



Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Automatizálási és Alkalmazott Informatikai Tanszék

## Alacsony hálózati késleltetésre épülő AR megoldások telefonra és okosszemüvegre

MSc ÖNÁLLÓ LABORATÓRIUM 1-2.

*Készítette*

Márta Boldizsár

*Konzulens*

Dr. Forstner Bertalan

2022. május 26.

## Tartalomjegyzék

<b>1. A feladat leírása</b>	<b>1</b>
<b>2. A használni kívánt technológiák vizsgálata</b>	<b>2</b>
2.1. Robotkar vezérlés . . . . .	2
2.2. Valós idejű vezérlés IP alapon . . . . .	3
2.3. VR/AR megjelenítés . . . . .	4
2.4. Videójel továbbítása a robotkartól a vezérlő kliensbe . . . . .	5
WebRTC videó streamelés . . . . .	6
2.5. A rendszer tervezett felépítése . . . . .	7
<b>3. Megvalósítás</b>	<b>8</b>
3.1. ROS . . . . .	8
3.2. Felhasználói kliens . . . . .	9
VR/AR kliens . . . . .	9
Webes kliens . . . . .	10
3.3. Kommunikáció . . . . .	11
3.4. Valós robotkar vezérlése . . . . .	12
<b>4. Összefoglaló gondolatok</b>	<b>14</b>
<b>5. További tervezettségek</b>	<b>15</b>

## 1. A feladat leírása

Az 5. generációs mobilhálózatok elhozták az alacsony kommunikációs késleltetés korát. Az egyik fontos alkalmazási területe ennek a különböző kiterjesztett valóság alkalmazásokon keresztül tálalt valósidejű információkra építő ipar, szórakoztató vagy egyéb információs alkalmazások.<sup>1</sup>

Rendkívül sok lehetőget tartogat magában egy olyan projekt, amely 5G-re és valamilyen kiterjesztett- vagy virtuális valóság technológiát használ. Látszik, hogy a jövőnek fontos része lesz ez a technológia, így hasznos, ha ismereteket szeretnénk róluk, ezért szerettem volna ezzel foglalkozni.

A feladatom egy olyan szoftverrendszer elkészítése lett, melynek segítségével távolról, 5G hálózaton keresztül lehet vezélni egy robotkart VR környezetből, miközben kameráképen látja is a felhasználó, hogy mi történik a robot körül. Ez a rendszer számos konkrét felhasználásnak adhat alapot, többek között:

- egy sűrűn berendezett gyáregységben lehetőséget nyújthat a robotkarok egyszerű konfigurálására, esetleg egyszerűbb hibák elhárítására anélkül, hogy közel kéne menjünk hozzájuk
- ha egy eszköz meghibásodik, akkor akár az azt telepítő cég mérnöke távolról, a gyár területén kívülről is tudja vizsgálni
- egy robotkar beállítása után lehetőséget adhat a robotkarral együtt végzett munka egyszerű kivitelezésére is, például félautomata selejtdetekció során a robotkar csak felemeli az adott vizsgálni kívánt terméket, majd a kezelő utasítására vagy tovább engedi a gyártási folyamatot, vagy selejtként kiszelektálja

A felhasználói felület esetében két használati mód, és ezzel együtt felület is van. Az egyik a konfigurációt végző személynek hasznos, itt be tudja állítani a robotkar általános pozícióit, esetleges határait, műveleteit. A másik mód minden nap használatra alkalmas, ebben képes a kezelő felügyelni és utasítani a robotot működés közben.

A konkrét felhasználás, amire a feladat megoldást kíván adni, az futószalagon érkező termékek félautomata selejtdetekviója. Az érkező termékeket a robot kamerán keresztül felismeri, felemeli, majd egy kamerába megmutatva egy kezelő eldöntheti, hogy a termék hibás, vagy megfelelő minőségű.

Egy ilyen rendszer esetében rendkívül fontos, hogy minimális késleltetéssel rendelkezzen, hiszen nehéz úgy precízen pozicionálni egy robotkart, ha másodpercek elteltével látjuk csak a mozgatási utasításunk eredményét. Erre kitűnő lehetőséget adnak az 5G hálózatok, ha megfelelő technológiákat használunk felette adatküldésre.

---

<sup>1</sup> A téma leírásából

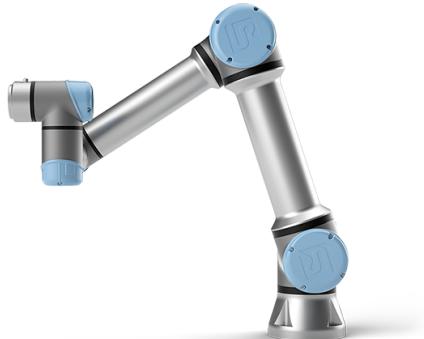
## 2. A használni kívánt technológiák vizsgálata

Mint manapság bármilyen szoftverprojekt esetén, a szóban forgó témakörben is rengeteg lehetőség közül választhat egy architekt, illetve fejlesztő, hogy milyen technológiákat, keretrendszeret szeretne használni projektje során. Igyekeztem minél több lehetőséget vizsgálni, kipróbálni, és kiválasztani a számomra leginkább megfelelőt.

### 2.1. Robotkar vezérlés

A robotkar vezérléséhez a legkézenfekvőbb keretrendszer a ROS, vagyis a Robot Operating System<sup>2</sup> volt. A ROS egy nyílt forráskódú szoftvercsomag, amely hasznos fejlesztői eszközökkel, algoritmusokkal és driverekkel rendelkezik bármilyen robotokkal kapcsolatos projekt számára. A beépített moduljain kívül rendkívül sok kiegészítőt és csomagot lehet telepíteni hozzá. Tartalmaz minden olyan alapvető funkcionálitást, amely szinte minden robotvezérléshez szükséges, így nem kell mindenkinnek ezeket megírni, illetve mivel sokan fejlesztik gyorsabb, hatékonyabb is lesz. Több, mint 10 éves múltja során hatalmasra nőtte magát a projekt, számos kutatási és oktatási projektben is jelen van.

A ROS-nak több verziója is van, én eleinte a ROS 1 Kinetic disztribúcióját használtam, mert azzal kompatibilis a konfigurációs csomag, mely szükséges volt az általam használt robotkarhoz. A ROS-ra épülve a MoveIt!<sup>3</sup> keretrendszer is használtam, mely egy absztraktiós szintet képezve lehetővé teszi, hogy a megfelelő konfiguráció és driver használata esetén ugyanazon vezérlő logika segítségével bármilyen fizikai kar vezérelhető legyen. Ezen kívül számos hasznos funkciót kínál még, amik könnyebbé teszik a fejlesztést, például a robotkar mozgásának tervezését, ütközésdetekciót és még sok másat.



