszi, hogy olyan AR-alkalmazásokat hozzon létre, amelyek képesek reagálni bizonyos 2D-s képekre, például termékcsomagolásokra vagy filmplakátokra. A felhasználók AR-élményeket válthatnak ki, amikor a telefon kameráját bizonyos képekre irányítják - például a telefon kameráját egy filmplakátra irányíthatják, és egy szereplő felbukkan, és eljátszik egy jelenetet.” [11] A következő képen a kiterjesztett képek beolvasása és az ehhez a képhez kötött 3D objektumok betöltésének az állapot átmenetei láthatóak.

Először le kell tölteni a megfelelő képet, majd azt futásidőben meg kell adni referencia képnek. Általában nem szükséges futásidőben cserélni a referencia képet, ezért ez nem volt egyszerű feladat, de végül sikerült megoldani. Egy referencia könyvtárba, amúgy több kép is lehet, amihez egy objektum van kötve amit megjelenít, de most mindig csak egy képet töltünk bele.

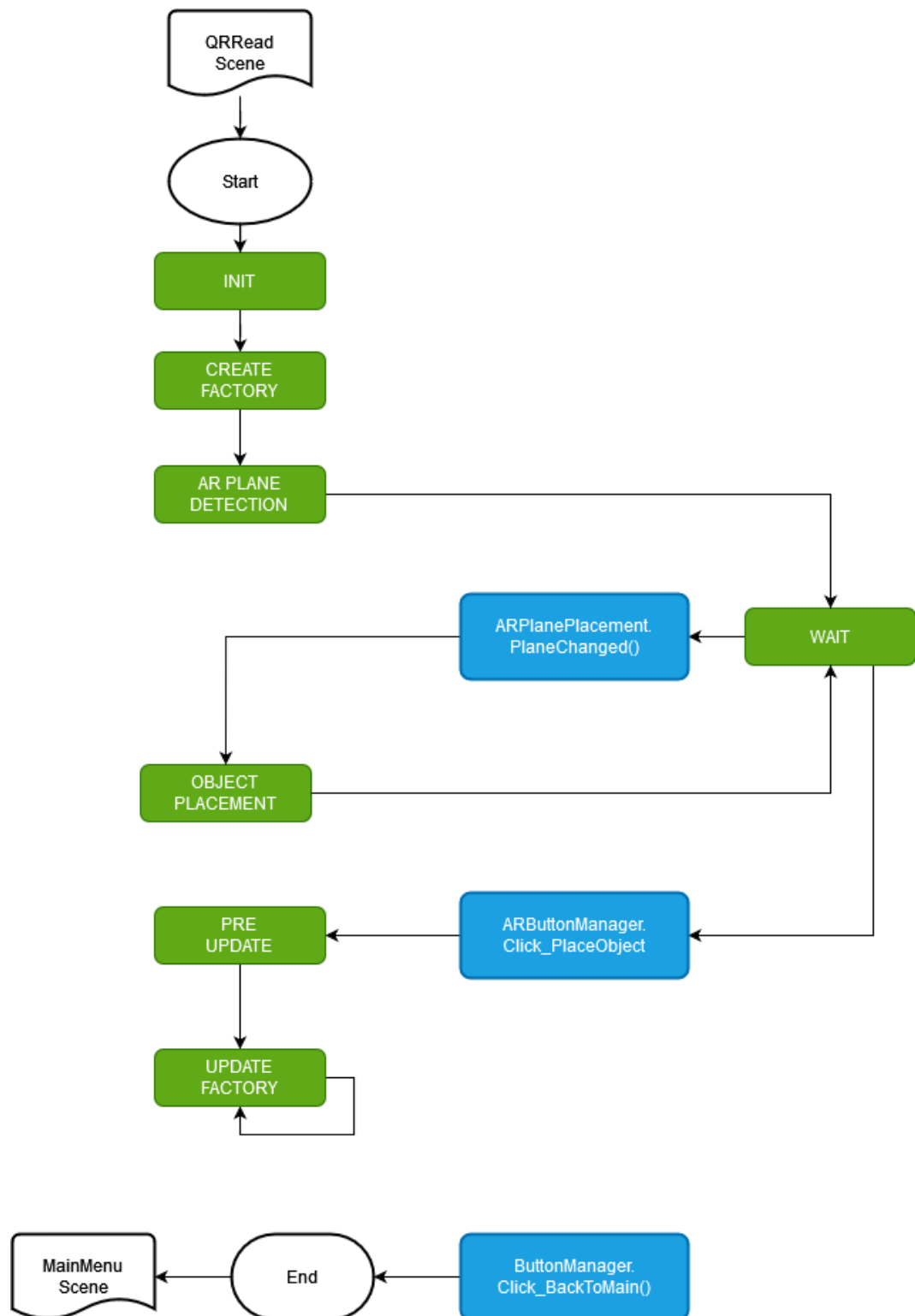
A 3D objektum a PLC virtuális HMI-je lesz az objektum és arra kerülnek a gombok, lámpák és szövegek, de ezt egy későbbi fejezetben részletezem.



