



SAPIENTIA
ERDÉLYI MAGYAR
TUDOMÁNYEGYETEM
Marosvásárhelyi Kar

TUDOMÁNYOS DIÁKKÖRI DOLGOZAT

FELHŐALAPÚ TÁVVEZÉRLÉS ÉS OPCUA INTEGRÁCIÓ IPARI ROBOTOKNÁL

Szerző Keresztes Kristof

Mechatronika BSc, IV. évf.

Konzulensek Dr. Forgó Zoltán

Egyetemi Docens

Dr. Fehér Áron

Adjunktus

SAPIENTIA ERDÉLYI MAGYAR TUDOMÁNYEGYETEM

**FELHŐALAPÚ TÁVVEZÉRLÉS ÉS OPCUA INTEGRÁCIÓ
IPARI ROBOTOKNÁL**

**CLOUD BASED REMOTE CONTROL AND OPCUA
INTEGRATION FOR INDUSTRIAL ROBOTS**

Keresztes Kristof

Konzulensek:

Dr. Forgó Zoltán

Dr. Fehér Áron

Kézirat lezárva: 2024.04.10

Abstract

This paper aims to present the development process of a remote-control application for Universal Robots™. A physical UR10e industrial robot and the URSim simulation environment were used for development and testing. The application allows remote control of the robot via either a cloud-based Flask server with a graphical user interface, or a local OPCUA server. The OPCUA server was also used to implement the digital twin of the physical UR10e robot in the Visual Components environment. The different modules of the software package were written in Python programming language. The communication over the local network, such as between the OPCUA server and the different software modules, was realized using ZeroMQ framework for simplicity and to minimize remote control latency. To run the Flask application, Microsoft's Azure services were used, which allowed remote control of the robot over the Internet. Furthermore, the Flask application was extended with video monitoring functionality as well. The focus during the development of the application was on simple and fast communication, modularity, and extensibility. The system produced promising initial results in regards of control latency.

Keywords: Universal Robots, Python, OPCUA, Flask, remote-control

Kivonat

A dolgozat célja egy távvezérlési alkalmazás fejlesztésének bemutatása Universal Robots™ típusú ipari robotok számára. A fejlesztés és tesztelés során egy fizikai UR10e típusú kollaboratív ipari robot, illetve az URSim szimulációs környezet volt használva. Az elkészített alkalmazás lehetővé teszi a robot távvezérlését egy Flask szerver grafikus felhasználói felületén keresztül, illetve az integrációját lokális OPCUA környezetbe megfigyelés és irányítás céljából egyaránt. Az OPCUA szerver segítségével a fizikai UR10e robot digitális ikerpárja is megvalósult Visual Components környezetben. A szoftver csomag különböző moduljai Python programozási nyelven voltak megírva. A különböző modulok közötti kommunikáció ZeroMQ keretrendszer felhasználásával volt megvalósítva az egyszerűség, illetve a távvezérlési késleltetés minimalizálásának érdekében. A Flask alkalmazás futtatására a Microsoft Azure környezet volt használva, ami megengedte a robot távvezérlését az interneten keresztül. Továbbá, a Flask alkalmazást bővítettük videómegfigyelő funkcióval is. Az alkalmazás fejlesztése során a hangsúly az egyszerű és gyors kommunikációra, modularitásra, illetve továbbfejleszthetőségre esett. A kifejlesztett rendszer ígéretes kezdetleges eredményeket produkált az irányítási késleltetés tekintetében.

Kulcsszavak: Universal Robots, Python, OPCUA, Flask, távvezérlés

Tartalomjegyzék

1.	Bevezető.....	8
1.1.	Universal Robots™ és UR Script	9
1.2.	RTDE keretrendszer.....	9
1.3.	OPCUA	10
1.4.	Flask keretrendszer	11
1.5.	ZeroMQ.....	12
2.	Célkitűzés és hipotézis	13
3.	Metodológia	14
3.1.	Rendszertervezés és architektúrális megfontolások.....	15
3.2.	Szoftvertervezés és implementáció.....	19
3.2.1.	Bridge modul	19
3.2.2.	OPCUA szerver	20
3.2.3.	Flask alkalmazás	21
3.2.3.1.	Flask Szerver (back-end)	21
3.2.3.2.	Felhasználói felület (front-end).....	22
3.2.3.3.	Videómegfigyelő rendszer	24
3.2.4.	MiddleWare	24
3.3.	Kommunikáció és integráció	25
3.3.1.	Flask szerver	26
3.3.1.1.	Dinamikus IP cím kezelése a felhőben	27
3.3.2.	A robot digitális ikerpárja	28
3.4.	Teljesítményoptimalizáció	29
3.5.	Adatgyűjtés és értékelés.....	33

4.	Konklúzió.....	34
5.	Továbbfejlesztési lehetőségek	34
5.1.	Webbiztonság.....	34
5.2.	Erő-visszacsatolás (force-feedback)	34
5.3.	Különböző vezérlési modulok	35
6.	Köszönetnyilvánítás.....	35
7.	Irodalomjegyzék	36

Ábrajegyzék

1. ábra	UR10e kollaboratív robot [12].....	14
2. ábra	Az UR10e interfész működése és összekötése a fejlesztett alkalmazással.....	16
3. ábra	A problematikus pont a Bridge és a szerverek között.....	16
4. ábra	A MiddleWare modul elhelyezése a fejlesztett rendszer diagrammjában	17
5. ábra	Lokális és felhőalapú Flask szerver csatolása.....	18
6. ábra	Megfigyelőkamerák implementálása	18
7. ábra	OPCUA szerver modul belső szerkezete, feltüntetve az interfészt.....	20
8. ábra	A Flask felhasználói felület – főoldal (képernyőmentés)	21
9. ábra	A Flask felhasználói felület – Joint Control oldal (képernyőmentés).....	23
10. ábra	A Flask felhasználói felület – videófelügyeleti oldalak (képernyőmentés)	24
11. ábra	A rendszer két különböző irányítási módszere.....	24
12. ábra	Lekérdezés mechanizmusa	28
13. ábra	OPCUA értékek Visual Components környezetben (képernyőmentés, részlet)	29
14. ábra	Az egy számítógépen (PC) futtatható modulok.....	31

1. Bevezető

A XXI. század technológiai forradalma új kihívások elé állítja a vállalkozásokat és iparágakat, különösen azokat, amelyek folyamatosan törekednek a hatékony termelésre. Az ipari robotok elterjedése jelentős mértékben hozzájárult a gyártási folyamatok átalakításához és optimalizálásához. Ezek a gépek nem csupán a monoton és ismétlődő feladatokat képesek hatékonyan végrehajtani, hanem növelik a termelés sebességét és precizitását is. Az emberi erőforrások felszabadulása révén a szakemberek képesek a komplexebb feladatokra koncentrálni, ami hozzájárul az innováció és a kreativitás fokozódásához.

Az automatizálás további fejlődése azonban nem csak a robotok funkcionalitásának növelését hozza magával, hanem egyre inkább a távvezérlés lehetőségének széleskörű alkalmazását is. Ez új lehetőségeket nyit meg a robotok működtetése és programozása terén, lehetővé téve a távoli felügyeletet és irányítást. A távvezérlés rugalmas munkarendeket tesz lehetővé, lehetővé teszi a szakértők távoli bevonását a problémamegoldásba, és alacsonyabb működési költségeket eredményezhet.

A felhőalapú távvezérlés kulcsfontosságú innováció, amely hatékony és kifinomult módon hozzájárul a termelési folyamatok optimalizálásához és rugalmasságának növeléséhez. A rendszer lehetővé teszi a robotok távoli irányítását és felügyeletét bárhol a világon, ahol internetkapcsolat áll rendelkezésre. Ez különösen hasznos olyan szituációkban, amikor a szakértők távoli helyen tartózkodnak, de szükség van azonnali beavatkozásra vagy monitorozásra. A felhőalapú távvezérlés lehetőséget teremt a robotok egyszerű és gyors konfigurálhatóságára. Az alkalmazások, paraméterek és feladatok távolról való módosítása vagy frissítése egyszerűbbé válik, minimalizálva a leállási időt és növelve a rendszerek reagálóképességét. Ezenkívül a felhőalapú távvezérlés lehetővé teszi a nagy mennyiségű adatok gyűjtését és elemzését, amelyekből értékes információk származhatnak a hatékonyság optimalizálásához és a folyamatok fejlesztéséhez. Felhőalapú platformokat biztosítanak az olyan szolgáltatások, mint a Microsoft Azure, az Amazon Web Service (AWS), vagy az újonnan kidolgozott UR Cloud Platform. A felsoroltak közül az utóbbi kimondottan robot alkalmazások felügyeletére és módosítására volt kifejlesztve.

Összességében a felhőalapú távvezérlés korszerű alapot teremt a decentralizált gyártási rendszerek számára, lehetővé téve a hatékonyabb, költséghatékonyabb és rugalmasabb termelési folyamatokat.

1.1. Universal Robots™ és UR Script

Az Universal Robots (UR) által gyártott robotok az ipari automatizálás egy új dimenzióját képviselik, kimagaslóan rugalmas és könnyen programozható kialakításuk révén. Az UR Robotok az ún. "cobots" (kollaboratív robotok) kategóriáját erősítik, melyek a tradicionális ipari robotoktól eltérően képesek együttműködni az emberi munkavállalókkal anélkül, hogy bonyolult biztonsági intézkedéseket igényelnének [1].

Továbbá, az UR robotok kifejezetten bővíthetők és alkalmazkodhatók az ipari környezet változatos igényeihez. A bővítési lehetőségek közé tartoznak az ipari standardoknak megfelelő digitális és analóg ki/bemenetek, valamint kommunikációs protokollok, mint például a Modbus, Profinet és Ethernet. Ezek lehetővé teszik a robotok számára, hogy széleskörűen integrálódjanak különböző automatizálási rendszerekbe és kapcsolódjanak más ipari eszközökhöz, maximalizálva ezzel az alkalmazási területeiket és funkcionalitásukat [2], [3].

Az UR Robotok kiemelkedő teljesítményük mellett a saját fejlesztésű programozási nyelvükkel, az URScript-tel is rendelkeznek. Az URScript kifejezetten a UR robotjainak programozására lett tervezve, és lehetőséget kínál a felhasználóknak arra, hogy intuitív és hatékony módon irányítsák és testre szabhassák robotjaik működését. Az URScript programozási nyelv rendkívül rugalmas és könnyen tanulható, így még azok számára is hozzáférhető, akik nem rendelkeznek mély programozási háttérrel. Az egyszerű szintaxis és a sokoldalú parancsok segítik a felhasználókat a gyors fejlesztésben és alkalmazások hatékony kialakításában. Emellett az URScript támogatja a változók, ciklusok és feltételes utasítások használatát, lehetővé téve a programozók számára a robotmozgások precíz vezérlését és a komplex feladatok programozását [4].