**1. ábra.** A tanszéken is megtalálható UR5e robotkar

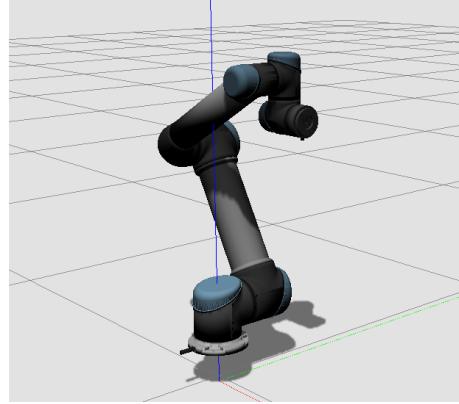
A projekt második félévének egyik első lépéseként átmigráltam a projektet ROS 1 Noetic verzióra, amivel a vezérlés hátterében futó operációs rendszert is ki kellett cserélni Ubuntu 20.04-re a korábbi Ubuntu 16-ról. A verzióváltás hátterében egyrészt az is állt, hogy a tanszéki robotkar mellett, annak vezérléséhez használt számítógépen mások által megtörtént ez a frissítés, de én is hasznosnak láttam. A továbbiakban részletezem még a hasznosságát, de talán az egyik legnagyobb különbség, hogy a Noetic verzió alatt már Python 3-at lehet használni, ami ideális, hiszen korábban a nyelv 2-es verziójához nem állt rendelkezésemre olyan csomag, ami hasznos lett volna.

A fejlesztéshez rendelkezésemre állt a tanszéken egy Universal Robots UR5e típusú robotkar, alapvetően erre fejlesztettem a rendszert. A MoveIt miatt viszont tulajdonképpen bármikor ki lehetne cserélni az eszközt.

<sup>2</sup><https://www.ros.org/>

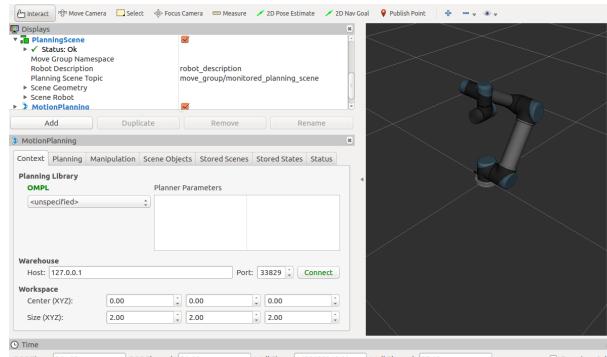
<sup>3</sup><https://moveit.ros.org/>

A Moveit másik előnye, hogy lehetőséget ad arra, hogy egy szimulátorban működő robotkarral is ugyanúgy képes működni, mintha egy valódi eszköz lenne hozzákapcsolva. Annyit kell csak tenni, hogy a szimulátorral együttműködő drivert kell elindítani a fizikai karral kommunikáló helyett.



**2. ábra.** Robotkar a szimulátorban

A robotkar megjelenítéséhez a Gazebo<sup>4</sup> nevű szimulátort választottam, ebben lehetőség van a robot környezetének virtuális megépítésére is, illetve ugyanúgy lehet vezérelni benne a robotkart, mintha fizikai valójában lenne jelen. A vezérlés teszteléséhez az RVIZ<sup>5</sup> használható, mely szoftverben láthatjuk akár azt amit a robot lát, illetve vizualizálhatjuk a robotnak szánt mozgások végrehajtását.



**3. ábra.** Az Rviz felülete

A ROS programozására lehetőség van Python és C++ nyelven is. Én a Python mellett döntöttem, mert a későbbiekben szeretném gépi látással is bővíteni a projektet, amelyre megítélésem szerint tapasztalat nélkül alkalmassabb ez a nyelv. Főleg a Python 3, ami az újabb ROS-sal működik rengeteg lehetőséget rejt magában.

## 2.2. Valós idejű vezérlés IP alapon

Fontos része a rendszernek az irányítási parancsok, illetve metaadatok szinte valós idejű átvitele is. Ennek érdekében vizsgáltam több adatátviteli lehetőséget is. A három talán

<sup>4</sup><http://gazebosim.org/>

<sup>5</sup><http://wiki.ros.org/rviz>

### 1. táblázat. Üzenetküldő szolgáltatások összehasonlítása

	Kafka	Pulsar	RabbitMQ
Maximális adatátvitel	605 MB/s	305 MB/s	38 MB/s
Késleltetés	5 ms (200 MB/s terhelés)	25 ms (200 MB/s terhelés)	1 ms (30 MB/s terhelés)

A RabbitMQ átviteli sebessége 30 MB/s terhelés felett jelentősen csökkeni kezd.

legelterjedtebb eszköz, melyeket találtam erre a cérla az Apache Kafka, Apache Pulsar, illetve a RabbitMQ.

A Kafka<sup>6</sup> egy nyílt forráskódú elosztott esemény közvetítő platform, az Apache alapítvány 5 leginkább futó projektjeinek egyike. Beépített adatfolyam-feldolgozással rendelkezik, számos szolgáltatáshoz tud kapcsolódni és sok programozási nyelven dolgozhatunk fel segítségével eseményeket. Nagy átviteli kapacitással rendelkezik, akár 2 ms-os késleltetést is lehetséges elérni vele. Több ezer cég használja, ami bizonyítja megbízhatóságát.

A Pulsar<sup>7</sup> egy szintén nyílt forráskódú, felhő-alapú elosztott üzenetküldő szolgáltatás, melyet kezdetben a Yahoo!-nál fejlesztettek ki, jelenleg ez is egy rendkívül fontos Apache projekt. A platform könnyen skálázható, számos programozási nyelvhez rendelkezik API-val és akár 5 ms-os késleltetés is lehetséges vele.

A RabbitMQ<sup>8</sup> az egyik legnépszerűbb nyílt forráskódú üzenetküldő bróker. Kis méretű, könnyen telepíthető akár felhőbe is. Több üzenetküldési protokollt is támogat, alapvetően AMQP-re épül. Üzenetsorok segítségével aszinkron üzenetküldést tesz lehetővé, webes felületén könnyen lehet monitorozni a rendszer forgalmát. Hasonlóan a fentebb ismertetett eszközökhöz, itt is lehetőség van számos kiegészítő telepítésére, melyek segítségével képes támogatni Continuous Integration mechanizmust, illetve integrálható más enterprise rendszerekkel.