4.2. ábra: QR leötlés és 3D objektum megjelenítés állapotai

A gyárterület beolvasáshoz és robotok megjelenítéséhez elég volt egy jelenet. Itt először összeállítjuk a gyárterületet és rá a robotokat, amiből egy GO jön létre majd ezt alapértelmezetten az eredeti mérethez képest egy harmadára méretezzük. Ez után várjuk, hogy az ARCore megtalálja a síkokat a kamera segítségével. Ha ez megvan akkor a

felhasználó rögzítheti a gyárterületet és ez után frissítjük a meglévő robotok adatait. Ennek a folyamatnak az ábrája alább látható.



4.3. ábra: Gyárterület kivétel állapottgép

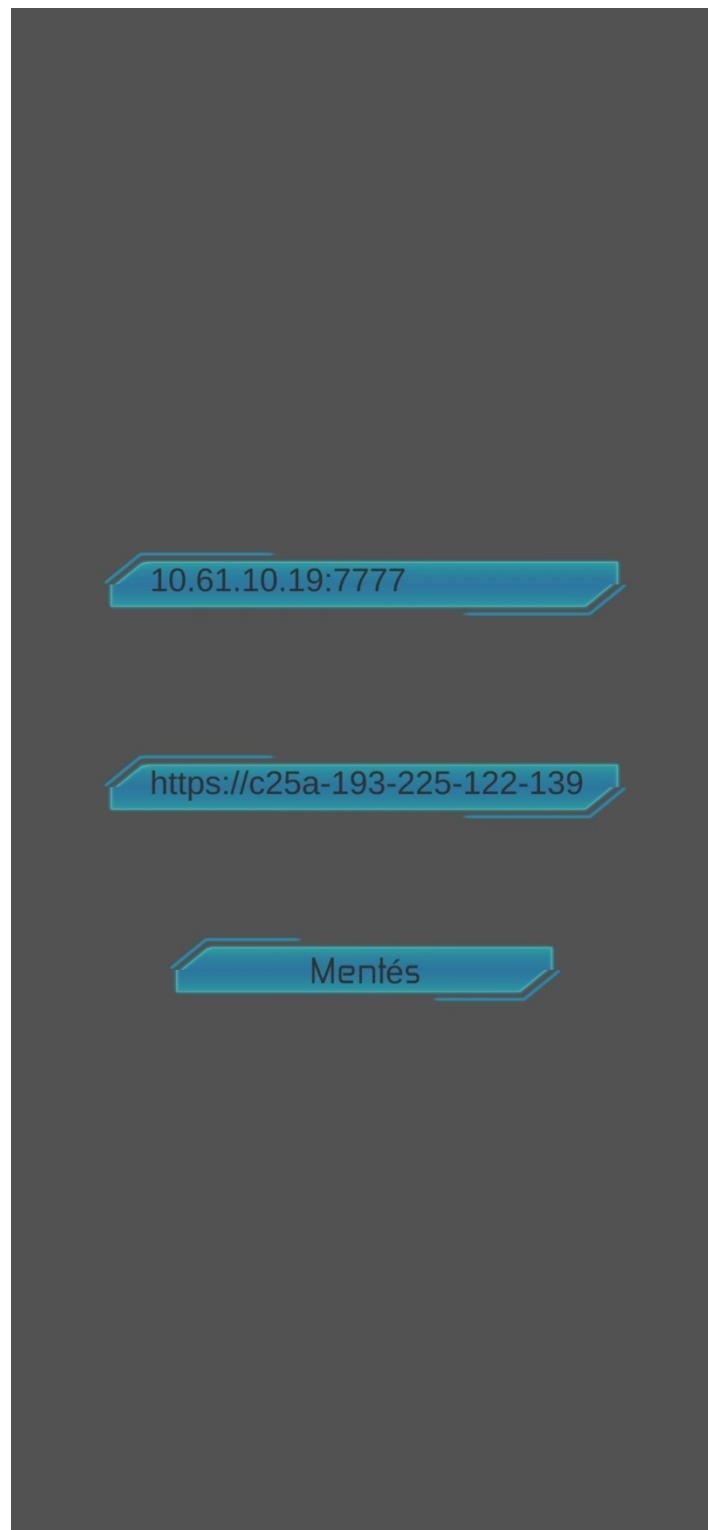
4.2.2. Főmenü és beállítások

A főmenüvel indul a program, ahol lehet választani, hogy eszköz leolvasás vagy gyárterület megjelenítés lesz. Innen érhetők még el a beállítások, amihez a fogaskerékre kell rányomni.