1.2. RTDE keretrendszer

A valós idejű adatcsere (Real-Time Data Exchange – RTDE) egy szoftvertervezési keretrendszer és kommunikációs protokoll, amely lehetővé teszi a valós idejű adatok cseréjét az UR robotvezérlő

és külső eszközök, számítógépek vagy más rendszerek között egy szabványos TCP/IP kapcsolaton keresztül. Azonban fontos megjegyezni, hogy a valós idejű időskála relatív, és az alkalmazás kontextusától függően változhat. Ennek megfelelően, a valós idejű adatcsere fogalmát az alkalmazás konkrét követelményei határozzák meg, és az adott feladathoz igazodva definiálható pontosan.

Esetünkben, az e-szériás robotokkal, az RTDE keretrendszer 500 Hz-es kommunikációt biztosít, míg a régebbi CB-szériás robotokkal 125 Hz-el tud adatcserét fenntartani. Figyelemmel kell lennünk viszont arra a tényre, hogy a kontroller belső, valós idejű folyamata prioritást élvez az RTDE interfésszel szemben. Abban az esetben, ha egy adatcsomag blokkolná a robotkontrollert, ezt el fogja dobni annak érdekében, hogy megtartsa a saját valós idejű működését. Az eldobott adatcsomagok nem lesznek később elküldve, a kontroller mindig a legfrissebb információval szolgál [5].

Az RTDE keretrendszer lehetővé teszi az UR vezérlő külső alkalmazásokkal való szinkronizálását, anélkül, hogy a vezérlő valós idejű tulajdonságai sérülnének. Az RTDE-interfész különböző funkciókat támogat, beleértve a fieldbus illesztőprogramokkal (pl. Ethernet/IP) való interakciót, a robot digitális kimeneteinek manipulálását és a robot állapotának ábrázolását (pl. robotpályák). Ezáltal a külső eszközök képesek folyamatosan monitorizálni és befolyásolni a robot állapotát, illetve adatait valós időben, lehetővé téve számos alkalmazás megvalósítását, például valós idejű robotmegfigyelés, adatgyűjtés, valamint a robot mozgásának terén. Az RTDE-interfész alapértelmezés szerint akkor érhető el, ha az UR-vezérlő fut. A keretrendszer segítségével a robotok könnyebben illeszthetők be a teljes automatizált rendszerekbe, és a felhasználók számára széles körű vezérlési és monitorozási lehetőségeket biztosít [5].

Összességében az RTDE egy olyan eszköz, amely elősegíti a valós idejű adatok átvitelét és megosztását az UR robotok és más rendszerek között, így lehetővé téve a hatékonyabb és testreszabhatóbb ipari automatizálást.

1.3. OPCUA

Az OPCUA (Open Platform Communications Unified Architecture) egy nyitott ipari kommunikációs szabvány, amely lehetővé teszi az ipari rendszerek, gépek és eszközök közötti

megbízható és interoperábilis adatcserét. E rendszer kialakítása során az alapvető cél az volt, hogy egy univerzális és szabványosított megoldást nyújtson a gyártók és felhasználók számára, elősegítve a különböző eszközök és rendszerek egységesítését és könnyű integrációját [6].

Az OPCUA rendszerek alkalmazása széleskörű az ipari automatizálástól az energetikai rendszerekig, és lehetővé teszi az eszközök közötti kommunikációt, az adatok cseréjét és az egységes interfészek kialakítását. Az OPCUA az előzőleg említett UR robotokkal összefüggésben is kiemelkedően fontos lehet, mivel lehetőséget biztosít a robotok és más ipari eszközök közötti kapcsolat kialakítására és szabványosított adatcserére.

Az UR robotok, amelyek rendkívül rugalmasak és könnyen programozhatók az URScript programozási nyelv segítségével, akkor válnak igazán hatékonyá, amikor bekapcsolódnak az OPCUA rendszerekbe. Az OPCUA segítségével a robotok könnyen kommunikálhatnak más eszközökkel és rendszerekkel, valós időben megoszthatják az adatokat, és integrálhatják magukat az ipari automatizációs folyamatokba. Továbbá, egy OPCUA szerver könnyedén képezhet egy interfészt a robot és egy irányító rendszer között, mint például egy hang- vagy gesztusfelismerő rendszer.

Ez a szabványosított kapcsolódási lehetőség jelentős előrelépést jelent az ipari automatizálásban, mivel lehetővé teszi az UR robotok és más eszközök egységesített, interoperábilis működését, javítva a rendszerek átláthatóságát, hatékonyságát és rugalmasságát.

1.4. Flask keretrendszer

A Flask egy könnyen megközelíthető keretrendszer, mely funkcionalitása lehetővé teszi egészen komplex webes alkalmazások fejlesztését. A Flask keretrendszer a WSGI (Web Server Gateway Interface) szabvánnyal működik, ami egy protokoll interfész a Python webes alkalmazások és a webes szerverek közötti kommunikációhoz. A könnyű súlyú jellege lehetővé teszi a rendkívül rugalmas alkalmazások kialakítását, ami kiváló választás olyan projektekhez, ahol fontos a gyors iteráció és a skálázhatóság. A legkiemelkedőbb jellemzője az, hogy teljes kreatív szabadságot ad a fejlesztőknek, ezt a keretrendszer megkötéseinek minimalizálásával éri el. A legtöbb keretrendszer esetében, ha egy probléma megoldására nem a hivatalosan implementált módszert kívánjuk alkalmazni, általában komoly kihívásokkal kell számolnunk. Ezzel ellentétben a Flask

szabad kezet ad: a fejlesztőknek lehetőségük van gyakorlatilag akármilyen (akár nem standardizált) adatbázis használatára [7]. Továbbá, a Flask által használt WebSocket (Flask-SocketIO) kommunikációs protokollnak köszönhetően kialakítható egy megbízható és alacsony látenciájú kapcsolat a Flask szerver és a web kliens között [8].

Összességében, a Flask keretrendszer kritikus jelentőségű szerepet tölt be a fejlesztett rendszer hatékonyságában, biztosítva a rugalmas és hatékony fejlesztést a projekt teljes életciklusában.

1.5. ZeroMQ

A ZeroMQ (ZMQ) egy könnyű, gyors és skálázható üzenetküldő könyvtár, amely különféle alkalmazások közötti aszinkron kommunikációt tesz lehetővé. Az elnevezése a "Zero Message Queue" rövidítése, és hangsúlyozza azt a célt, hogy minimalizálja a kommunikáció végrehajtásához szükséges erőforrásokat, és lehetővé tegye a hatékony és gyors üzenetküldést [9].

A ZMQ kommunikációs könyvtár jelentős jellemzői a dolgozat kontextusában:

- **Könnyű használat:** Az egyszerű és könnyen érthető API-nak köszönhetően a ZMQ lehetővé teszi a fejlesztők számára, hogy gyorsan és hatékonyan integrálják az üzenetküldést az alkalmazásaikba.
- **Aszinkron kommunikáció:** A ZMQ könyvtár lehetővé teszi a résztvevőknek az aszinkron üzenetküldést, ami rugalmasságot és hatékonyságot eredményez, különösen az elosztott rendszerekben.
- **Különféle mintázatok:** A ZMQ támogatja a különféle kommunikációs mintázatokat, beleértve a publish-subscribe (PUB-SUB), a request-reply (REQ-REP) és a push-pull típust.
- **Skálázhatóság:** A ZMQ könnyen skálázható, lehetővé téve az alkalmazásoknak a terhelés elosztását és a nagyobb rendszerek hatékony működését.
- **Nyelvi támogatás:** A ZMQ-t több programozási nyelven is elérhetjük, például C++, Python, Java, C#, és több más nyelven is.

Az ZMQ rendkívül elterjedt és hatékony azokban az alkalmazásokban, ahol gyors és hatékony aszinkron kommunikációra van szükség. Legyen szó mikroszolgáltatásokról, terheléelosztó

rendszerekről, vagy más elosztott alkalmazásokról, az ZMQ kiválóan alkalmazható az üzenetküldésre és a rendszerek összekapcsolására. A távvezérlési rendszerek szempontjából is ideális választásnak bizonyult azáltal, hogy azonnali és megbízható kommunikációt tesz lehetővé a rendszer egyes moduljai között [9].

2. Célkitűzés és hipotézis

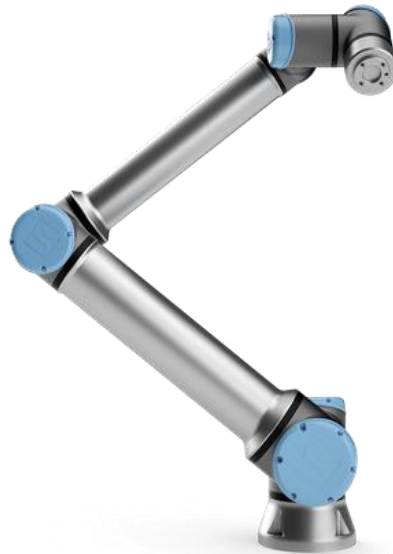
A dolgozat célja egy távvezérlési alkalmazás fejlesztésének bemutatása, mely lehetővé teszi az ipari robotok alacsony késleltetésű irányítását az interneten keresztül. A fejlesztett szoftver egy, az UiT The Arctic University of Norway – Narvik Campus-án korábban elkészített távvezérlési rendszer teljesítményét hivatott javítani, melynek az irányítási késleltetése (control delay) $\sim 1,3$ s (± 200 ms). Emellett egy videómegfigyelő rendszer implementálása is a célok közé sorolható, mely egy grafikus felhasználói felületen keresztül lesz elérhető. Továbbá, a terv részét képezi egy OPCUA szerver integrációja, mely alkalmas úgy a robot paraméterek monitorizálására, mint az irányításra egyaránt. Az OPCUA szerver szerepe egy univerzális, egyszerű interfész létrehozása, mely segítségével további rendszereket csatlakoztathatunk a robothoz. A funkcionalitásra való tekintettel célunk az irányítás átkapcsolásának lehetősége is a lokális OPCUA szerver, illetve egy felhőalapú Flask szerver között. A felsorolt modulokra érvényes kritérium az is, hogy futtatni lehessen úgy Windows, mint Linux alapú operációs rendszereken egyaránt.