Mint az az 1. táblázatban is látszik, a Kafka, illetve a Pulsar egy teszt során jelentősen nagyobb adatátvitelre képes, mint a RabbitMQ. Nekem azonban mivel nem tervezek nagy mennyiségű adatot átvinni, fontosabb az üzenet késleltetése, ami a RabbitMQ esetén a legkevesebb. A 4. generációs mobilhálózatok 200ms-os késleltetésével szemben az 5G-s hálózatokon akár 1 ms-ig is csökkenthető a látencia. Ennek kihasználására a tesztek alapján a RabbitMQ a legalkalmasabb.

A felsorolt technológiák közül a RabbitMQ-val találkoztam, azt korábban használtam már. Előfordulhat, hogy emiatt elfogultságom miatt, de valószínűleg a kutatásom során talált tényekre alapozva amellett döntöttem az alkalmazás üzenetküldéseinek kezelésére. További előnye a RabbitMQ brókernek, hogy futtatható Docker konténerként, ami nagyban megkönnyíti a telepítést.

### 2.3. VR/AR megjelenítés

A téma elején szeretném ismertetni, hogy eddigi életem során a legkomolyabb VR „élményt” egy Google Cardboard<sup>9</sup> jellegű VR szemüvegen éltem át, így sok tapasztalom nincs ilyen berendezésekkel. A komolyabb VR szemüveg kipróbálásán kívül értelemszerűen nem is fejlesztettem még ilyen eszközre. Szerettem volna viszont megismerni velük, ezért is kezdtem bele ebbe a projektbe.

<sup>6</sup><https://kafka.apache.org/>

<sup>7</sup><https://pulsar.apache.org/>

<sup>8</sup><https://www.rabbitmq.com/>

<sup>9</sup><https://arvr.google.com/cardboard/>

Ebben a téma körben az első, talán legfontosabb döntés, melyet meg kellett hozzak, hogy VR, vagy AR megjelenítést szeretnék. Mivel alapvetően távvezérlést szeretnék megvalósítani, ezért nincs feltétlen értelme a felhasználót körülvevő valóságot látni, így virtuális valóság mellett döntöttem a kiterjesztett valósággal szemben.

VR megjelenítésre több szemüveg közül lehet választani, azonban tervezési szempontból igazából mindegy, hogy melyiket szeretnénk a későbbiekben használni, hiszen a legtöbb fejlesztői környezet képes bármelyikre előállítani alkalmazást. Eltérsége lehet a VR megjelenítőkhöz tartozó kézi irányítószervek között, ígyekeztem azokat is megvizsgálni, melyik mire alkalmas. Végül én HTC Vive-ra terveztem megvalósítani a szoftvert, viszont végül a szemüvegen való kipróbálásig nem jutottam el a projektben.

Ezen alapvető kérdések eldöntése után már csak azt kellett eldöntsem, hogy milyen fejlesztői környezetben szeretném elkészíteni a VR-képes alkalmazást. Talán a két legnépszerűbb piaci szereplő ilyen célokra a Unity, illetve az Unreal Engine. A kettő közül a Unity a jobban elterjedt környezet VR fejlesztésekben. Alapjában véve egy játékmotor, viszont nem csak szórakoztató alkalmazásokat lehet készíteni benne. Hatalmas közösség van mögötte, rengeteg segítő dokumentum és könnyen letölthető asset áll rendelkezésre. A Unity talán legnagyobb ellenfele az Unreal Engine, amely szintén egy játékmotor, VR képességekkel. Az ő esetében is rendelkezésre áll számos letölthető erőforrás, a megjelenítése pedig valósághűbb tud lenni. Ez a keretrendszer fiatalabb, modernebb, mint a Unity, jobb teljesítményre képes.

Mivel egyik fejlesztői környezetet sem használtam még korábban, a tanulási görbüjük kutatásom szerint nagyjából azonos, viszont a Unity mögött nagyobb támogatottság áll VR fejlesztés terén, ezért azt választottam.

## 2.4. Videójel továbbítása a robotkartól a vezérlő kliensbe

Egy igen nagy feladat volt számomra a videó átviteli technológia kiválasztása a ROS-t futtató gép és a vezérlő kliens között. Ennek az esetében is rendkívül fontos, hogy minél gyorsabban jusson el a kép a felhasználó szeme elé.

Első ötletem egy RTMP stream volt, hiszen ezzel már találkoztam, igen elterjedt a videómegosztó platformokra történő streameléshez. Arra azonban nem gondoltam hirtelen, hogy ott nem fontos a késleltetés minimalizálása. Ez is lett a fő probléma ezzel az átviteli móddal, hiszen 10 másodperces késleltetés alá szinte lehetetlen levinni a közvetített videót. Emiatt ez teljesen alkalmatlan erre a felhasználásra, így más technológiák után kezdtem kutatni.

Eleinte a lehetséges átviteli technológiák közül a WebRTC-t, vagyis Web Real Time Communication-t választottam ki, mert azt találtam róla, hogy amellett, hogy szöveges üzeneteket lehet vele küldeni, rendkívül egyszerűen működik vele a videóátvitel is, nevéből is adódóan szinte valós időben. Azonban mivel Unity-ben szerettem volna működésre bírni, sajnos nem volt vele szerencsém a tesztelés során. Webes alapon sikeres volt megjelenítenem videostreamet viszonylag gyorsan, viszont sajnos Unity-ne belül többszöri próbálkozás után sem épült fel a kapcsolat a videót streamelő fél és a Unity kliens között.

A viszonylag sok nem működő teszt közben felmerült az ötlet bennem, hogy akár NDI-on keresztül is lehetne videót átjátszani. Az NDI egy broadcast iparban feltörekvő IP alapú videotovábbító szabvány, mely produkciós környezetbe tervezettsége miatt kitűnő minőségű átvitelre képes. Arra a döntésre jutottam viszont, hogy az általa képviselt minőség miatti nagyjából 100-150 MB/s-os adatátvitelle felesleges ebben az alkalmazásban. Ezen kívül nem tudom, hogy publikus hálózaton keresztül hogy működne, így elvetettem ezt az ötletet.

