Budapesti Mûszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Távközlési és Médiainformatikai Tanszék

Felhő alapú kooperatív robotikai keretrendszer

Készítette
Horváth István

Konzulens
Dr. Vidács Attila

2018. október 26.

Tartalomjegyzék

Kivonat

A robotok hasznos szerepet töltenek be mindennapi életünkben. Külön-külön is képesek hatékony működésre, azonban vannak olyan kihívások, melyek csak tudatos kooperációval oldhatóak meg. A robotok képességei és számítási erőforrásai végesek, ezek korlátokat állítanak a feladat végrehajtása elé. Ezen megszorítások egy része kiküszöbölhető, ha a robotcellák együtt, a funkcionalitásuk uniójaként érik el a kívánt eredményt. Erre egy egyszerű példa, ha egy *UGV* (*Unmanned Ground Vehicle*) rakományát szeretnénk feljuttatni egy térben magasabb szintre. Az *UGV* két dimenzióban képes elmozdulásra, ezért egy *UAV* (*Unmanned Aerial Vehicle*) - például drón - siet a segítségére, így a szállítmány célba érhet.

Feltörekvő kutatási-fejlesztési terület továbbá a felhő alapú robotika, amely szerint a vezérlőegység a felhőben foglal helyet. A felhő robotika általánosan két célt határoz meg: a virtualizáció és a cella tehermentesítése a komplex számítási feladatok alól. Az általam tervezett rendszer alkalmazza ezen koncepciót.

A rendszer magja *SOA* (*Service Oriented Architecture* - szolgáltatás-központú architektúra) alapokra épül, ahol a komponensek szolgáltatásokon keresztül kommunikálnak egymással. Ez az elgondolás egy moduláris és rugalmas szoftver-architektúrát eredményez, melynek segítségével lehetőség nyílik arra, hogy az eszközök képességeit additívan kezeljük egy bizonyos absztrakciós szinten.

A dolgozat célja egy olyan robotikai együttműködő rendszer kutatása és létrehozása, amely generikusan kezeli a robotok képességeit és adaptív a változásokkal szemben, ezzel elősegítve az optimális problémamegoldást. A fizikai robotok szimulációja végrehajtható virtuális környezetben a *ROS* (*Robot Operating System*) és a *Gazebo* segítségével, melyeket előszeretettel alkalmazzák a robotikai fejlesztési feladatokban. A rendszer továbbá támogatja a komplex robotikai feladatok végrehajtását, amelyben a robotcellák közösen, együttműködve vesznek részt.

Abstract

Robots are very important part of our lives. They can efficiently as individual devices, however there are some particular challenges which in they have to work together. The robot devices have physical constraints that can cause strict limitations in problem solving. We can cure these disadvantages as we handle their functionalities all together, creating their unions. Consider the case following case as an example: We would like to lift a package from an *UGV* (*Unmanned Ground Vehicle*) to an upper floor of the building. The *UGV* can only move in two-dimensional space, so it is unable to accomplish the task. There is another robot device, an *UAV* (*Unmanned Aerial Vehicle*) - e.g. a quadcopter -, which can help the *UGV*.

Cloud robotics is an emerging technology in research and development. The cloud-based robotics system has a controller in the cloud, which has got two main functionalities: virtualisation and computational offloading. I use the preceding concept in the system design.

The core of the system based on the *SOA* (*Service Oriented Architecture*), where the components of the system communicate over services. This concept results in a modular and flexible software architecture. So we can create the additive abstraction of the physical abilities of the robot devices.

The goal of this paper is to research and develop an adaptive, cooperative robotics system, which handles the abilities generically that facilitates optimal problem solving. *ROS* (*Robot Operating System*) and *Gazebo* can simulate the physical robot devices. They are used in numerous robotics development tasks. Furthermore, the system supports solving complex cooperative robotic tasks.