4.4. ábra: Főmenü szerkezete

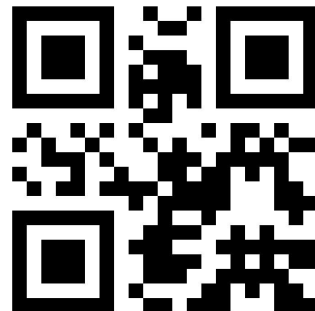
A beállítások alatt lehet megadni a webservert és az adatszerver címét. A webservert itt egy ngrok által adott URL, de ha egy szabadon konfigurálható alhálóban vagyunk megfelelő „*port forwarding*”-gal, akkor a két cím lehetne ugyan az is, csak a szerverek portjai térnének el.



4.5. ábra: Beállítások menü

4.2.3. Eszközök leolvasása és működtetése

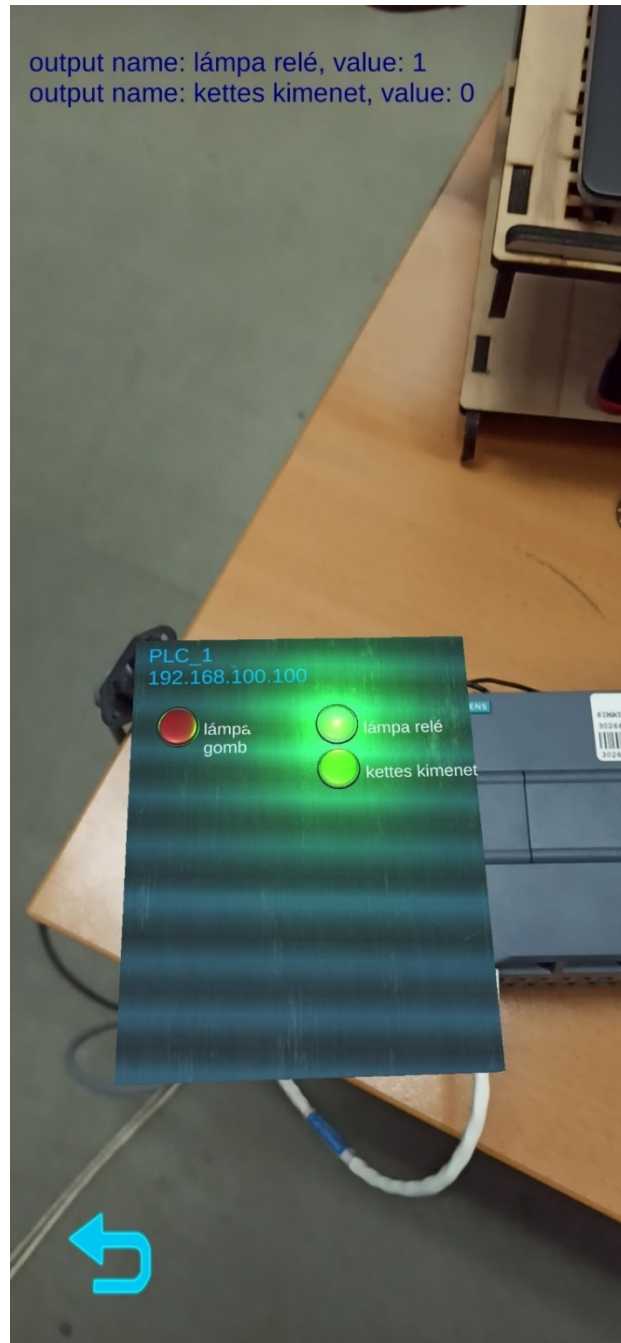
Az eszköz leolvasása menüelemet választva először egy kereső ablak jön be, aminek a közepén a kamera képe látható. Ezzel kell a QR kódot megkeresni és beolvasni. Ha ez megvan akkor értelmezi a tartalmát a kódnak és átlép a következő folyamatra.



4.6. ábra: QR kód leolvasás és maga a kód

A következő ábrán a tényleges legenerált kép látható, amelynek most itt a szövegtartalma egy kettes szám, a PLC azonosítója. Mint látható a kamera képe igencsak zajos ehhez az eredeti képhez képest, ezért nem is lett volna jó ez a 3D objektum nyomkövetéséhez. Ezért döntöttem az mellett, hogy akkor inkább töltsse le a program a webszerverről az eredeti, jó minőségű képet.