A kiválasztott kommunikációs könyvtár (ZMQ) és az általa használt protokoll (TCP/IP) által okozott késleltetések az 1 – 5 ms tartományba esnek [10]. Továbbá, az RTDE keretrendszer 500 Hz-es maximális frissítési frekvenciáját figyelembe véve következtethetünk a robot és a legközelebbi interfésze közötti késleltetésre, amely ~ 2 ms. Az említettekre alapozva kijelenthető, hogy a leírt rendszer elméletileg teljes mértékben kivitelezhető. Ez abban az esetben is érvényes, ha figyelembe vesszük a tervezett rendszer moduláris szerkezetét, mely újabb kommunikációk szükségét okozva további késleltetéseket vezet be a rendszerbe. Az elvárt késleltetés, amikor az irányítás a felhőben futó Flask szerveren keresztül történik, az aktuális internetkapcsolat minőségétől függően akár 500 ms alatt is lehet [11], míg a lokális Flask vagy az OPCUA szerver által közvetített irányítás esetén 100 ms alatti késleltetésre számíthatunk.

3. Metodológia

A fejlesztés során egy UR10e típusú ipari robothoz volt hozzáférésem a norvégiai UiT egyetem Narvikban található karán. Ugyancsak ezen robot segítségével végeztük el a rendszer tesztelését.

Az UR10e robot az Universal Robots e szériájának tagja. A régebbi, CB szériás UR robotokkal szemben, az e szériás UR robotok képesek a végberendezésre ható erő mérésére is. Ez a funkcionalitás új kapukat nyit a fejlesztők és programozók számára, lehetőséget teremtve innovatív alkalmazások kifejlesztésére.



1. ábra UR10e kollaboratív robot [12]

1. Táblázat: UR10e paraméterek [12]

Paraméter	Érték
Szabadságfok (DoF)	6 (Rotációs csuklók)
Max hasznos teher	12,5 kg
Tömeg	33,5 kg
Max kinyúlás	1300 mm
Összteljesítmény	615 W
IP klasszifikáció	IP54 vízállóság
Működési hőmérsékletintervallum	0 – 50 °C

A projekt elkészítése során a kisebb fejlesztési lépések ellenőrzésére nem lett volna alkalmas a fizikai robot használata. A hosszabb indítási idő mellett figyelemmel kell lenni a fejlesztés során fellépő hibák lehetőségére, melyek a robot ütközéséhez és esetleges meghibásodásához vezethet. Ezekből az okokból kifolyólag a projekt fejlesztési fázisának nagyrésztében az URSim szimulációs szoftvert alkalmaztuk.

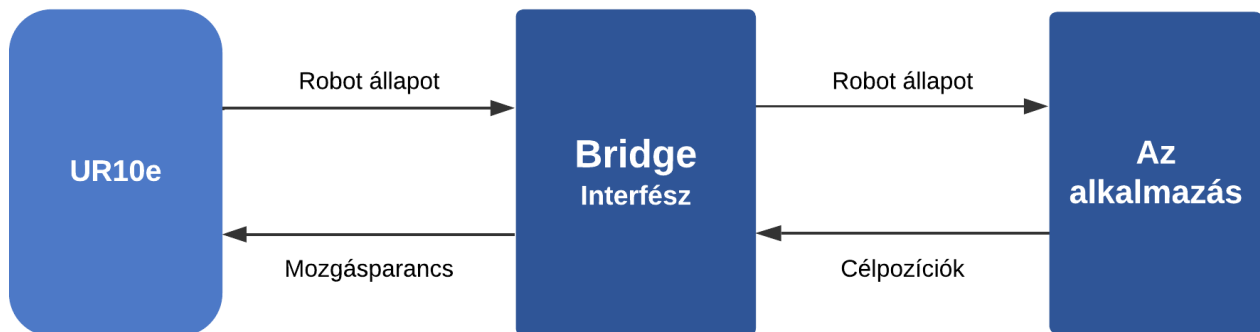
Az URSim egy szimulációs csomag, melyet egy virtuális számítógépen lehet futtatni (VM Ware, Oracle VM stb.). A szimulációs csomag tartalmaz egy 32 bit-es Ubuntu operációs rendszert, illetve a robotvezérlő programját a különböző UR robot modellek számára (UR3e, UR5e stb.). Ez a program tökéletesen megegyezik a fizikai robot vezérlőegységének (teach pendant) a szoftverével úgy vizuálisan, mint funkcionálisan egyaránt. Összességben az URSim használata felgyorsította a fejlesztési folyamatot.

3.1. Rendszertervezés és architektúráis megfontolások

Annak érdekében, hogy a tervezett rendszer el tudja érni a kívánt teljesítményt, aszinkron programozási könyvtárak használata mellett döntöttünk. Ez megengedi a szoftvercsomag különböző moduljai közötti gyors kommunikációt, anélkül, hogy egyes adatcsomagok akadályoznák a többi kommunikációt. A Python programozási nyelvnek beépített része az AsyncIO, mely lehetővé teszi az egyes rutinok és függvények futtatását egyidejűleg.

A távvezérlési rendszer tervezése során a hangsúly az irányítási látencia minimalizálására esett. A kezdetleges elképzelés szerint felállított architektúráis diagramm alapján kapcsolatot kellett teremtenünk a robot, és egy felhőalapú server között. Ezen diagramm iteratív fejlesztése során jutottunk el a végleges szoftverstruktúrához.

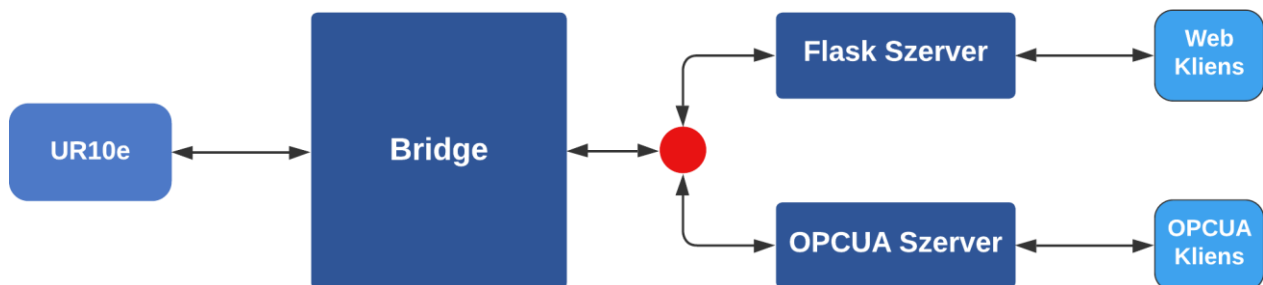
Legelső lépésként szükség volt egy interfész létrehozására, mely a lehető leggyorsabban és egyszerűbben adatot tud továbbítani a robottól a rendszernek, illetve a rendszertől a robotnak. Ezt a szerepet tölti be a Bridge modul.



2. ábra Az UR10e interfész működése és összekötése a fejlesztett alkalmazással

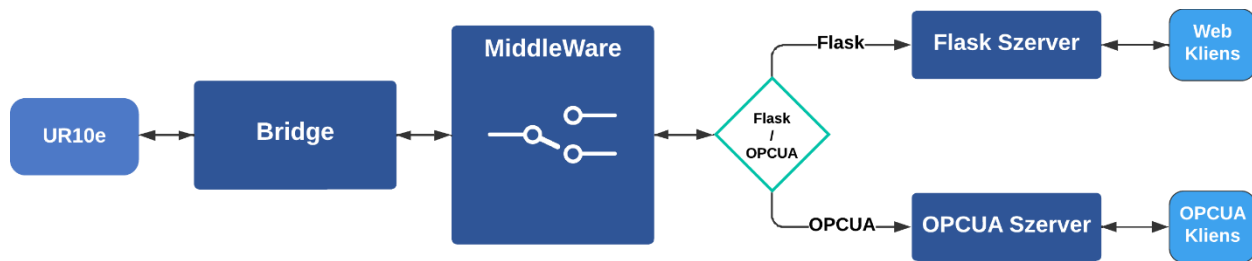
Ugyancsak a Bridge feladatai közé sorolható a célpozíciók (6 db. csuklópozíció) olyan formátumba való átalakítása, mely a robotkontroller számára érthető. Ezen mozgásparancsok az RTDE keretrendszer és könyvtár felhasználásával vannak előállítva.

Egy kezdetleges Flask, illetve egy OPCUA szerver elkészítése után ezeket könnyedén hozzá sikerült kapcsolnunk a Bridge modulhoz. Az ekkor felmerülő probléma az volt, hogy a rendszer nem rendelkezik a szükséges logikával, ami megengedné, hogy mindkét szerver egyszerre működjön. Egy másik felmerülő probléma az adatok szinkronizálása a Bridge és a két szerver között. Továbbá, szükségessé vált egy „kontroll kapcsoló” funkció implementálása is, mely megengedi, hogy a két szerver közül mindig kizárólag az egyik legyen vezető funkcióban. Ezen funkció fontos a rendszer biztonságos működésének érdekében.



3. ábra A problematikus pont a Bridge és a szerverek között

A pirossal jelölt problematikus pont megoldása számos új funkciót igényel. Ezen funkciók beültetése a már meglévő Bridge modul kódjába kétségtelenül túlkomplikálnák azt, ezért egy új modult vezettünk be: a MiddleWare.



