



Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

Alacsony hálózati késleltetésre épülő AR megoldások telefonra és okosszeművegre

MSC ÖNÁLLÓ LABORATÓRIUM 1.

Készítette
Márta Boldizsár

Konzulens
Forstner Bertalan

2021. december 10.

Tartalomjegyzék

1. A feladat leírása	1
2. A használni kívánt technológiák vizsgálata	2
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 személyig	5
2.5. A rendszer tervezett felépítése	6
3. Megvalósítás	7
3.1. ROS	7
3.2. Kommunikáció	7
4. További tervek	8

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 tárolt valósídejű információkra építő ipar, szórakoztató vagy egyéb információs alkalmazások.¹

Rendkívül sok lehetőséget 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 szerzünk róla, ezért szerettem volna ezzel foglalkozni.

A feladatom egy olyan szoftverrendszer elkészítése lett, melynek segítségével távolról, 5G hálózaton keresztül lehet vezérelni egy robotkart VR környezetből, miközben kameraképen 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 folyamaton, vagy selejtként kisselektálja

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 mindennapi használatra alkalmas, ebben képes a kezelő felügyelni és utasítani a robotot működés közben.

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

¹A téma leírásából

2. A használni kívánt technológiák vizsgálata

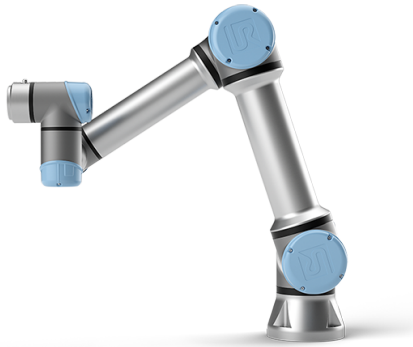
Mint manapság bármilyen szoftverprojekt esetén, a szóban forgó témakörben is rengeteg lehetőség közül választhat egy architekt, illetve fejlesztő, hogy milyen technológiákat, keretrendszereket szeretne használni projektje során. Igyekeztem minél több lehetőséget vizsgálni, kipróbálni, és kiválasztani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

2.1. Robotkar vezérlés

A robotkar vezérléséhez a legkézenfekvőbb keretrendszer a ROS, vagyis a Robot Operating System² 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 funkcionalitást, amely szinte minden robotvezérléshez szükséges, így nem kell mindenkinek 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 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!³ keretrendszert is használtam, mely egy absztrakciós 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 más.

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.



1. ábra. A tanszéken is megtalálható UR5e robotkar

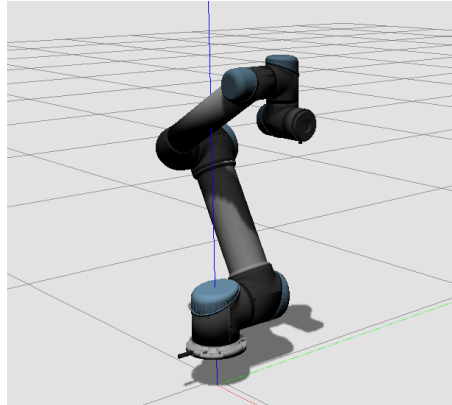
A Moveit másik előnye, hogy lehetőséget ad arra, hogy egy szimulátorban működő robotkarral is ugyanúgy képes működni, mintha egy valódi eszköz lenne hozzákapcsolva. Annyit kell csak tenni, hogy a szimulátorral együttműködő drivert kell elindítani a fizikai karral kommunikáló helyett.

A robotkar megjelenítéséhez a Gazebo⁴ nevű szimulátort választottam, ebben lehetőség van a robot környezetének virtuális megépítésére is, illetve ugyanúgy lehet vezérelni

²<https://www.ros.org/>

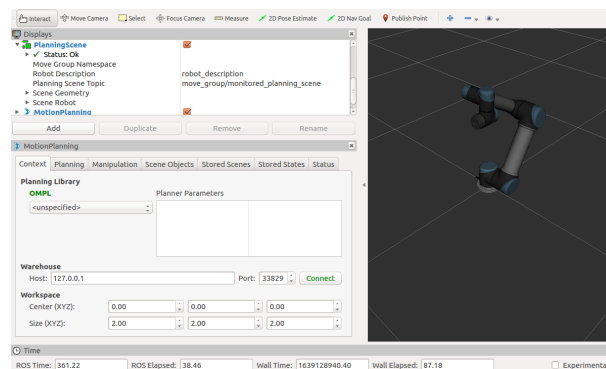
³<https://moveit.ros.org/>

⁴<http://gazebo-sim.org/>



2. ábra. Robotkar a szimulátorban

benne a robotkart, mintha fizikai valójában lenne jelen. A vezérlés teszteléséhez az RVIZ⁵ 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 alkalmasabb ez a nyelv.