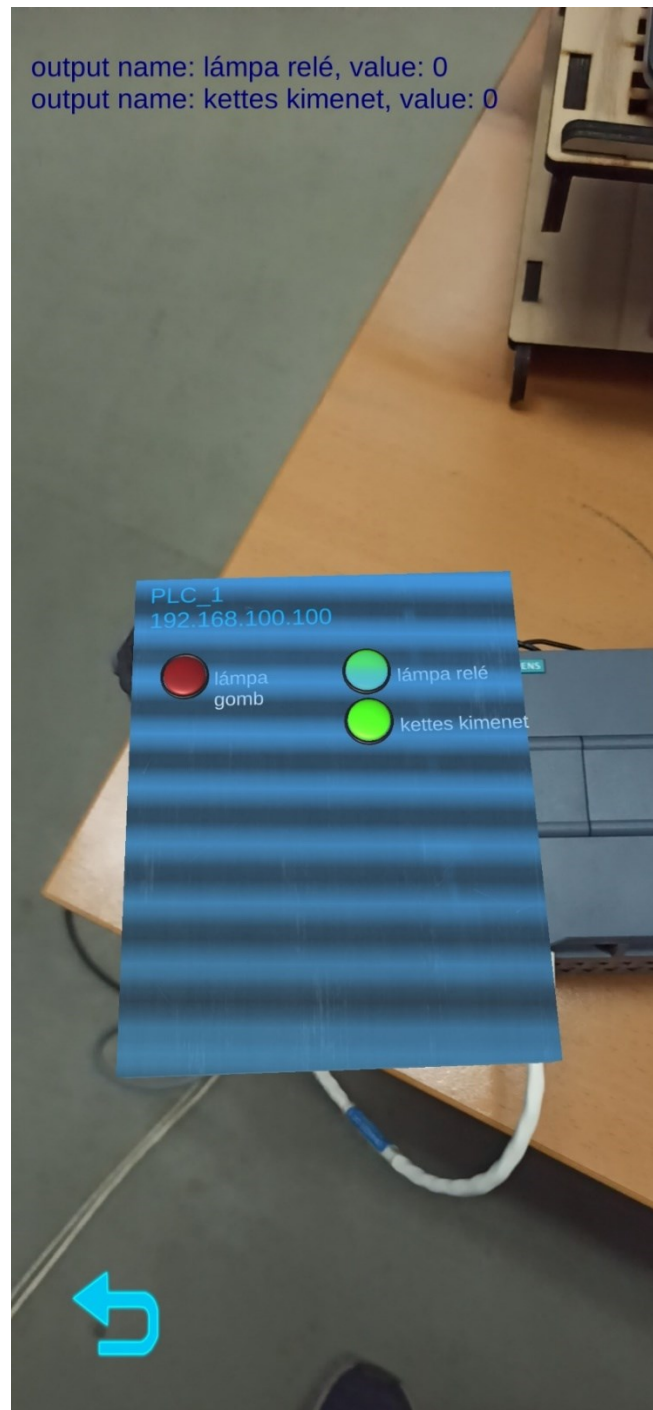
Ha már letöltődött webserverről képként a kód, akkor azt betöltjük, hogy nyomkövethető legyen és hozzárendeljük a PLC és az ahhoz tartozó I/O adatokkal összeállított 3D modellt, ahogy azt a következő kép is mutatja, ahol világít a lámpa kimenet virtuális visszajelző LED-je, amit ténylegesen megvilágít egy Unity-s fény komponens.



4.7. ábra: Betöltődött PLC, egyik kimenet aktív

A következő képen pedig az látható, hogy amint megnyomtuk a piros Lámpa gombot, a fény kialszik és a PLC-n is kikapcsol a hozzá tartozó kimenet. A második kimenetet előtt azért nincs gomb, mert szerettem volna reprezentálni, hogy ez egy

rugalmas rendszer. Csak az adatbázisba felvitt I/O elemek jelennek meg és annak megfelelően alakul a HMI is, ez nincs előre belekódolva.