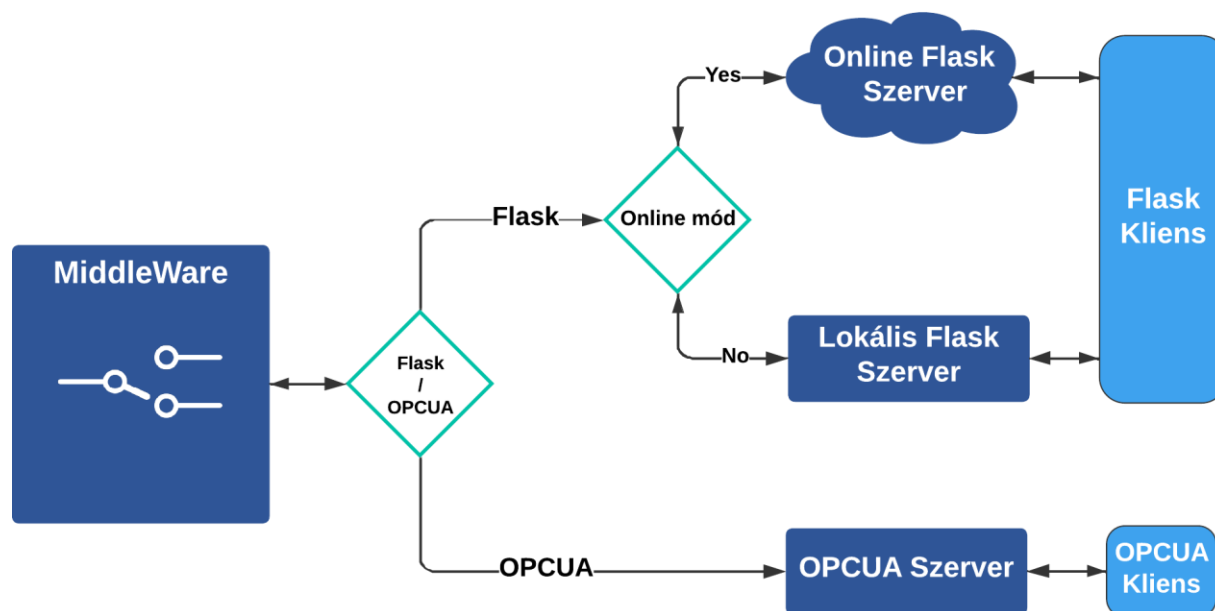
4. ábra A MiddleWare modul elhelyezése a fejlesztett rendszer diagrammjában

Az így kialakult rendszer jelképezi a szoftver struktúrájának alapját. A MiddleWare modulnak köszönhetően van egy megbízható, centrális adatbázisa a rendszernek. A szoftver valós-idejű jellege miatt fontos volt megbizonyosodnunk arról, hogy az adatforgalom minden pillanatban kellően kezelve van.

A MiddleWare modul tölti be az összes többi modult összekötő elem szerepét. A modulba implementált adatbázis teszi lehetővé a rendszer elemei közti szinkronizálást. Előnye az is, hogy megengedi, hogy a Bridge modul, mely interfészt alkot a robot és a fejlesztett rendszer között, a lehető legegyszerűbb maradjon.

A MiddleWare modulba van implementálva az a funkció is, mely megengedi az aktív kontroller kijelölését. Ebből mindig kizárólag egy lehet, a Flask vagy az OPCUA szerver. Ez a funkcionalitás rendkívül fontos biztonsági okokból kifolyólag (egyszerre csak egy helyről érkezhetsz mozgásparancs). Ezen felül még egy fontos feladata a modulnak az, hogy az inaktív kontrollert is mindig szinkronizálja a központi adatbázissal. Ez megengedi, hogy a két modul (Flask és OPCUA) párhuzamosan fusson, illetve azt, hogy az inaktív kontrollert is használhassuk megfigyelés céljából. Az OPCUA szerver kimondottan kézenfekvő megoldást képez a robotparaméterek monitorizálására.

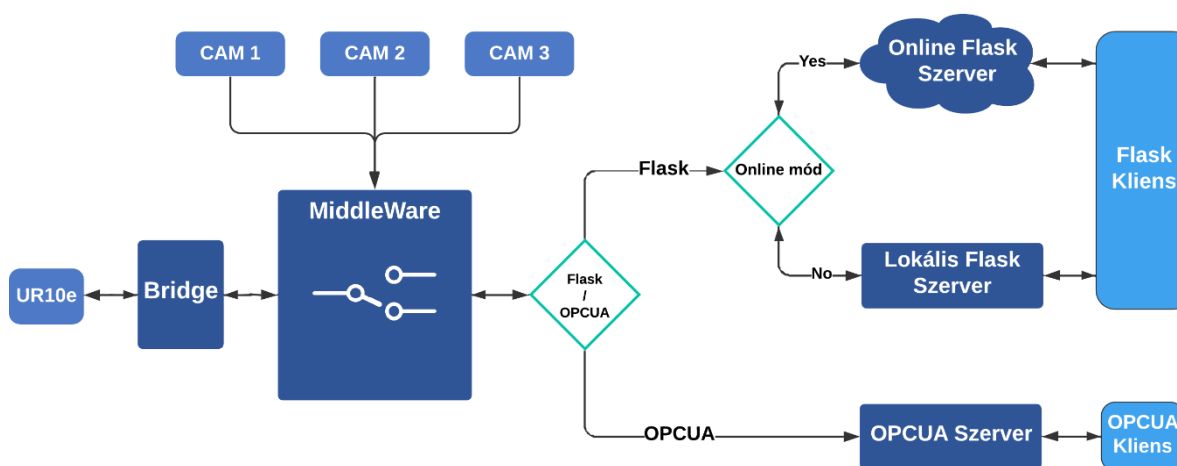
A következő lépés a Flask szerver módosítása volt oly módon, hogy ez futhasson úgy a lokális hálózaton (lokális üzemmód), mint a Microsoft Azure felhőszolgáltatáson egyaránt (online üzemmód). A rendszert ez annyiban érintette, hogy a MiddleWare modul felelősségköre gazdagodott, így lehetővé tette a Flask szerverrel való kapcsolat megalapítását mindkét esetben (online és lokális mód).



5. ábra Lokális és felhőalapú Flask szerver csatolása

A fenti ábrán látható, hogy függetlenül attól, hogy a Flask szerver milyen üzemmódban fut, a felhasználói felület (Flask Kliens) mindkét esetben ugyanaz.

Az ezt követő lépés a videómegfigyelő rendszer implementálása volt. Mivel a robotot megfigyelő kamerák a lokális hálózatra vannak csatlakoztatva, ésszerű a tőlük származó adatfolyamot a MiddleWare modulon keresztül vezetni. Emiatt a MiddleWare feladatköre újból bővült egy annyival, hogy a robotparaméterek mellett a kamerafelvételt is küldi a Flask szervernek.



6. ábra Megfigyelőkamerák implementálása

3.2. Szoftvertervezés és implementáció

Az szoftver fejlesztéséhez a Python programozási nyelvet választottuk. Ennek oka nyelv „high-level” aspektusa, melynek köszönhetően a fejlesztés során kevesebb figyelmet kellett fordítanunk a memóriakezeléshez hasonló részletekre, és annál nagyobb hangsúlyt fektethettünk az alkalmazás funkcionalitásának optimalizálására. Továbbá, a Python programozási nyelv széleskörben támogatott; ezt tükrözi a rendelkezésre álló könyvtárak mennyisége is. A használt fejlesztői környezet: JetBrains – PyCharm IDE.

Az egyes modulok függvényeit, rutinait az AsyncIO beépített Python modul segítségével futtattuk. A könyvtár megengedi az alapvetően blokkoló függvények, mint az adatküldő vagy az üzenetekre váró függvények, aszinkron módon történő futtatását az *await* kifejezés felhasználásával [13]. Az *await* kifejezéssel jelölt programsorok végrehajtása során a program lehetőséget ad más aszinkron függvények futtatására, így ideális egy valós-idejű alkalmazás fejlesztésére. Az aszinkron rutinok megírása után az *asyncio.gather()* függvényt használtuk ezek futtatására. Ez megengedi, hogy a rutinokat (az *asyncio.gather()* argumentumait) egyszerre futtassuk.

A távvezérlési alkalmazás leírása az objektumorientált programozási (OOP) stíluson keresztül volt megvalósítva, mivel így lehetővé vált a moduláris, könnyen karbantartható és kiterjeszthető kód írását. Az OOP segítségével osztályokat és objektumokat alkalmaztunk, illetve a kommunikációért felelős kódrészleteket metódusokba tudtuk sorolni, ezzel tovább rendszerezve a programok struktúráját. Ennek eredményeként a kód átláthatóbbá és érthetőbbé vált, mely megkönnyítette az új funkciók fejlesztését. Továbbá, az OOP megközelítés előnyös lesz a rendszer továbbfejlesztésének egyes lépéseinél is.

3.2.1. Bridge modul

Mint az elsődleges interfésze a robotnak, a Bridge modul elsősorban az RTDE keretrendszert és az ez alapján fejlesztett RTDE könyvtárat (ur-rtde) alkalmazta. Ezek biztosították a megfelelő kommunikációt a robottal.

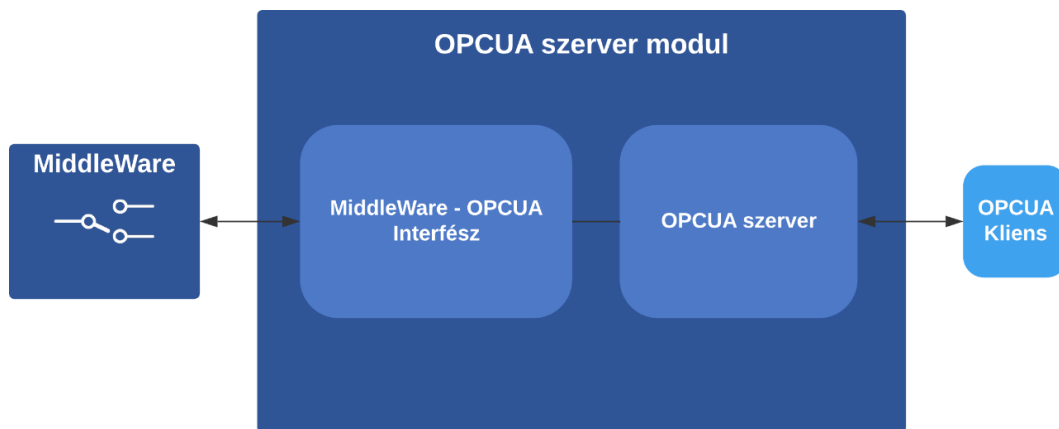
Az RTDE könyvtár több részre bontható:

- RTDE Receive Interface – adatok lekérdezése a robotkontrollertől;
- RTDE Control Interface – mozgatóparancsok küldése a robotkontrollernek;
- RTDE IO Interface – Input/Output műveletek a robotkontrolleren keresztül;
- RTDE Dashboard Client API – üzenetek küldése a robotirányító készülékre (teach pendant).

A Bridge modul, a ZMQ és RTDE kapcsolatok megalapítása után inicializálja a modul aszinkron rutinait. Ezek felelősek az adatok fogadásáért és küldéséért, a célpozíciók alapján a mozgásparancsok megfogalmazásáért, illetve a robot státuszának periodikus lekérdezéséért. A leggyakrabban előforduló státuszok, mint például a *PROTECTIVE STOP* vagy *EMERGENCY STOP* vannak lekezelve a Python programban. Az RTDE könyvtár lehetőséget ad úgy a jelenlegi státusszám lekérdezésére, mint egyes státuszok törlésére is. Ez megengedi, hogy egy biztonsági leállítási státuszt a Bridge modul konzoljából töröljünk és folytassuk a robot használatát. A lekezelt státuszok listája könnyen bővíthető.