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

A Kafka⁶ 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.

⁵<http://wiki.ros.org/rviz>

⁶<https://kafka.apache.org/>

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 csökkenni kezd.

A Pulsar⁷ 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 RabbitMQ⁸ 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 emiatti 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 Cardboard⁹ jellegű VR szemüvegen éltem át, így sok tapasztalatom nincs ilyen berendezésekkel. A komolyabb VR szemüveg kipróbálásán kívül értelem szerűen nem is fejlesztettem még ilyen eszközre. Szerettem volna viszont megismerkedni velük, ezért is kezdtem bele ebbe a projektbe.

Ebben a témakörben 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. Eltérés lehet a VR megjelenítőkhöz tartozó kézi irányítószervek között, igyekeztem azokat is megvizsgálni, melyik

⁷<https://pulsar.apache.org/>

⁸<https://www.rabbitmq.com/>

⁹<https://arvr.google.com/cardboard/>

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 valószínűbb tud lenni. Ez a keretrendszer fiatalabb, modernebb, mint a Unity, jobb teljesítményre képes.

Mivel egyik fejlesztői környezetet sem használtam még korábban, a tanulási görbéjük 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 személyig

Egy igen nagy feladat volt számomra a videó átviteli technológia kiválasztása a ROS-t futtató gép és a vezérlő kliens között. Ennek az esetben 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 sikerült megjelenítenem videóstreamet viszonylag gyorsan, viszont sajnos Unity-ne belül többszöri próbálkozás után sem épült fel a kapcsolat a videót streamelő fél és a Unity kliens között.

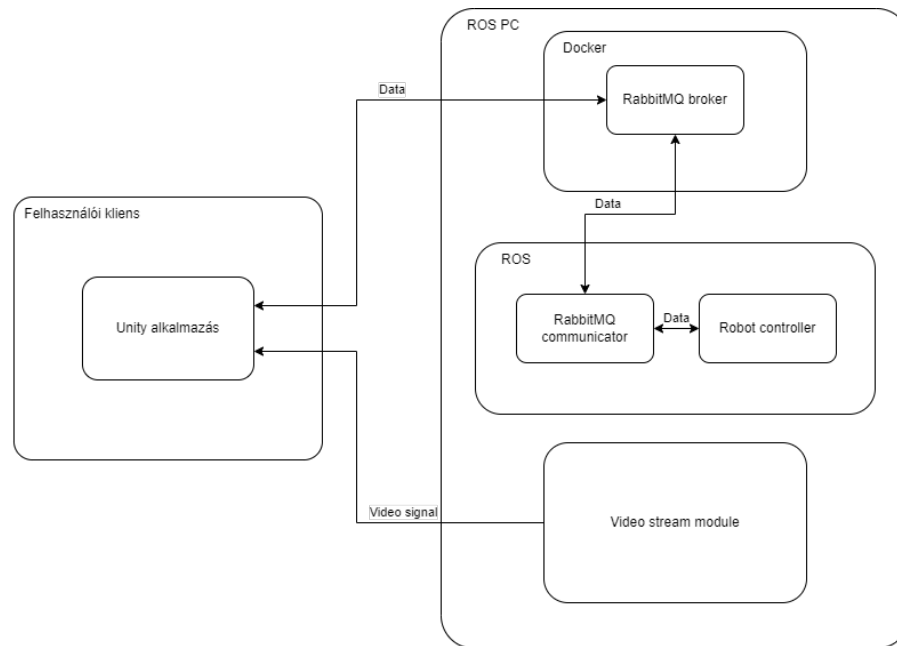
A viszonylag sok nem működő teszt közben felmerült az ötlet bennem, hogy akár NDI-on keresztül is lehetne videót átjátszani. Az NDI egy broadcast iparban feltörekvő IP alapú videótovábbító szabvány, mely produkciós környezetbe tervezettsége miatt kitűnő minőségű átvitelre képes. Arra a döntésre jutottam viszont, hogy az általa képviselt minőség miatti nagyjából 100-150 MB/s-os adatátvitel 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. FFmpeg¹⁰ 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.

¹⁰<http://www.ffmpeg.org/>

2.5. A rendszer tervezett felépítése

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



4. ábra. A rendszer tervezett felépítése

Mint az látható a 4. 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 felelepítettem sajnos nem működött rendesen. Mivel az Ubuntu egy 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 ROS-on belül két fő node-ot valósítottam meg Python-ban. Az egyik felelős az IP alapú kommunikáció feldolgozásáért és átfordításáért a ROS-on belülre. Tőle kapja meg a mozgató információkat a másik fő modul, a robotkar mozgató komponens. Ő végzi a robottal való kommunikációt, utasítja, hogy hova mozogjon, és kiolvassa az állapotát is.

3.2. Kommunikáció

rabbit dockerben

4. További tervek

realebb time