4.8. ábra: Kialud a LED a lámpakimeneten

4.2.4. Virtuális gyárterület

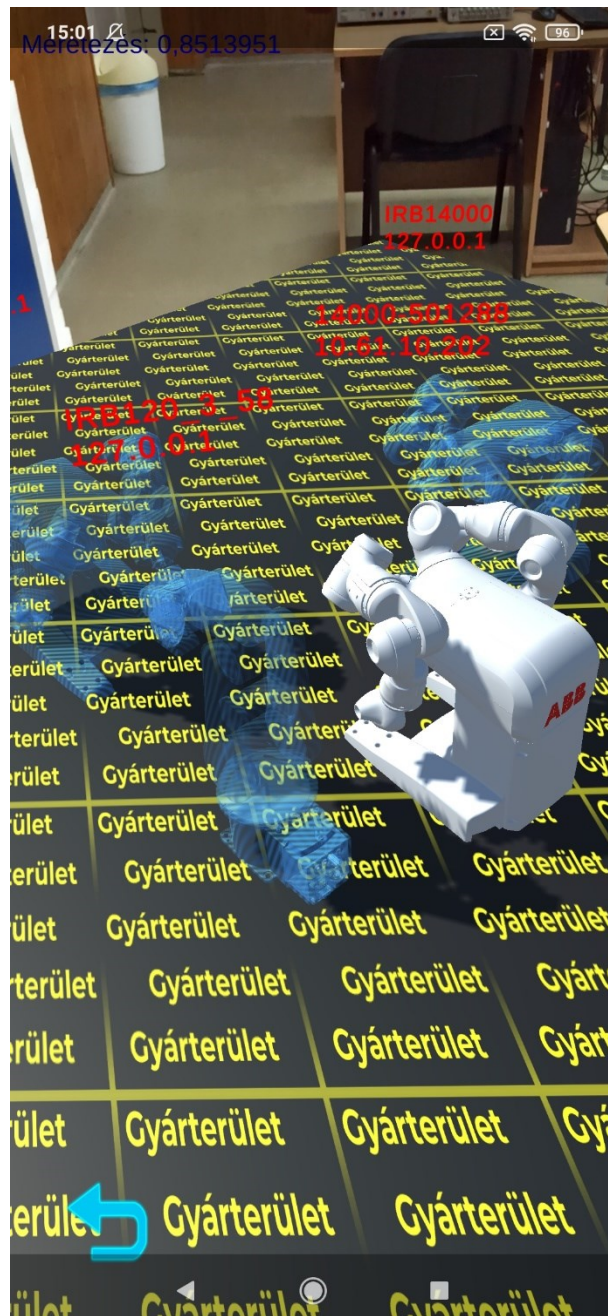
A virtuális gyárterület betöltődésekor először a telefon kameráját egy sík, lehetőleg jól megvilágított területre kell mutatni és ha talál értelmezhető egyenes felületet, akkor az előre letöltött gyár és robot adatok alapján összeállított 3D objektumot rávetíti erre síkra.



4.9. ábra: Virtuális gyárterület

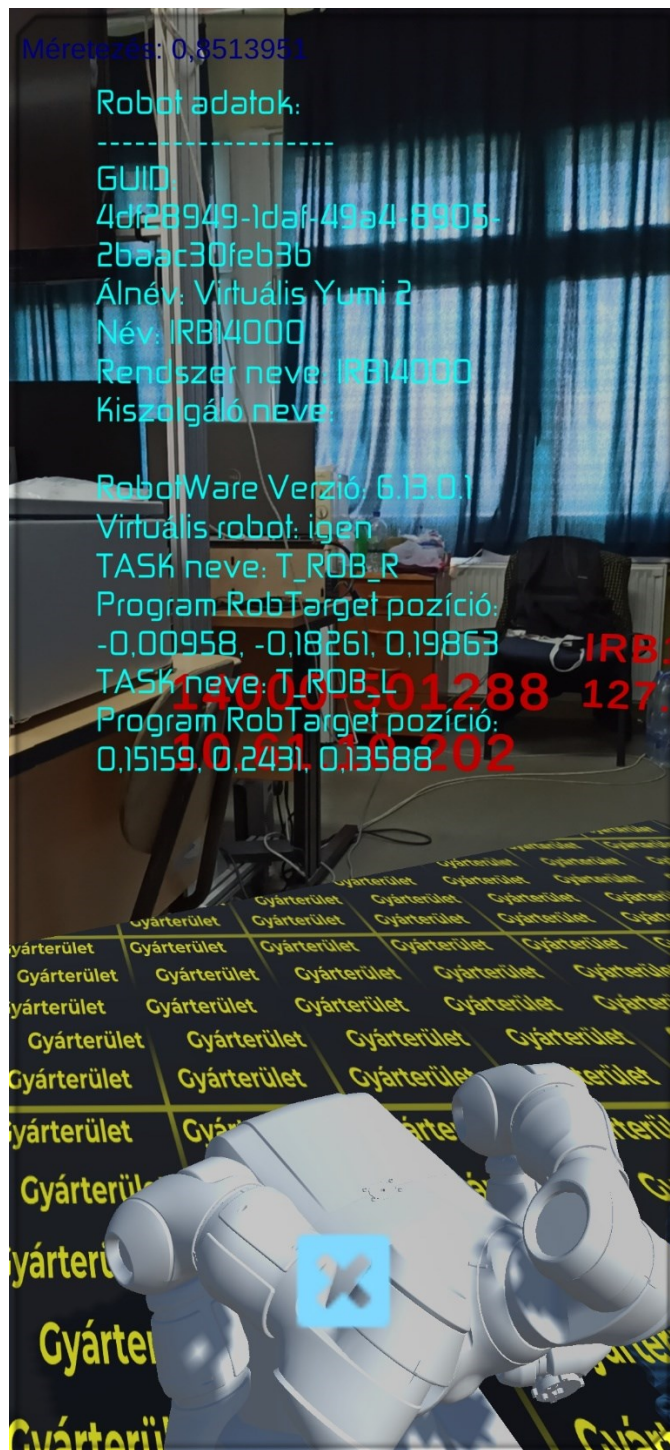
Ahogy a fenti ábrán látható a terület kisebb, mint ami az adatbázisban meg volt adva (5x5 méter) és a arányosan a rárakott robotok is kisebbek, de pozíciójuk egymáshoz képest és a terület origójához képest is arányos. Ez azért van mert 1/3-os léptékkel jelenik