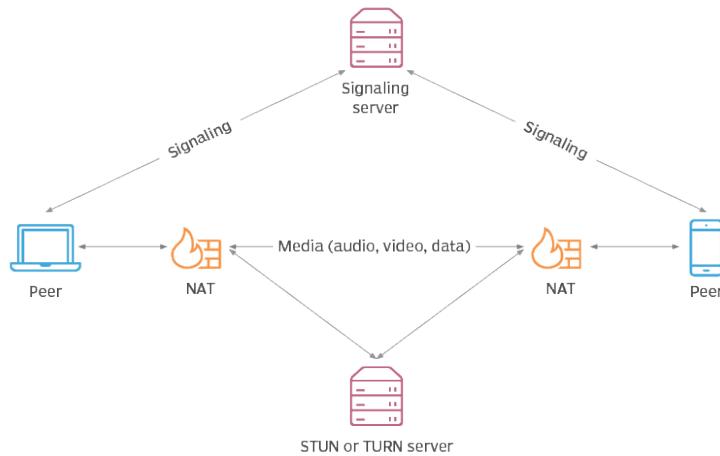
Az NDI megfontolása utáni további sikertelen RTC tesztek után jutottam el az RTP-hez. FFmpeg<sup>10</sup> segítségével könnyen tudtam készíteni egy teszt streamet egy videó segítségével, melyet a generált *SDP* fájl minimális módosítása után le is tudtam játszani VLC media playerrel. Ez után már könnyen megvalósítható volt, hogy Unity-ben is megjelenjen a stream, hiszen létezik könyvtára és azt tudtam alkalmazni.

## WebRTC videó streamelés

Az első félévben elvettem a WebRTC-n történő videójel átvitelt, ami annak volt köszönhető főként, hogy nem tudtam megoldani, hogy Python-ból ebben a környezetben streameljek. Most azonban a ROS újabb verziójával kompatibilis Python3-hoz van olyan könyvtár, amely lehetővé teszi ezt, úgyhogy visszatértem ehhez a megoldáshoz.

A WebRTC mellett szól, hogy igen könnyen lehet webböngészőben is megjeleníteni belőle videót, hiszen főleg erre lett kifejlesztve, illetve gyorsabbnak mondják, mint az RTP-t, amit korábban használtam. Azzal nekem is az volt a tapasztalom, hogy körülbelül egy-két másodperc eltelt aközött, hogy megmozdult a robot és, hogy láttam ezt a mozgást a videón is. Az én alkalmazásomban kritikus lenne a lehető legkisebb késleltetés, hiszen csak úgy lehet hatékonyan irányítani bármit, ha azonnal látjuk mi történik vele.

## How WebRTC works



**4. ábra.** A WebRTC architektúrája

A 4. ábrán látszik a WebRTC működési struktúrája:

- Első lépésként a videót küldeni kívánó partner elküld egy SDP leírót a fogadó félnek címzve, ekkor ő még nem tudja annak a címét, így szükség van egy signaling controller-re, amely továbbítani tudja az üzenetet.
- Ez után a fogadó fél megjegyzi a kapott információkat és generál rá egy választ, szintén egy SDP leírás formájában, amit a signaling serveren keresztül eljuttat a küldő félnek.

<sup>10</sup><http://www.ffmpeg.org/>

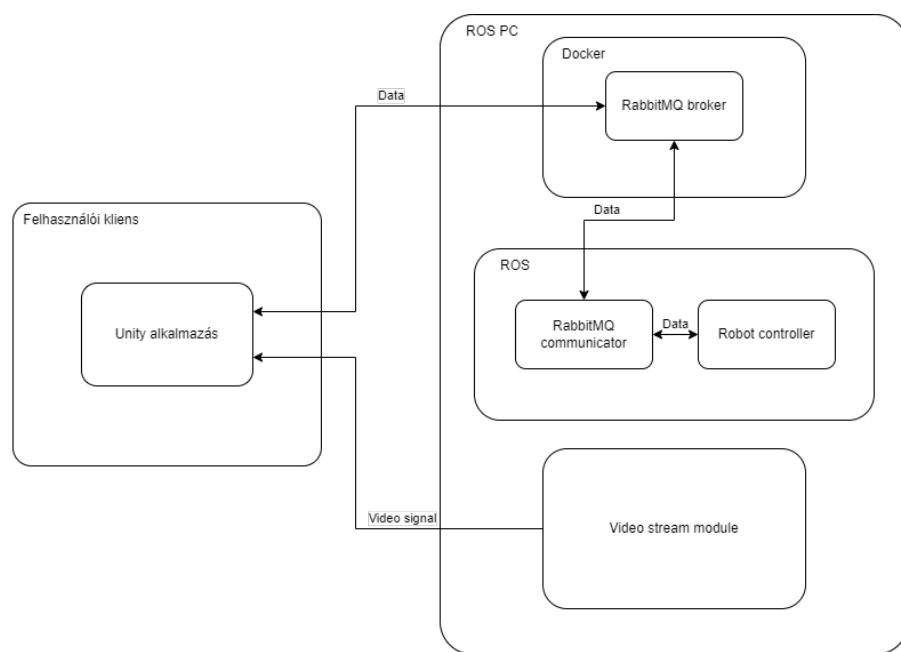
- Ekkor tulajdonképpen fel tud épülni a kapcsolat közöttük. Amennyiben NAT mögött vannak a felek szükség lehet STUN vagy TURN szerverekre, amelyek központi továbbítóként helyet foglalnak a két fél között, hiszen a NAT miatt közvetlen kapcsolat nem lenne lehetséges.

Signalingra többféle megoldás közül lehet választani. A legegyszerűbb módja, ha valaki kézzel átmásolja az üzeneteket a két fél között, bár ez minden napos használati környezetben érhető okokból nem kivitelezhető. Mivel a cél csupán annyi, hogy az üzenetek eljussanak bármilyen megbízható úton, tulajdonképpen bármit választhatunk, ami nekünk kényelmes. Lehet az egy TCP kapcsolat, vagy más.

Signaling szervernek én a már amúgy is használt RabbitMQ brókert választottam, abban definiáltam üzenetküldő sorokat, melyeken tudják a felek küldeni egymásnak a szükséges információkat. Annak érdekében, hogy az üzenetek a megfelelő félhez jussanak el, két sort definiáltam, egyet, melyen a küldő fél elküldi az offer-t, illetve egy másikat amelyen a vezérlő kliens - aki fogadni fogja a videót - válaszol. Ilyen módon elkülönül a kommunikáció iránya, és nem történhet meg véletlenül sem az, hogy az, aki elküldi az üzenet meg is kapja azt.