3.2.2. OPCUA szerver

Az OPCUA modul legelső feladata egy OPCUA szerver létrehozása. Ez az *opcua* könyvtár segítségével történik. Továbbá, népesíteni szükséges a szervert az összes kívánt adattal, melyek esetünkben a robotparaméterek és a robot irányítását szolgáló jelek. A modul egy interfészt is képez a MiddleWare és az OPCUA szerver között, hiszen az adatokat át kell alakítani a két modul között.

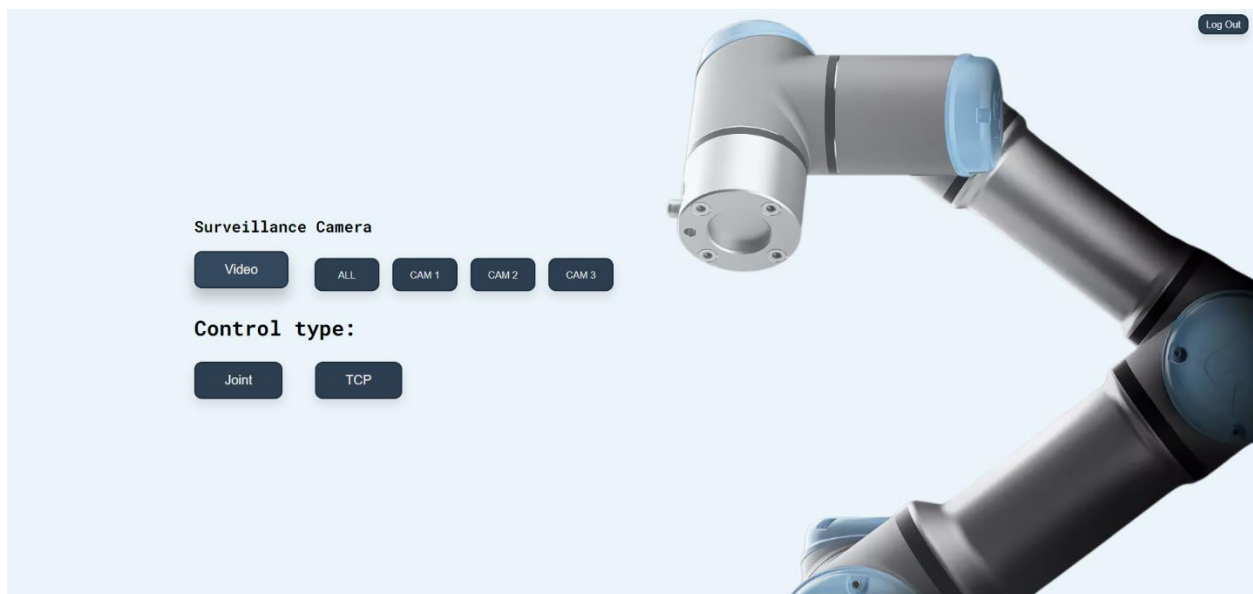


7. ábra OPCUA szerver modul belső szerkezete, feltüntetve az interfészt

Az aszinkron funkciók két osztályba vannak rendezve: *OpcuaValueHandler* és *MiddleWareHandler*. Ezen osztályok tartalmazzák az összes szükséges metódust, melyekkel elvégezhetik az összes adatmozgatási feladatot. Ezek közé tartozik az adatok fogadása a MiddleWare modultól, viszont abban az esetben, ha az OPCUA az aktív kontroller, képes célpozíciók és különböző kontroll jelek küldésére is.

3.2.3. Flask alkalmazás

A Flask alkalmazás felelős a robot mozgására alkalmas web felhasználói felület, illetve egy videófelügyeleti oldal biztosítása. Egy bejelentkezési (login) folyamatot követően, hozzáférést kapunk a főoldalhoz.



8. ábra A Flask felhasználói felület – főoldal (képernyőmentés)

3.2.3.1. Flask Szerver (back-end)

A szerver ellátja a kommunikációt a MiddleWare modullal, illetve a webes felhasználói felülettel. A MiddleWare-el való kommunikációhoz a ZMQ keretrendszert, a felhasználói felülettel való kommunikációhoz pedig SocketIO keretrendszert alkalmaz.

A szerver belső szerkezetében megjelenik az egyesített adatstruktúra, melyet az összes többi modul oszt. Ez az adatbázis a MiddleWare-től kapott csomagok alapján frissül.

A felhasználói tapasztalat javítása érdekében a szerver kódja tartalmazza a szükséges logikát, mely egy konfigurálható változó alapján eldönti, hogy lokálisan vagy a felhőben fut, és a döntés alapján inicializálja a ZMQ kapcsolatokat. Ennek további előnye az, hogy a felhőre való költöztetés során a kódot egyáltalán nem volt szükséges megváltoztatni.

A szerver feladatai közé tartozik a különböző utasítások kezelése is, melyek a felhasználói felületről érkeznek. Ide sorolható a célpozíció továbbítása, robot mozgatása és megállítása, a *control_mode* változó megváltoztatása, mellyel átadhatjuk az irányítást az OPCUA szervernek. A Flask platformba integráltunk egy kezdetleges robotprogram szerkesztő funkciót is, mellyel a felhasználó egy célpozíció listát tud alkotni. Lehetőségünk van hozzáadni és törölni célpozíciókat, melyek mentén a robotot végig tudjuk mozgatni. A listában a mozgás sebessége, illetve típusa is meghatározható (világkoordinátarendszerben vagy csuklótérben lineáris mozgás)

Ugyancsak a Flask szerver feladata a MiddleWare modultól érkező videó stream fogadása és továbbküldése a felhasználói felületre.

3.2.3.2. Felhasználói felület (front-end)

A felhasználói felület megalkotása HTML, illetve CSS (Cascading Style Sheets) segítségével, a funkcionalitásának implementálása pedig JavaScript nyelven történt.

A felület, a betöltés pillanatában üzenetet küld a szervernek, melyre válaszként egy inicializáló csomagot kap. Ez a csomag tartalmazza az összes szükséges aktuális információt ahhoz, hogy a weboldal betöltésekor megjelenített adatok szinkronizálva legyenek a MiddleWare adatbázisával, rajta keresztül pedig a robottal.

A felhasználói felület biztosít egy kezdetleges irányítási platformot a felhasználó számára. Lehetőséget ad a mozgástípus kiválasztására:

- TCP Control – Világkoordinátarendszerben lineáris mozgásparancsot küld a robotnak, a célpozíciót X, Y és Z, illetve Roll, Pitch és Yaw szögekben lehet megadni
- Joint Control – Csuklótérben lineáris mozgásparancsot generál, a célpozíciót a 6 db. rotációs csuklók pozíciójaként lehet megadni

Észrevehető a csúszkák jobb, illetve bal oldalán megjelenő számértékek. A bal oldal a robottól érkezik és az aktuális pozíciókat mutatja. A jobb oldalon a csúszkák pozícióját olvashatjuk le, melyeket a felhasználó állíthat be. Miután a felhasználó beállította a kívánt célpozíciót a csúszkák segítségével, a zöld szívű *MOVE* gombra kattintva tudja mozgatni a robotot a kijelölt helyre. Ezt követően, a csúszkákon egy világosszürke gömb jelképezi a robot aktuális állapotát, mely a mozgás során közeledni fog a beállított értékekhez. Az aktuális értékek 0,1 fokos vagy mm-es felbontással jelennek meg, a mozgástípus függvényében. A célpozíciók beállítása, gyakorlati okokból kifolyólag, 1 fokos / mm-es felbontással adható meg.

Egy lenyitható panel (I/O Control) segítségével a robot digitális kimeneteit is szükség szerint állíthatjuk. A panel kijelzi a jelenlegi kimenetek állapotát is megfigyelés érdekében.

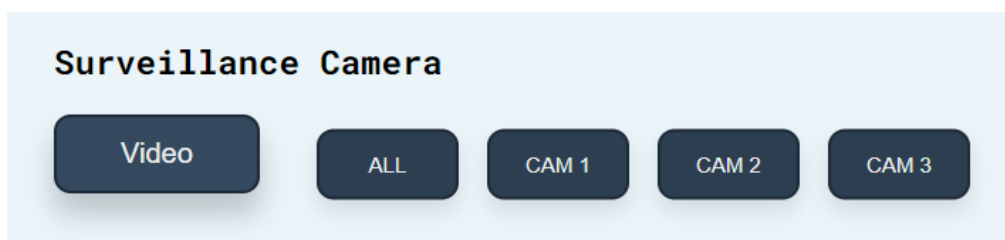
The screenshot displays the 'Joint Control' interface of a Flask application. At the top, there are buttons for 'TCP Control', 'Switch Control to OPCUA', and 'Home Page'. The main section, titled 'Joint Control' with a subtitle 'degrees', contains eight sliders for joint positions and two for speed and acceleration. The sliders are labeled: Base, Shoulder, Elbow, Wrist 1, Wrist 2, Wrist 3, Speed, and Acceleration. Each slider has a current value on the left and a target value on the right. Below the sliders, there is a timestamp '21:20:37' and a button 'I/O Control'. This button opens a panel with three sections: 'Standard', 'Configurable', and 'Tool'. Each section has a row of 8 bits, labeled 'I/O Bit ID' from 0 to 7. The 'Input Bits' row shows all bits as off (light orange). The 'Output Bits' row shows bits 1, 2, 5, 6, and 7 as on (dark blue). The 'Target Output Bits' row shows bits 1, 2, 5, 6, and 7 as on (dark blue). Below this panel is a 'Send Output Bits' button. At the bottom of the interface are buttons for 'Randomize Joint Values', 'Send Home', 'MOVE' (green), and 'STOP' (red).

9. ábra A Flask felhasználói felület – Joint Control oldal (képernyőmentés)

A SocketIO keretrendszer `socket.on("üzenet", {végrehajtandó rutin});` funkciójának felhasználásával voltak kezelve a felhasználói felületen megjelenő adatok frissítései. Ezeket a Flask szerver back-end oldaláról vannak kezdeményezve minden adatcsomag érkezésekor.