meg a terület, hogy még könnyen kezelhető legyen. De telefonokról ismert két ujjas csípéssel nagyíthat és kicsinyíthető az egész modell.



4.10. ábra: Robotok fölé vetített információ

A 4.10-es ábrán az látható, hogy közelebb hajoltam a kivetített robotokhoz és a fölöttük pirossal olvasható nevük és az IP címük. Ez a felirat mindig a felhasználó felé néz. A virtuális robotok hologramként, míg az egyetlen igazi robot sima textúrával jelenik meg.



4.11. ábra: Részletes robot adatok

A 4.11-es képen az látható, hogy ha egy robotot a virtuális térben megérintünk, akkor a telefon képernyőjére (ez nem az AR térbe van kivetítve) kiírja a robot adatait és rajta futó program utoljára érintett szerszám középpont koordinátáját, az egyes feladatokhoz (TASK). Az információs panel bezárásához az X gombot kell megnyomni.

5. További célok és fejlesztési lehetőségek

A rendszert az elejétől fogva próbáltam rugalmasra tervezni, de a véges erőforrások miatt kénytelen voltam kompromisszumokat kötni. A jövőben újabb gyártók robotjait és PLC-it lehetne felvenni a támogatott eszközök közé, amikhez az adat szervert kell kibővíteni, hogy ezekkel az eszközökkel is képes legyen kommunikálni. Ebből a szempontból a mobilos alkalmazást még nem kellene módosítani, hiszen a szerver API-n keresztül meg van határozva, hogy minek milyen formában kell átmennie a mobil alkalmazásba.

De a robotok 3D-s modelljei jelenleg az alkalmazásba vannak tárolva, ezeket jó lenne, ha a jövőben a QR kódokhoz hasonlóan a webszerver szolgáltatná ki, így drasztikusan csökkenne a mobil alkalmazás mérete és ha ez meg lenne oldva akkor az alkalmazáshoz már tényleg nem kellene hozzányúlni újabb gyártók termékeinek a felvitele után.

Szeretném még megoldani, hogy a gyárterület több szintes legyen vagy akár több gyárterület között is lehessen válogatni. Ez vagy manuálisan vagy a felhasználó hitelesítése után automatikusan történne meg. Tehát csak az láthatja az adott terület gépeit, akinek arra jogosultsága van.

Végül pedig szeretném megoldani, hogy a robotok aktuális TCP pozíciója a kivetített gépeken is aktuális legyen. Ehhez inverz kinematikát kell használni és minden 3D robot modellt fel kell „rigging” -elni. Erre nincs magyar kifejezés, de ez azt jelenti, hogy egy belső csontvázat kap a modell, ahol a csontok és azokat összekötő ízületek egymáshoz képest kényszerekkel vannak ellátva. Tehát meg van határozva, hogy melyik csont meddig fordulhat vagy mozoghat a többihez képest. Az inverz kinematika a szerszám középpontjából képes visszaszámolni, hogy a robot többi tagja milyen pozícióban lehet. Mivel a TCP-t a roboton futó programból ki tudtam venni és át is juttatom az alkalmazásba, ezért ezt a nem könnyű, de látványos és egy karbantartó mérnök számára hasznos információt szeretném, ha majd meg tudnám jeleníteni a programban. Hiszen, ha a látja, hogy egy robot áll és robot karja milyen pozícióban áll, akkor abból már távolról következtethet, hogy mi lehet a hiba.

6. Összefoglaló

Diplomafeladatomban az kitűzött célokat sikerült elérnem, így létre tudtam hozni egy olyan okostelefonon futó alkalmazást, ami kiterjesztett valóság segítségével képes ipari folyamatokat megjeleníteni vagy akár azok működésébe bele is tud avatkozni.

Ennek a célnak ez eléréshez elkészítettem egy olyan adat szervert, ami API-ként és köztes interfészként működik az egyes gépek és vezérlők, valamint a mobil alkalmazás közé ékelődve. Ez a program képes olvasni és módosítani a Siemens PLC kimeneteit, valamint képes megjeleníteni egy háttér adatbázisban megfelelően felkonfigurált ABB robotokat, amik elérhetőek a szerver hálózatában.