## 2.5. A rendszer tervezett felépítése

A technológiák kiválasztása után a következő felépítés alakult ki a tervezemben:



**5. ábra.** A rendszer tervezett felépítése

Mint az látható az 5. ábrán, a felhasználói kliens a RabbitMQ brókeren keresztül kommunikál IP alapon a ROS-on belül található communicator ROS node-al. Ez továbbítja a beérkező üzeneteket a robotkar mozgatásáért felelő modulnak, illetve helyzetjelentést küld a kar állapotáról a kliensnek.

Ettől függetlenül történik a kamerakép streamelése a kliensnek IP alapon.

### 3. Megvalósítás

#### 3.1. ROS

Mivel alapvetően Windows operációs rendszert használok, eredetileg megpróbáltam erre telepíteni a ROS környezetet, viszont miután feltelepítettem sajnos nem működött rendesen. Mivel az Ubuntu az ajánlott operációs rendszer, ezért készítettem egy virtuális gépet, melyre Ubuntut, majd a ROS-t telepítve szerencsére működött a rendszer.

A ROS alap csomagok telepítése után elkészítettem a projekt alapját adó Catkin workspace-t a megfelelő mappastruktúra kialakításával, majd a *catkin\_make* parancs futtatásával. Ennek a könyvtárrendszernek a *src* mappájába kell írjuk a forráskódunk, melyet szeretnénk futtatni ROS-on belül. Ebbe a mappába helyeztem el a robotkarhoz tartozó MoveIt konfigurációt is, melyet futtatni kell, hogy a vezérlése eljusson hozzá.

A robotkarhoz fontos, hogy megfelelő drivert használunk. Több ilyet is kipróbáltam, mert változó kompatibilitással rendelkeznek ezek a csomagok. Az általam használt csomag eleinte a Githubon megtalálható *Universal Robot*<sup>11</sup> volt, viszont mikor elkezdtem a fizikai robotkarral dolgozni rá kellett jöjjek, hogy azzal ez nem működik, ekkor tértem át az *Universal Robots ROS Driver*<sup>12</sup> csomagra. Mindkét csomag rendelkezik MoveIt! támogatással is és használhatjuk vele az Universal Robots számos karját. (A korábban használt csomag nem támogatja többek között a gyártó újabb, e-szériás robotkarjait, amilyen a tanszéki laborban is megtalálható.)

A ROS-on belül eleinte két fő node-ot valósítottam meg Python-ban, mint az az 5. ábrán is látszik. Az egyik felelős az IP alapú kommunikáció feldolgozásáért és átfordításáért a ROS-on belülre. Tőle kapja meg a mozgatási információkat a másik fő modul, a robotkar mozgató komponens. Ő végzi a robottal való kommunikációt, utasítja, hogy hova mozogjon, és kiolvassa az állapotát is.

A 6. ábrán látszónak a fejlesztés közben kialakult új modulok, illetve az azok közötti kommunikáció. Az egyik legnagyobb változás a webes vezérlő megjelenése, amelyet a 3.2. részben tárgyalok. Ezen kívül látszik, hogy hogy illeszkedik a rendszerbe az új videóküldő modul.

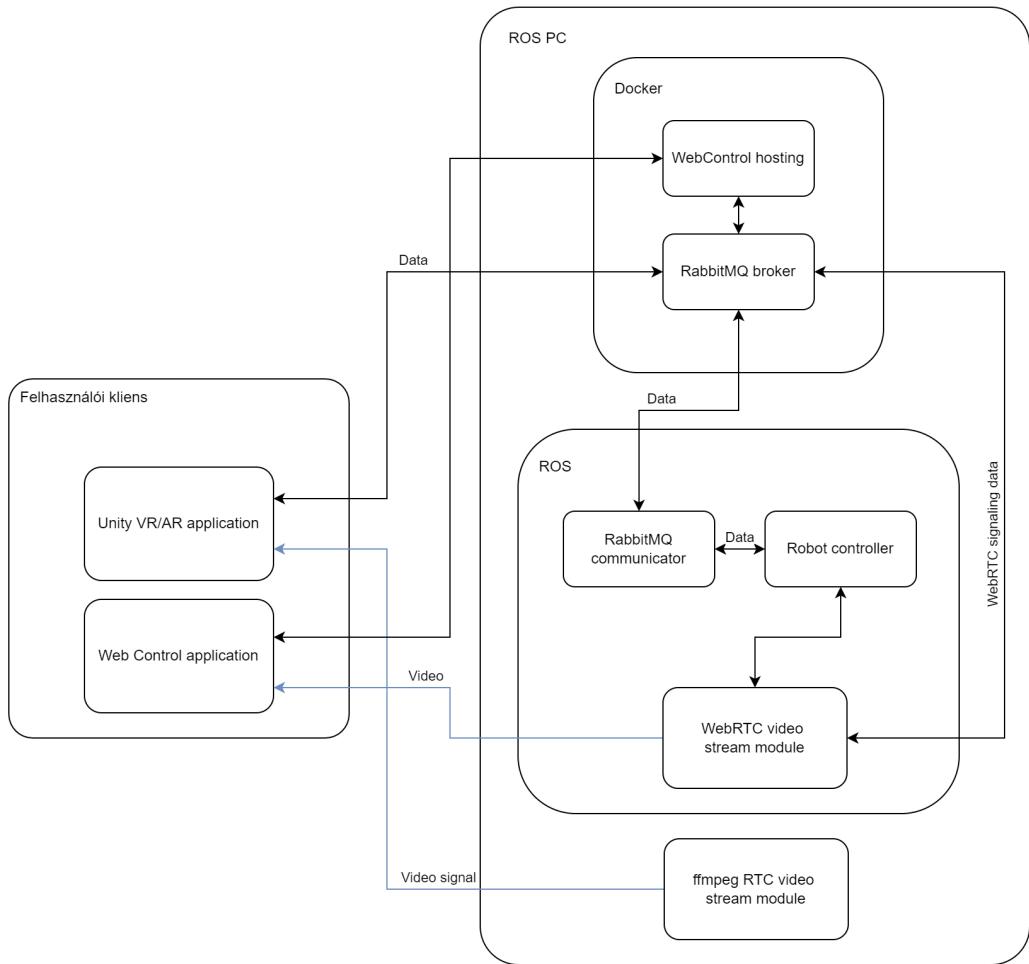
A feladat leírásában szerepel, hogy egy futószalagon érkező termékeket egy, a robotkar mellett elhelyezett kamerán keresztül lehet látni. Az első félév során ilyen szintre nem jutott el a fejlesztés, viszont a kamerakép továbbítását mindenkorban szerettem volna valamilyen módon helyettesíteni. Erre a célról akkor az ffmpeg segítségével egy képernyőképet továbbítottam a felhasználói kliens felé, mint az látható a 7. ábrán.