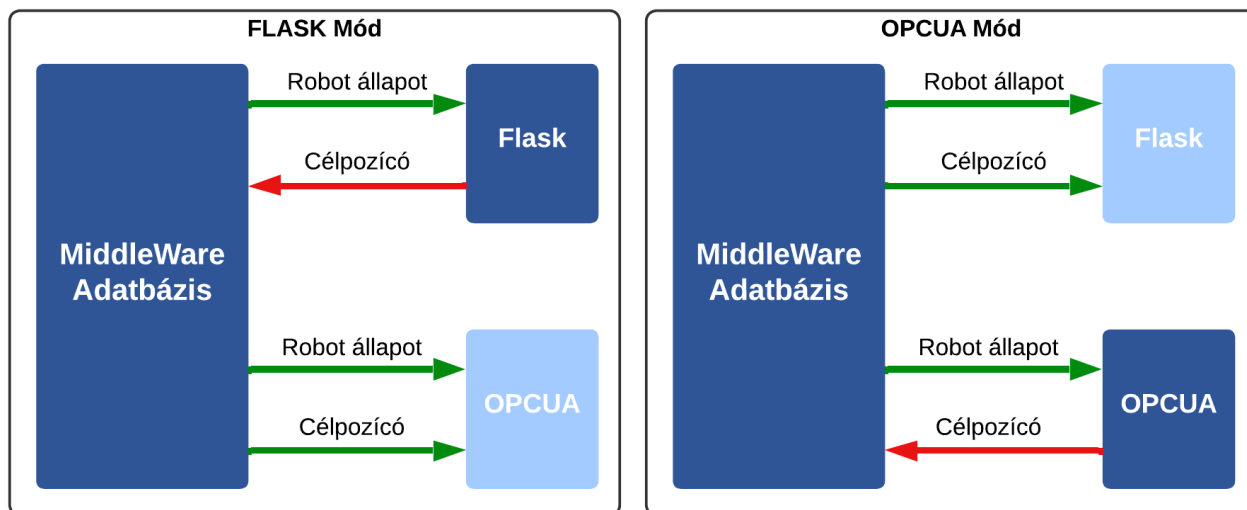
3.2.3.3. Videómegfigyelő rendszer

A MiddleWare modul részéről érkező video stream-ek megjelenítése a *Video Surveillance* oldalon történik. A felhasználónak lehetősége van a 3 video stream megtekintésére külön-külön vagy egyben. Ezekre külön weboldalak adnak lehetőséget.



10. ábra A Flask felhasználói felület – videófelügyeleti oldalak (képernyőmentés)

3.2.4. MiddleWare



11. ábra A rendszer két különböző irányítási módszere

A fenti ábrán látható a MiddleWare modulba implementált logika, amely alapján kezeli az adatforgalmat az aktív kontroller függvényében.

A MiddleWare modul biztosítja azt, hogy a robotot egyszerre kizárólag egy szerver irányíthat. Egy *control_mode* változó bevezetésével volt implementálva a funkció, mely értékének függvényében minden időpillanatban szigorúan egy aktív és egy inaktív kontroller van. Minden célpozíció érkezésének alkalmával a MiddleWare modul leellenőrzi a *control_mode* változó értékét, és csak abban az esetben küldi tovább az adatokat a robot fele (a Bridge modulnak), ha az aktív kontrollertől érkezett az adat. A *control_mode* változó megváltoztatására mindkét szervernek (Flask és OPCUA) van lehetősége, viszont biztonsági okokból kifolyólag ez megváltoztatható, oly módon, hogy az OPCUA szerver élvezzen prioritást a Flask szerverrel szemben. Ez abból a megfontolásból ered, hogy a lokális hálózaton futó OPCUA szerverhez nincs hozzáférés az interneten keresztül, így biztonságosabb.

Továbbá, a MiddleWare modul felelősségei közé tartozik az is, hogy biztosítsa a robotparaméterek szinkronizálását az összes többi modul között. Ez, többek között azt is jelenti, hogy az aktív kontrollertől érkező célpozíció esetén ezt az adatot nem csak a robothoz, hanem az inaktív kontrollerhez is el kell juttatnia. Ez abból a szempontból is fontos, hogy hasznos és megbízható maradjon az inaktív kontroller is, mint felügyelő platform. Egy fajta betekintést biztosít abba, hogy pontosan milyen értékek vannak a MiddleWare (és az egész rendszer) adatbázisában.

A megfigyelő kamerák adatfolyama ugyancsak a MiddleWare modulok keresztül van a rendszerhez kötve. A videó kereteket, kódolás után továbbküldi a Flask szervernek, ami dekódolja az adatokat és lehetőséget ad a videók megtekintésére.

3.3. Kommunikáció és integráció

A fizikai robot (vagy az URSim program) és a Bridge modul közötti kommunikáció az RTDE keretrendszer (1.2 fejezet) segítségével történik. Az összes többi modul között (amely kapcsolatban van a MiddleWare-el) ZMQ kommunikációt alkalmaztunk (1.5 fejezet).

A ZMQ kommunikációk típusa Publish-Subscribe (PUB-SUB). Ennek oka az aszinkron jellegű adatcsere biztosítása. A PUB-SUB esetén a küldőnek (Publisher) nem kötelező megvárnia, amíg

választ kap a fogadótól (Subscriber) az üzenet kézbesítéséről, így valódi aszinkron működést biztosít. További előnye az is, hogy egyes elakadt üzenetek esetében nem fog megakadni az egész rendszer, így megbízhatóbbá téve ezt.

A különböző számítógépek közötti kommunikáció, beleértve a ZMQ, illetve az RTDE keretrendszerrel megvalósított kommunikációt is, TCP/IP protokollt alkalmaz. Ennek következménye az adatküldés sztochasztikus természete [14], viszont a lokális kommunikációból származó késleltetés elhanyagolhatónak minősül az interneten keresztül történő kommunikációból adódó késleltetésekkel szemben.

Az OOP megközelítés miatt minden modul el van látva egy inicializáló osztállyal. Ezek az osztályok tartalmazzák azon metódusokat, melyek létrehozzák az ZMQ (Bridge modul esetében ZMQ és RTDE) kapcsolatokat.

3.3.1. Flask szerver

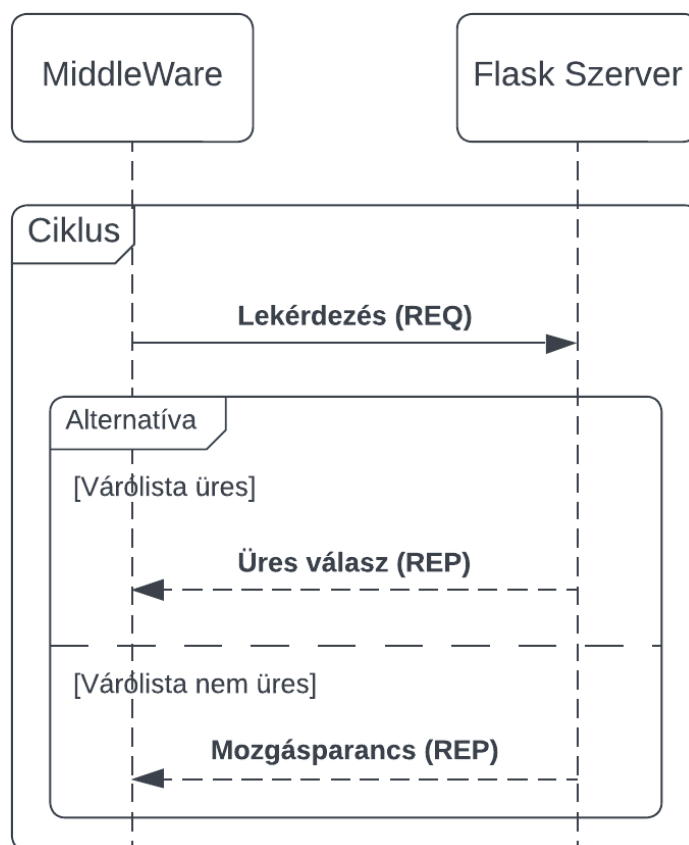
A Flask szerver és a front-end közötti kommunikációhoz a Flask-SocketIO könyvtárat alkalmazzuk, amely websocket alapú adatátvitelt tesz lehetővé. Ez az eszköz lehetővé teszi valós idejű kommunikációt a szerver és a böngésző között, így megengedve a dinamikus, interaktív felhasználói felület kialakítását. A Flask-SocketIO használatával a szerver képes aszinkron üzenetek fogadására és küldésére, ami hatékonyabbá teszi a kommunikációt a klienssel. Ezáltal lehetővé válik az alkalmazás részletes vezérlése és állapotának valós idejű frissítése. A websocket alapú megoldás gyors és megbízható adatátvitelt biztosít, ami nélkülözhetetlen az olyan alkalmazásokban, ahol az időzítés és az interakció sebessége kritikus jelentőségű.

A kameráktól érkező adatokat a MiddleWare modul dolgozza fel. Érkezésük után *.jpg* formátumba tömöríti a videó képkockáit, majd byte formátumba alakítja, hogy továbbküldhesse a Flask szervernek. Az adattovábbítás ZMQ PUB-SUB formában történik. A küldő a MiddleWare modul, a fogadó pedig a Flask szerver. A dinamikus IP címből adódó probléma kiküszöbölése érdekében a Flask szerver (Subscriber) oldalán IP címtől függetlenül várjuk az adatokat, bizonyos meghatározott portokon.

3.3.1.1. Dinamikus IP cím kezelése a felhőben

Amikor Flask szervert átvittük a Microsoft Azure felhő platformjára, felmerült egy kommunikációs probléma. Míg a PUB-SUB megoldás előnyös, hiszen csak a Publisher IP címét szükséges ismernünk, ugyanakkor csak egy irányú kommunikációt valósít meg. A kétirányú kommunikáció kialakítása érdekében két csatornát kell létrehoznunk, mindkét végpont IP címének ismeretében. Azonban a legtöbb hálózat publikus IP címe dinamikus, DHCP (Dynamic Host Configuration Protocol) protokoll alkalmazása miatt, ami gondot okoz a kétirányú kommunikációban. Ennek következményeként a lokális hálózatról az ismert IP című felhőalapú számítógépünkre tudtunk adatokat küldeni, viszont a fordított irányban, a felhőből a dinamikus című lokális hálózatra már nem.

A megoldás a Request-Reply (REQ-REP) típusú adatcserében rejlik, mely a ZMQ keretrendszeren belül működik. Ez a kommunikációs forma lehetővé teszi válaszok küldését beérkező üzenetekre anélkül, hogy a címet meg kelljen adni. A REQ-REP típusú kommunikáció segítségével a fejlesztők nem látnak bele abba, hogy az üzenet honnan érkezett, azonban a keretrendszer képes választ küldeni. Ennek kihasználásával felállítottunk egy lekérdezési mechanizmust (polling), mely rendszeresen lekérdezéseket küld a felhőben futó Flask szervernek.