Az egyszerű PLC és robot menedzsmenthez létre tudtam, hozni egy web-es felületet, amin keresztül könnyedén lehet újabb eszközöket felvinni az adatbázisba és azokat módosítani vagy törölni.

Irodalomjegyzék

- [1] Google, „ARCore support,” [Online]. Available: <https://developers.google.com/ar/devices>.
- [2] Siemens, Simatic S7 EasyBook.
- [3] K. Béla, Robottechnika 1.
- [4] ABB, Application manual IRC5.
- [5] ABB, Operating Manual IRB 14000.
- [6] J. M. a. T. N. Paris Buttfield-Addison, Unity Game Development Cookbook, O'Reilly, 2019.
- [7] ELTE, „A GLSL programozási nyelv,” [Online]. Available: <http://nyelvek.inf.elte.hu/leirasok/GLSL/index.php?chapter=1>.
- [8] PomeloFoundation, „Pomelo.EntityFrameworkCore.MySql,” [Online]. Available: <https://github.com/PomeloFoundation/Pomelo.EntityFrameworkCore.MySql>.
- [9] J. P. SMITH, Entity Framework Core in Action, M A N N I N G.
- [10] A. Lock, ASP.NET Core in Action 2. edition, Manning.
- [11] Google, “AR Core - Augmented Images,” 2022. [Online]. Available: <https://developers.google.com/ar/develop/fundamentals>.

Mellékletek

```
+---App
| | Assembly-CSharp-firstpass.csproj
| | Assembly-CSharp.csproj
| | DiplomaApp.sln
| |
+---Assets
| | +---Materials
| | | | FactorFloorBG.mat
| | | | FactoryFloorMaterial.mat
| | | | Grey.mat
| | | | plscreen.mat
| | | | Red.mat
| | | | RobotHoloMaterial.mat
| | | | settings_mod.png
| | | | smartphone.png
| | | | White.mat
| | | | \---Materials
| | | | | dashboard-l_preview.mat
| | +---Models
| | | | ABB_Base_Material.mat
| | | | ABB_Logo_Material.mat
| | | | RobotArm_rigged.fbx
| | | | YUMI.fbx
| | | | \---button
| | | | | button.blend
| | | | +---Base Material
| | | | | | Base.mat
| | | | | | Button.mat
| | | | | | ButtonBaseTex.png
| | | | | \---Button Material
| | | | | | ButtonTex.png
| | | | | | Light_Matalliic.png
| | | | | \---Materials
| | | | | | ButtonTex.mat
| | | | | | Light_Matalliic.mat
| | +---Prefabs
| | | | AR Session Origin.prefab
| | | | ARDetectPanel.prefab
| | | | button.prefab
| | | | FactoryFloor.prefab
| | | | GUI.prefab
| | | | Light.prefab
| | | | PLCHMI.prefab
| | | | QRCanvas.prefab
| | | | status.prefab
| | | | _ButtonManager.prefab
| | | | \---robotok
| | | | | IRB120.prefab
| | | | | IRB14000.prefab
| | +---Resources
| | | +---qr-scanner
| | | | zxing.dll
| | | \---Riptide
| | | | RiptideNetworking.dll
| | | | RiptideNetworking.xml
| | +---Scenes
```

```

| | | DeviceRead.unity
| | | Factory.unity
| | | MainMenu.unity
| | | QRRead.unity
| | | sample.unity
| | | SettingsMenu.unity
| | +---Scripts
| | | ButtonToggle.cs
| | | Settings.cs
| | | _ButtonManager.cs
| | +---DeviceRead
| | | ButtonControl.cs
| | | DeviceRead_Program.cs
| | | QRDownloader.cs
| | | RefImageLoader.cs
| | | RiptideServiceDevice.cs
| | +---Factory
| | | ARPlanePlacement.cs
| | | AutoPlaceObject.cs
| | | Factory_Program.cs
| | | RiptideServiceFactory.cs
| | | RobotDetail.cs
| | | _ARButtonManager.cs
| | +---Main
| | | PermissionManager.cs
| | +---Models
| | | Plc.cs
| | | Robot.cs
| | +---QRreader
| | | QRScanner.cs
| | | QR_Program.cs
| | \---Settings
| | | DataServerUrlLoad.cs
| | | WebServerUrlLoad.cs
| | +---Shaders
| | | factory.floor.shader.shadergraph
| | | holo.shadergraph
| | +---Textures
| | | Factor_texture_base.png
| | | HologramLines_Cool.psd
| | | HologramLines_Simple.png
+---PLC
| \---IndustrialDemoController
| | IndustrialDemoController.ap15_1
| | -projekt fájlok -
+---Robotstudio
| \---Solution5
| | - projekt fájlok -
\---Szerverek
+---adatbázis
| | diploma_db.sql
+---CentralServer
| | ngrok.exe
| | Program.cs
+---Models
| | DiplomaContext.cs
| | Factory.cs
| | IoPort.cs
| | Plc.cs