A második félévben, mivel váltottam ROS Noetic disztribúcióról, át kellett álljak Python3-ra. Ez különösebben nagy változtatásokat nem igényelt az eddig megvalósított komponensekben, viszont új lehetőséget nyitott meg előttem. Már korábban is megtaláltam az *aiortc*<sup>13</sup> Python könyvtárat, viszont használni nem tudtam a ROS kinetic Python2 kötöttsége miatt. Újabban viszont már bele tudtam építeni a rendszerbe és mivel kisebb késleltetést ígér a WebRTC az RTP-nél, meg is tettem. Ezzel a videostreamelés is be tudott kerülni a ROS-on belül egy modulba, ami által jobban egy egységet képez a rendszer és a videós modul is tud kommunikálni a ROS ökoszisztemája többi tagjával, hogy például letiltsa a robot mozgását, ha megszűnik a videós kapcsolat, hiszen az igen veszélyes helyzetet tudna előidézni, ha a kezelő nem látja mit csinál, de a robot továbbra is végrehajtja az utasításait.

<sup>11</sup>[https://github.com/ros-industrial/universal\\_robot](https://github.com/ros-industrial/universal_robot)

<sup>12</sup>[https://github.com/UniversalRobots/Universal\\_Robots\\_ROS\\_Driver](https://github.com/UniversalRobots/Universal_Robots_ROS_Driver)

<sup>13</sup><https://github.com/aiortc/aiortc>



**6. ábra.** A rendszer felépítése a 2022 tavaszi félév végén

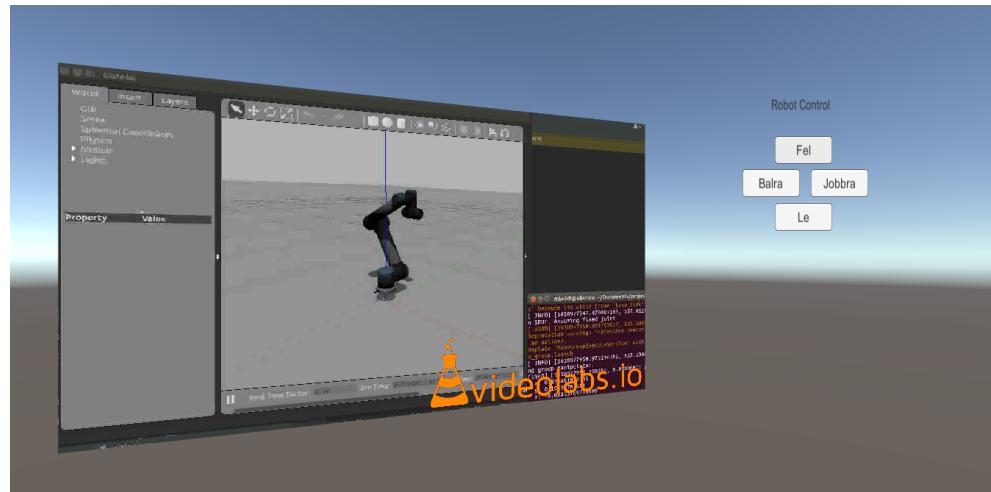
### 3.2. Felhasználói kliens

#### VR/AR kliens

Mint azt a tervek között is írtam, a kliensoldali alkalmazást, mely VR szemüvegen, annak kezí vezérlőivel kiegészítve kerül használatra, Unity környezetben készítettem. Korábban még nem használtam ezt a feljesztői eszközt, így eleinte nehézséget okoztak az alapvető feladatok is. Számos tesztalkalmazást készítettem különböző használni kívánt technológia kipróbálására, ezeknek a sikeres működése után kezdtem el elkészíteni a tényleges szoftvert.

Az alkalmazás design-ja jelenleg igen kezdetleges, minden képpen szeretnék rajta még változtatni, illetve javában szükséges bővíteni is. Helyet kapott egy virtuális képernyő, melyen megjelenik a ROS felől érkező videójel. Emellett kaptak helyet az egyelőre gomb formájában megvalósult vezérlőszervek. Jelenleg csupán alapvető mozgatási parancsokat tudunk küldeni a robotkarnak, ez is bővülni fog a jövőben, viszont a tényleges vezérlést nem ilyen féle felülettel szeretném elkészíteni, hanem a VR szemüveghez tartozó kézi vezérlőkkel. Az ezek adta lehetőségeket kihasználva minél kevesebb hagyományos interakciós eszközöt szeretnék használni, mint például gombokat.

A 7. ábrán látható az elkészült kliens felülete. Megvalósításra vár még a tervekben említett több futási mód, illetve szeretném a robot virtuális megjelenítését is, esetleg a környezetével együtt, így is megkönnyítve a kezelő munkáját, lehetővé téve neki, hogy minden-



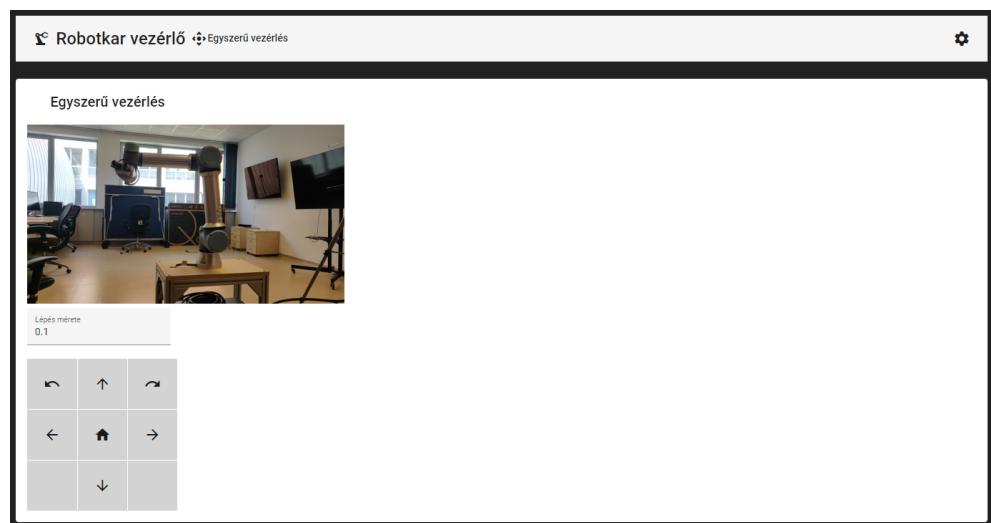
**7. ábra.** A felhasználói kliens felülete IP alapon fogadott videóképpel

lássa a robotkar aktuális állapotát.