12. ábra Lekérdezés mechanizmusa

A szerver oldalon egy várólistára kerülnek az összes elküldendő parancsok (mozgásparancsok, I/O parancsok), majd a következő lekérdezés érkezésekor ezeket válaszként elküldik. Ezzel a megoldással sikerült megvalósítani az adatküldést a felhő és a lokális hálózat között anélkül, hogy ismernénk a helyi hálózat publikus IP címét.

3.3.2. A robot digitális ikerpárja

Az OPCUA szerver segítségével létrehoztuk a robot digitális ikerpárját is. Ez a Visual Components szimulációs program környezetében történt, hiszen elérhető ezen program részére egy kiegészítő modul (plugin), mely megengedi, hogy OPCUA szerverekre csatlakozzunk. A program könyvtárában több ipari robot található, melyek közül választhatunk, például az UR10e típust vagy bármely más 6 rotációs csuklóval rendelkező robotot. Ezt követően egyszerűen illesztenünk kell a virtuális robot egyes csuklóparamétereit az OPCUA szerveren található paraméterekhez. Ennek eredményeként a digitális robot pontosan tükrözni fogja a fizikai társának mozgásait.

Structure	Simulation variable	...	Simulati...	Prepared...	Latest va...	..	Server variable
- Server - Simulation to server - Server to simulation - J1 - J2 - J3 - J4 - J5 - J6	- UR10e-Series.J1.J1 - UR10e-Series.J2.J2 - UR10e-Series.J3.J3 - UR10e-Series.J4.J4 - UR10e-Series.J5.J5 - UR10e-Series.J6.J6	- R 1.2 - R 1.2 - R 1.2 - R 1.2 - R 1.2 - R 1.2	-179.97999... -60.770000... -110.05 -99.180000... 90 180.02000...				- Current Base - Current Shoulder - Current Elbow - Current Wrist 1 - Current Wrist 2 - Current Wrist 3

13. ábra OPCUA értékek Visual Components környezetben (képernyőmentés, részlet)

3.4. Teljesítményoptimalizáció

A fejlesztett rendszer, az alacsony késleltetés érdekében, aszinkron működésű. Emellett további lépéseket tettünk annak érdekében, hogy biztosítsuk a gyors és hatékony működést.

Elsősorban, az RTDE keretrendszer teljesítményének teljes fokú kihasználását tűztük ki célul. A robotkontroller és a Python modulok között a maximális, 500 Hz-es frissítési frekvenciát alkalmaztuk. A MiddleWare modul is ezzel a frekvenciával kommunikál a Bridge moduldal, így a robot és a rendszer központi adatbázisa között a lehető legkisebb késleltetést biztosítottuk, mivel a rendszer összes többi moduljának működése ezen adatbázisra támaszkodik.

A kamera rendszer teljesítményének optimalizálása érdekében több lépést is megtettünk. Először is, a videostreamek kezelésére a *threading* könyvtárat alkalmaztuk, hogy párhuzamosan tudjuk kezelni a három kamera adatfolyamát. Ez lehetővé teszi a hatékonyabb és gyorsabb feldolgozást a rendszerben.

A második lépés a képminőség optimalizálása volt. Mivel a felhasználási célhoz az 1080p felbontás feleslegesnek bizonyult, a videostreameket 720p felbontásra állítottuk. Ez a felbontás még mindig több, mint elegendő a rendszer által nyújtott szolgáltatásokhoz, hiszen kitűnően megfigyelhető a robot, miközben csökkenti a feldolgozási- és a sávszélesség-igényt a MiddleWare és a felhőben futó Flask szerver között.

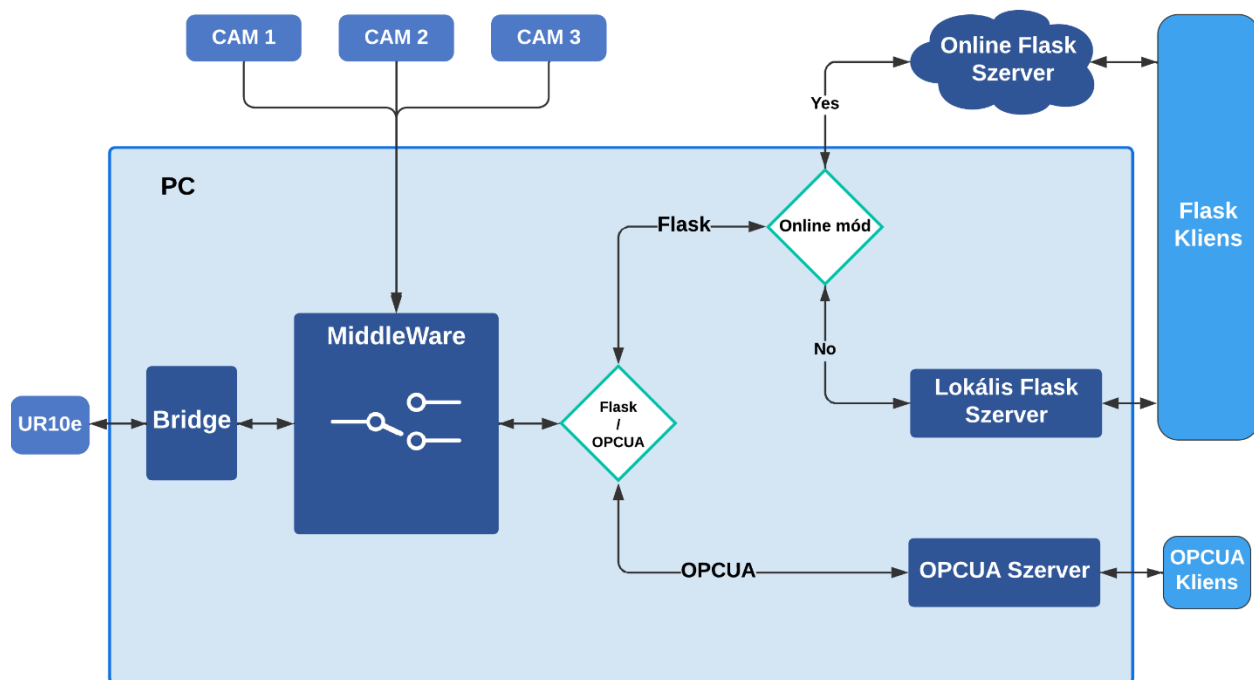
Végül, hogy tovább optimalizáljuk a rendszert, a videóstream csak akkor működik, ha valaki aktívan megnyitja és használja a videófelügyelő oldalt. Ezt egy 10 másodperces időkorláttal értük el a MiddleWare oldalon. Ha senki sem nézi a videókat, akkor a MiddleWare automatikusan megszakítja a videostream továbbküldését a felhőbe, ezáltal tovább csökkentve a felesleges adatforgalmat és erőforrás-felhasználást.

A Flask szerver frissítési frekvenciájának optimalizálásával is hozzájárultunk a rendszer hatékonyságához. Mivel nem volt szükség a felhasználófelület frissítésére 500 Hz-es frekvenciával, empirikus módon meghatároztuk, hogy akár a 20 Hz-es frekvencia is tökéletesen megfelel. A végső döntés 50 Hz-es frissítési frekvenciára esett, hiszen ez távolról sem közelíti meg a kommunikációs sáv szélesség határait, illetve az ennél magasabb frissítési frekvencia csak feleslegesen növelte volna a rendszer hálózati igényeit.

Ezen túlmenően a front-end frissítési periódusa dinamikusan változik. Amikor a robot nem mozog, egy alacsonyabb, stand-by frissítési frekvenciát alkalmazunk, amely tovább csökkenti a felesleges adatforgalmat. Ezt egy *is_moving* paraméter bevezetésével értük el, mely az egész rendszernek jelzi, hogy a robot éppen mozgásban van, igényelve a magas frekvenciájú frissítést, vagy sem. Ezen optimalizációs lépések együttesen javítják a rendszer hatékonyságát és teljesítményét, miközben minimalizálják a felesleges hálózati terhelést és erőforrás-felhasználást.

Fontos megemlíteni, hogy a Flask szerver frissítési frekvenciájának optimalizálása során a „zéró kompromisszum” megközelítést alkalmaztuk. Ezen változtatások nem jártak vizuális különbséggel a felhasználói felületen. Annak ellenére, hogy a frissítési frekvenciát csökkentettük és a dinamikus frissítési periódust bevezettük, a felhasználók számára ugyanazt a felhasználói élményt biztosítottuk, miközben a rendszer belső optimalizációja megtörtént.

Ha a rendszert a lehető legalacsonyabb késleltetéssel szeretnénk működtetni, lehetőségünk van a Flask szerveren kívül az összes többi modult egy számítógépen futtatni (az alábbi ábrán a „PC” kereten belüli modulok). Ezáltal kiiktatunk számos hálózaton keresztül történő adatátvitelt, mely kis mértékben tovább csökkenti a késleltetést.