```

```

| | PlcType.cs
| | Robot.cs
| | RobotType.cs
| +---Pages
| | FileApi.cs
| | _Index.cshtml
| | _Index.cshtml.cs
| | _ViewImports.cshtml
| | _ViewStart.cshtml
| +---plcs
| | Create.cshtml
| | Create.cshtml.cs
| | Delete.cshtml
| | Delete.cshtml.cs
| | Edit.cshtml
| | Edit.cshtml.cs
| | Index.cshtml
| | Index.cshtml.cs
| +---ioports
| | Create.cshtml
| | Create.cshtml.cs
| | Delete.cshtml
| | Delete.cshtml.cs
| | Edit.cshtml
| | Edit.cshtml.cs
| | Index.cshtml
| | Index.cshtml.cs
| \---plctype
| | Create.cshtml
| | Create.cshtml.cs
| | Delete.cshtml
| | Delete.cshtml.cs
| | Edit.cshtml
| | Edit.cshtml.cs
| | Index.cshtml
| | Index.cshtml.cs
| +---robot
| | Create.cshtml
| | Create.cshtml.cs
| | Delete.cshtml
| | Delete.cshtml.cs
| | Edit.cshtml
| | Edit.cshtml.cs
| | Index.cshtml
| | Index.cshtml.cs
| +---factory
| | Create.cshtml
| | Create.cshtml.cs
| | Delete.cshtml
| | Delete.cshtml.cs
| | Edit.cshtml
| | Edit.cshtml.cs
| | Index.cshtml
| | Index.cshtml.cs
| \---robottype
| | Create.cshtml
| | Create.cshtml.cs
| | Delete.cshtml
| | Delete.cshtml.cs
| | Edit.cshtml

```

```

| | | Edit.cshtml.cs
| | | Index.cshtml
| | | Index.cshtml.cs
| | \---Shared
| | | _Layout.cshtml
| | | _Layout.cshtml.css
| | | _ValidationScriptsPartial.cshtml
| | \---wwwroot
| | | \---qr
| | | 2.jpg
+---+DataServer
| | | DataServer.csproj
| | | GlobalSuppressions.cs
| | | Program.cs
+---+ModelExtensions
| | | Robot.cs
+---+Models
| | | DiplomaContext.cs
| | | Factory.cs
| | | IoPort.cs
| | | Plc.cs
| | | PlcType.cs
| | | Robot.cs
| | | RobotType.cs

```

Ábrajegyzék

2.1 Rendszerterv	16
2.2. ábra: S7-1200 PLC, kimenetei	17
3. ábra: PLC memória címkék	18
4. ábra: Adatblokk tartalma	19
5. ábra: PLC-n futó program.....	19
2.6. ábra: Robotok a RobotStudio-ban.....	20
2.7 ábra: ABB IRB 14000 weboldala	22
2.8 ábra: Ipari labor ABB IRB14000 robotja	23
2.9. ábra: Szkript életciklusa (forrás: https://docs.unity3d.com/Manual/ExecutionOrder.html)	25
2.10. ábra: ShaderGraph működés közben	25
2.11. ábra: Adatbázis táblák kapcsolata.....	27
2.12. ábra: Robotkar riggel-ve	28
3.1. ábra: ngrok webservert alagút	35
3.2. ábra: Robotok listája	36
3.3. ábra: Új robot felvitel.....	37
3.4. ábra Gyárterület lista.....	37
3.5. ábra Gyárterület felvitele	38
3.6. ábra Robottípusok listája	38
3.7. ábra: PLC-k listája	39
3.8. ábra: PLC felvitele	40
3.9. ábra: I/O portok listája	40
3.10. ábra: I/O port felvitele.....	41
4.1. ábra: QR kód leolvasás állapot gépe.....	45
4.2. ábra: QR leolvasás és 3D objektum megjelenítés állapotai	47
4.3. ábra: Gyárterület kivetítése állapotgép	48
4.4. ábra: Főmenü szerkezete.....	49
4.5. ábra: Beállítások menü.....	50
4.6. ábra: QR kód leolvasás és maga a kód	51
4.7. ábra: Betöltődött PLC, egyik kimenet aktív	52
4.8. ábra: Kialud a LED a lámpakimeneten.....	53

4.9. ábra: Virtuális gyárterület	54
4.10. ábra: Robotok fölé vetített információ	55
4.11. ábra: Részletes robot adatok	56