A videó fogadását a VLCLib Unity alá készült verziója végzi, ami egy viszonylag nagyméretű vízjelet helyez el a képen. Ezt szeretném megoldani majd, hogy ne látszódjon, akár más fogadó könyvtár használatával.

### Webes kliens

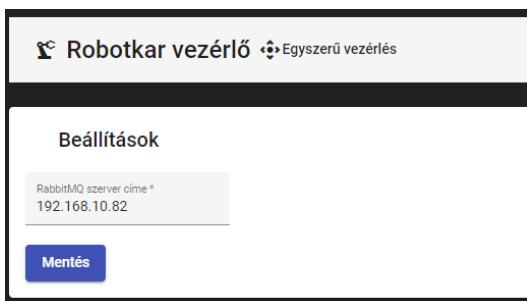
A korábbi fejlesztés során több új keretrendszerrel is meg kellett ismerjek egyszerre, úgy éreztem ez csökkenti a fejlesztés hatékonyságát, így megpróbáltam kizártani legalább egyet a teljesen ismeretlen platformok közül. Mivel eleinte inkább a robot vezérlésének összeépülésére szerettem volna koncentrálni, ezért úgy döntöttem a VR kliens és annak környezete, a Unity helyett készítettem egy egyszerű webes vezérlőfelületet. Az elmúlt években több webalkalmazást is készítettem már, így abban kényelmesebben érzem magam, mint Unity alatt, ahol még semmit nem készítettem.



**8. ábra.** A webes kliens vezérlő oldala

A kliens fő oldala a vezérlőfelület, ahol látjuk a robottól érkező videóképet, illetve tudunk neki üzeneteket küldeni. Ez a felület látható a 8. ábrán. Lehetőség van a kar fejét több irányba mozgatni, forgatni a kart, illetve egy előre definiált home pozícióba való visszatérésre is tudjuk utasítani. Ez még mindig nem a megfelelő mennyiségű szabadsági fokkal rendelkező irányítás, ami a könnyű vezérléshez szükséges, de tesztelésre megfelelő. Kísérleteztem a félév során egy webkamerán keresztül a felhasználó kezét felismerő rendszer használatával, viszont annak a fejlesztése nem került olyan állapotba, hogy rendesen működjön.

A VR felülettel korábban probléma volt, hogy ha máshol futtattam a rendszer, akkor a változó IP címek miatt változtatnom kellett a forráskódot. Ez normális környezetben értelemszerűen nem kivitelezhető, így a webes kliens kapott egy beállítások oldalt, ahol meg tudjuk adni a ROS PC IP címét, amin elérhetővé válik számunkra. Ez a felület látszik a 9. képen. Itt a továbbiakban még más beállítási lehetőségek is meg fognak jelenni tervezem szerint.



**9. ábra.** A webes kliens beállításokat tartalmazó felülete

A webes kliens feladata tehát leginkább az volt, hogy teszt jelleggel könnyebben tudjak küldeni vezérlő üzeneteket a ROS felé és tudjam tesztelni az onnan érkezőket. A használata közben rájöttem viszont, hogy bizonyos konfigurációs feladatokra valószínűleg praktikussabb, mint a VR környezet, így valószínűleg a későbbiekben is meg fogom tartani.

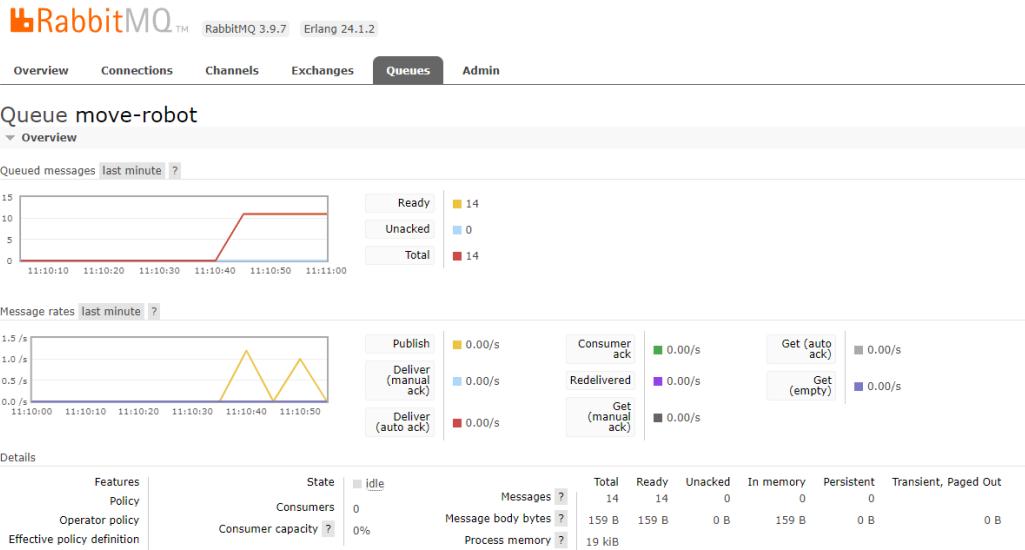
### 3.3. Kommunikáció

A kliens és a ROS közti kommunikáció RabbitMQ-n keresztül zajlik. Ehhez szükséges egy központi irányító szervert futtatni, ezt én Docker-ben tettem meg, hiszen rendkívül egyszerűen kezelhető így. Mivel szerencsére C# és Python alá is rendelkezik könyvtárral, mellyel könnyen megoldható a kommunikáció, nem volt vele nagy gondom. A 10. ábrán is látható, hogy rendelkezik egy praktikus webes felülettel is, melyen nyomon lehet követni az üzenetsoraink állapotát.

A webes kliens nem képes közvetlenül AMQP protokollon kapcsolódni a RabbitMQ szerverhez, így szükség volt arra is, hogy a Rabbit-et kiegészítsem egy pluginnal, amely WebSocket felett, STOMP protokoll segítségével is továbbít üzeneteket, amiket már tud fogadni a kliens.