14. ábra Az egy számítógépen (PC) futtatható modulok

A program futtatásához szükséges konfigurációs információk és a programcsomag futtatásának egyszerűbb kezelése érdekében létrehoztunk egy központi konfigurációs fájlt. Ez egy Python állomány, mely tartalmazza az összes szükséges változót, melyet a program fejlesztése és futtatása során változtatni kell:

- `online_mode` változó a Flask szerver számára
- Robot és különböző modulok IP címe
- Szerver és ZMQ végpontok portjai
- Frissítési frekvenciák a Bridge modul, illetve a Flask és az OPCUA szervernek
- RTSP linkek a megfigyelőkameráknak
- Robotcsuklók határértékei
- `Home_position` csuklópozíciók meghatározása a Web felületnek

```

1  # ----- ONLINE -----
2
3  ONLINE_MODE = True
4  FLASK_DOMAIN = 'http://www.uitremote.com'
5
6  FLASK_USERNAME = 'user'
7  FLASK_PASSWORD = 'admin'
8
9  # ----- IP Addresses -----
10
11 IP_UR10e = "192.168.1.3"      # URSim on Snak's Laptop
12
13 IP_BRIDGE = "192.168.1.225"
14 IP_MWARE = "192.168.1.225"
15 IP_OPCUA = "192.168.1.225"
16
17 IP_FLASK_LOCAL = "0.0.0.0"    # join on 127.0.0.1
18 IP_FLASK_CLOUD = "51.120.242.4"
19
20 # ----- Ports -----
21
22 # Server Ports
23
24 PORT_FLASK_SERVER = 80        # port of Flask Server
25 PORT_OPCUA = 5001             # port of OPCUA Server
26
27 # ZMQ Ports
28
29 PORT_B_MW = 5553              # bridge to MW
30 PORT_MW_B = 5554              # MW to bridge
31
32 PORT_MW_OP = 5559             # MW to OPCUA
33 PORT_OP_MW = 5560             # OPCUA to MW
34
35 PORT_FLASK_POLL = 8091        # MW to Flask - and back
36 PORT_FLASK_UPDATE = 8092     # MW to Flask - update data
37
38 PORT_FLASK_VIDEO_1 = 5555     # live camera feed 1 to Flask
39 PORT_FLASK_VIDEO_2 = 5556     # live camera feed 2 to Flask
40 PORT_FLASK_VIDEO_3 = 5557     # live camera feed 3 to Flask
41
42 # ----- Frequencies -----
43
44 RTDE_FREQ = 500               # [1 - 500]
45 FLASK_FREQ = 50               # [1 - rtde_frequency]
46 OPCUA_FREQ = 50               # [1 - rtde_frequency]

```

3.5. Adatgyűjtés és értékelés

Az alkalmazás elkészítése után sikerült kezdetleges méréseket is végeznünk. Annak érdekében, hogy az alkalmazás teljesítményét földrajzi elhelyezkedésünktől függetlenül tudjuk felmérni, különböző országokból mértük meg a visszacsatolási késleltetést (feedback delay). A méréseket a Flask szerver weboldalának JavaScript kódjába implementáltuk, mely segítségével mérhettük a parancs elküldése és a robot által generált válasz visszaérkezése közötti időt.

Fontos megemlíteni, hogy a mért késleltetés kétirányú (round-trip) kommunikációra érvényes, ebből adódóan az irányítási késleltetés (a parancs elküldése és a robot megmozdulása közti idő) várhatóan ezen értékeknek nagyjából fele. Minden tesztelési lokáció esetében 25 különálló mérést végeztünk, majd ezeket legkisebb és legnagyobb értékét tüntettük fel, annak érdekében, hogy a szórást is meg tudjuk becsülni.

2. Táblázat: A fejlesztett rendszer visszacsatolási késleltetése

Lokáció	Késleltetés [ms]	Kapcsolat típusa
UiT Narvik, NO	76 – 91	Ethernet
Narvik, NO	75 – 140	Wi-Fi (Eduroam)
Észak Karolina, USA	166 – 194	Wi-Fi
Marosvásárhely, RO	171 – 208	Ethernet
Ontario, CA	291 – 337	Wi-Fi

A fenti táblázatban látható, hogy a legnagyobb késleltetés 337 ms, mely a Kanadában történő tesztelés legnagyobb késleltetési eredménye. Az elméleti kutatás során felállított hipotézisünk igazolódott, mely szerint 500 ms alatti késleltetést érhetünk el egy hasonló rendszerrel. Gyakorlati méréseink igazolták, hogy a fejlesztett rendszer az elvártnál kisebb késleltetést biztosít.

A Flask szerver lokális hálózaton való futtatása esetében a visszacsatolási késleltetés átlagosan 22 ms volt. Ebből arra következtethetünk, hogy a késleltetés jelentős részéért az interneten keresztül történő kommunikáció felelős. Az OPCUA szerverrel történő irányítás esetében a késleltetés

mérésére nem adódott lehetőségünk, viszont az előző megfigyelésre alapozva ez az érték is várhatóan kisebb, mint 100 ms.

4. Konklúzió

Összességben elmondhatjuk, hogy sikeresen fejlesztettünk egy távvezérlési rendszert UR10e típusú ipari robotok számára. A felhőalapú rendszer hatékony fejlesztése után lehetőségem volt elkészíteni úgy egy grafikus felhasználói felületet a Flask szerver weboldalának, mint egy videómegfigyelő rendszert is. A rendszer teljesen ellát minden tervezett funkcionalitást, többek között a vezérlés átkapcsolását is az említett Flask szerver, illetve egy lokális OPCUA szerver között. Az UiT - Narvik Egyetem közreműködésének köszönhetően a teljes rendszer alapos tesztelésére is lehetőségünk volt.

5. Továbbifejlesztési lehetőségek

Az ezen fejezetben tárgyalt fejlesztési irányok megvizsgálása segíthet a rendszer funkcionalitásának további optimalizálásában és a felhasználói élmény fokozásában, ami különösen fontos egy távvezérlési rendszer esetén, ahol a precíz irányítás elengedhetetlen. A rendszer modularitásának köszönhetően egy megbízható és terjeszthető alapot biztosít a szoftvercsomag továbbfejlesztésének, ami elősegíti az új funkciók hozzáadását és a rendszer skálázhatóságát a jövőben.

5.1. Webbiztonság

A jelenlegi rendszer funkcionalitását továbbfejlesztendő terület a webbiztonság. Fontos, hogy a rendszer védelmet nyújtson a potenciális támadások ellen, különös tekintettel a távoli elérési lehetőségekre. Míg sikeresen implementáltunk egy bejelentkezési funkciót a Flask szerver weboldalára, ez csupán egy kezdetleges megoldást jelent. A továbbiakban az SSL/TLS titkosítás implementálása kulcsfontosságú szerepet fog játszani a rendszer biztonságos működésében.

5.2. Erő-visszacsatolás (force-feedback)

A rendszer funkcionalitásának gazdagítása érdekében tervben van egy erő-visszacsatolási rendszer integrálása. Ez lehetővé tenné a felhasználó számára, hogy valós idejű visszajelzést kapjon a robot

által végrehajtott műveletek során fellépő erők méretéről és irányításáról. Az erő-visszacsatolás hozzáadása növelheti a felhasználó által végzett távvezérlés minőségét és javíthatja a precizitást, mindezt amellet, hogy teljesen új lehetőségeket hozna létre a robotok alkalmazási területeire való tekintettel.

5.3. Különböző vezérlési modulok

A rendszer funkcionalitásának bővítése érdekében érdemes lenne más vezérlési modulok integrálását is mérlegelni. Ezen modulok implementálására alkalmas a fejlesztett OPCUA interfész. A különböző vezérlési modulok közé tartoznak:

- Gesztus vezérlés: a gesztusvezérlés bevezetése lehetővé tenné a felhasználók számára, hogy intuitív mozdulatokkal irányítsák a robotot. Ez növelheti a felhasználói rugalmasságot és kényelmet
- Hangvezérlés: a hangvezérlés integrálása lehetővé tenné a felhasználók számára, hogy hangutasításokkal irányítsák a robotot. Ezáltal még inkább növelhető a vezérlés egyszerűsége

A távvezérlési rendszer modularitásának köszönhetően számos más vezérlési modul implementálása válik lehetségessé.

6. Köszönetnyilvánítás

Ezúttal szeretném kifejezni hálám a dolgozat konzulensinek, dr. Forgó Zoltán és dr. Fehér Áronnak, illetve a norvégiai UiT The Arctic University of Norway – Narvik Campus Mérnöki Tudományok és Technológia Tanszékének, külön köszönettel Bjørn Solvang professzornak és Hao Yu egyetemi docensnek, akik kulcsfontosságú szerepet játszottak a projekt elkészítésében a kutatási körülmények biztosításával. Emellett külön elismerés illeti Beibei Shu posztdoktori kutatót, aki a felügyeletével támogatott a projekt elkészítése során, végigkövetve a fejlesztés lépéseit a „RoNo Mobility for Industry X.0” (22-MOB-0027) EEA Grant keretén belül.

7. Irodalomjegyzék

- [1] F. Sherwani, M. M. Asad, és B. S. K. K. Ibrahim, „Collaborative Robots and Industrial Revolution 4.0 (IR 4.0)”, in *2020 International Conference on Emerging Trends in Smart Technologies (ICETST)*, IEEE, márc. 2020, o. 1–5. doi: 10.1109/ICETST49965.2020.9080724.
- [2] „UR - Connecting internal I/O on the robot’s controller”. Elérés: 2024. március 29. [Online]. Elérhető: <https://www.universal-robots.com/articles/ur/interface-communication/connecting-internal-inputs-and-outputs-io-on-the-robots-controller/>
- [3] „UR - Profinet Guide for e-Series”. Elérés: 2024. március 29. [Online]. Elérhető: <https://www.universal-robots.com/articles/ur/interface-communication/profinet-how-to-guide-e-series/>
- [4] *The URScript Programming Language for e-Series*. 2021. Elérés: 2024. március 11. [Online]. Elérhető: https://s3-eu-west-1.amazonaws.com/ur-support-site/115824/scriptManual_SW5.11.pdf
- [5] „REAL-TIME DATA EXCHANGE (RTDE) GUIDE”, <https://www.universal-robots.com/articles/ur/interface-communication/real-time-data-exchange-rtde-guide/>.
- [6] Wolfgang Mahnke, Stefan-Helmut Leitner, és Matthias Damm, *OPC Unified Architecture*. Spring, 2009.
- [7] Miguel Grinberg, *Flask Web Development*, 2. kiad. O’Reilly Media, 2018.
- [8] Andrew Lombardi, *WebSocket*. O’Reilly Media, 2015.
- [9] Pieter Hintjens, *ZeroMQ: Messaging for Many Applications*, 1. kiad. O’Reilly Media, 2013.
- [10] Y. Yang, L. Zhang, és X. Wang, „Time Delay Performance Analysis of Distributed Communication Platform Based on ZeroMQ”, in *2019 International Conference on Communications, Information System and Computer Engineering (CISCE)*, IEEE, júl. 2019, o. 319–323. doi: 10.1109/CISCE.2019.000078.

- [11] „Azure network round-trip latency statistics”. Elérés: 2024. március 14. [Online]. Elérhető: <https://learn.microsoft.com/en-us/azure/networking/azure-network-latency?tabs=Europe%2CCentralEurope>
- [12] „UR10e Specifications”.
- [13] „asyncio — Asynchronous I/O”. Elérés: 2024. március 18. [Online]. Elérhető: <https://docs.python.org/3/library/asyncio.html>
- [14] E. Altman, K. Avrachenkov, és C. Barakat, „A stochastic model of TCP/IP with stationary random losses”, in *Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, New York, NY, USA: ACM, aug. 2000, o. 231–242. doi: 10.1145/347059.347549.
- [15] „Request/Reply - ZMQ”. [Online]. Elérhető: https://learning-0mq-with-pyzermq.readthedocs.io/en/latest/pyzermq/patterns/client_server.html