A terveimet úgy készítettem el, hogy a különböző célú üzenetek más-más üzenetsorokban kerülnek küldésre. Ezek a sorok többek között:

- *robot-status*: itt küldi a ROS felőli oldala a rendszernek a robot állapotát kis időközönként



10. ábra. A RabbitMQ webes felülete

- *move-robot*: itt megadhatunk egy (x, y, z) alakú vektort, amivel a robot fejét szeretnénk elmozgatni
- *set-joints*: ebben a sorban lehet olyan üzenetet küldeni a ROS fele, amelyben a robot csuklóinak egy kívánt állapota szerepel
- *webrtc*: WebRTC signalinghoz használt csatorna
- *webrtc-resp*: WebRTC signalinghoz használt csatorna

Az alkalmazás selejtdetekcióra való használatához további üzenetcsatornák lesznek szükségesek, ezeknek a tervezését még nem véglegesítettem.

A ROS-on belül a saját topic alapú kommunikációját használtam. Többek között kerülnek átfordításra az IP alapon kívülről érkező csomagok is, egyéb szükséges kommunikáció is ezen keresztül zajlik a ROS node-ok között.

### 3.4. Valós robotkar vezérlése

A második féléves munka egyik nagyobb célja volt, hogy a szimulátoros tesztelés után a valós robotkar is megmozduljon.

A legnagyobb akadályokat ebben is az okozta, hogy nem rendelkezem tapasztalattal ilyen területen, minden fel kellett fedezzék és körül kellett járjak, mielőtt sikerült megvalósítanom. Az általam eredetileg használt, a szimulátorban működő driver a robotkarhoz nem volt megfelelő, így több verziót ki kellett próbáljak. Volt olyan driver is, amelynek buildelése során a `catkin_make` hibával leállt, viszont később kiderült számomra, hogy az a megfelelő driver, csupán mivel korábban használtam egy régebbi verzióját, azért nem tudott lefutni és ha kitöröltem a korábban előállt futtatható állományokat, akkor már nem ütközött hibába.

A megfelelő driver kiválasztásával egyidőben ismerkedtem azzal is, hogy miként lehet kapcsolódni a robotkarhoz fizikailag, illetve hogy működik annak a saját vezérlője. A robot központi vezérlője és a ROS-t futtató számítógép egyszerű IP hálózaton keresztül kommunikálnak, ami jelen esetben csupán egy, a két eszközöt összekötő UTP kábel. A robothoz



**11. ábra.** Az Universal Robots 5e robotkar a laborban

tartozik egy kézi vezérlő egység is, ezen keresztül lehet konfigurálni, illetve saját programokat készíteni rá. Mivel ez a robotkar egy kollaboratív robot, a megfelelő opció kiválasztása után kézzel is lehetőség van mozgatni a kart egy kívánt pozícióba. A kézi egység érintőkijelzőjén megjelenő gombokkal lehetőség van manuálisan is mozgatni a kart, hasonlóan ahhoz a rendszerhez, melyet én készítettem, csupán ennek az esetében korlátozza a vezérlő személy hollétét az egységet és a robotkart összekötő kábel hossza.



**12. ábra.** A robotkarhoz tartozó kézi vezérlő egység

Több a robotkarhoz való sikertelen kapcsololódás után derült ki számon, hogy nem megfelelően próbáltam elérni ezt a célt, ugyanis a kar vezérlőjén el kell indítani egy programot, amely felépít a kapcsolatot a robot és a számítógépen futó vezérlés között. Miután ezt is megfelelően csináltam, már működött is a vezérlés az én rendszerem által. Tudtam vezérelni a kart és kaptam tőle visszajelzést is az állapotáról, a csuklók pozíciójáról.

## 4. Összefoglaló gondolatok

Azt gondolom, hogy 1 év alatt, ami óta ezt a projektet csinálom, jelentősen nagyobb előrehaladás is elérhető lett volna, ha rendelkezem tapasztalattal a releváns területeken. Nagyjából minden, amit használtam a rendszer összeállításához, a ROS, a Unity, kis késleltetésű videostreamelés, maga a robotkar, de Pythonnal, Ubuntuval se foglalkoztam sokat korábban. Mivel nem egy minden nap, bárki által otthon is fejleszthető alkalmazás a robotika, jelentősen kevesebb leírás, erőforrás áll rendelkezésre is az interneten ebben a témakörben.

Rendkívül sok új technológiát ismertem meg a rendszer fejlesztése során. A használt eszközök nagyon szerteágazók, a robotikán kívül sokat kellett foglalkozzak még más olyan területekkel is, amelyek számomra nem voltak egyértelműek, rendkívül sokat kellett után-aolvassak dolgoknak. Nagyon nehéz volt számomra úgy fejleszteni egy rendszert, hogy ha megcsináltam sok próbálkozás után egy részét, akkor a következő részénél is ugyanúgy majdnem minden lépést sok kutatás kellett megelőzzön, nem volt egy biztos pontja, amit jól ismertem. (Ezért döntöttem a második félév elején a webes irányítás elkészítése mellett, amelynek szintén voltak olyan részei, amihez hasonlót korábban még nem csináltam.)

A második félév alatt összességében sajnos nem jutott annyi időm a rendszer fejlesztésére, amennyit szerettem volna eleinte, de remélem, hogy be fogok tudni pótolni az elmaradást.

## 5. További tervezek

A rendszer egy éves munkám alatt eljutott egy olyan fázisba, hogy lehet a robotkart távolról vezérelni és a vezérlő felületen láthatunk egy élő videóképet róla. Nagyon sok időm ment el arra, hogy megismerjem a használni kívánt technológiákat és megfelelő módon kezdjem el használni őket. A jelenlegi állapot már azt gondolom egy jó alap annak, hogy meg tudjam kezdeni a konkrétabb alkalmazások ráépítését és, hogy ki tudjam dolgozni a kezdetekkor megállmodott VR/AR alapú vezérlést is.

Jelenlegi konkrét fejlesztési terveim nagy vonalakban a következők:

- A videótávitel optimalizálása, gyorsítása.
- Kliensalkalmazás kibővítése rengeteg tervezett funkcióval, többek között:
  - Konfigurációs és kezelői mód különválasztása
  - Virtuális robot megjelenítése, amely tükrözi a valósat
  - VR rendszerhez tartozó kézvezérlők alkalmazása irányításra
- Gépi látással való foglalkozás, hogy a robotkar alapvető mozdulatokat végre tudjon hajtani magától