1. fejezet

Motiváció és célkitűzések

A felhő robotika feltörekvő kutatási terület napjainkban. Alapvetően a virtualizációt és a cella tehermentesítését írja elő a fejlesztés során. Fontos tulajdonsága, hogy a robotot vezérlő komponensek a felhőben foglalnak helyet. Ehhez persze szükségünk van egy modellre, amely megfelelően leírja a valós robot tulajdonságait és viselkedését. A modell segítségével szoftverből is hozzáférhetünk a robothoz, így kaphatunk információt róla valós időben és küldhetünk vezérlő parancsokat neki. Ezen képességek leírhatóak egymástól független szolgáltatásként. Ez az elgondolás lehetővé teszi a szolgáltatás-alapú architektúra alkalmazását a felhő robotikai rendszerben, amely rugalmas és skálázható felépítést eredményez, valamint elősegíti a robotok együttműködését mind egymással, mind pedig a felhasználóval.

A valós robotok fizikai korlátokkal rendelkeznek, melyek jelentős hatással vannak az általuk végzett munkára. Ilyen lehet például az akkumulátor kapacitása mobil robotok esetén, valamint a teherbíró képesség az anyagmozgató robotok faladatainál. Több robot együttműködése során szükséges ezen tulajdonságok összehangolása, így biztosítva a folytonos, gördülékeny és robosztus működést. Az előbb említett fizikai hiányosságok szoftveres úton kompenzálhatóak, azonban a megfelelő algoritmus kutatása komoly kihívást jelent.

A dolgozatban olyan felhő robotikai vezérlési struktúrát kutatok és fejlesztek, melynek célja a robotok együttműködésének támogatása, valamint a robosztus működés. A dolgozatban olyan keretrendszert definiálok, amely generikusan kezeli a robotok képességeit és adaptív a változásokkal szemben, ezzel elősegítve az optimális problémamegoldást.

A robotok által nyújtott szolgáltatások additívak, összegezve őket együttműködésre képesek, ha kompatibilisak egymással. Ebben az esetben egy rendszerként működnek, felsőbb szintről nézve egy szolgáltatásnak látszódik. Tehát például egy kar nem tud felemelni egy asztallapot biztonságosan, de kettő már igen. Ekkor a két kar külön-külön is egy szolgáltatást nyújt, amely valós és hasznos, viszont összeadva a kettőt válik igazán hasznossá a feladat szempontjából. Továbbá akár egy szenzorról is beszélhetünk (például hőmérséklet), amely önmagában is képes szolgáltatást nyújtani, azonban több szenzor fúziója hasznosabb funkcionalitással bír egyes feladatok végrehajtásában. Lehet fizikailag rögzített helyen, falra szerelve, de lehet akár a robot cellán. Utóbbi esetben több robot

cella jelenlétével csökkenthetjük a mérési bizonytalanságot, valamint a mobilitás is lehet szolgáltatás, ha egy helyszín hőmérséklete ismeretlen.

A mobil robotok esetén további kihívást jelent a vezetéknélküli közeg a felhő robotika szempontjából. A kapcsolat számos helyzetben lehet zajjal terhelt, ami megnöveli annak az esélyét, hogy megszakad a kommunikáció a felhő és a robot eszköz között. Erre megoldást jelenthet egy olyan vezérlési struktúra, amely kapcsolat nélküli üzemmódban is fenntartja a rendszer robosztusságát.

A dolgozat felépítése a következő szekvenciát követi: Először bemutatom a tervezés során használt technológiák elméleti hátterét és gyakorlati hasznukat. Az átfogó elméleti összefoglalót követően bemutatom az általam tervezett felhő robotikai rendszer felépítését, valamint példákon keresztül demonstrálom a működését. Ezután leírom az implementácó és a tesztelési fázis lépéseit. A rendszer eredményei alapján levonom a következtetéseket, majd ajánlást teszek további kutatási-fejlesztési feladatokra.

2. fejezet

Technológiai háttér

Ebben a fejezetben ismertetem a dolgozatban alkalmazott technológiák elméleti alapjait és ezek gyakorlati hasznát. Röviden bemutatom a felhő robotika alapgondolatát, valamint a szolgáltatás-központú architektúra elemeit. Összefoglalom a terület leggyakrabban használatos kommunikációs protokolljait is. Kitérek a leggyakrabban használt robotikai alkalmazásokra, szolgáltatásokra, valamint egy átfogó képet adok a *Robot Operating System* (*ROS*) és a *Gazebo* felépítéséről.

2.1. Felhő robotika

2.1.1. Felhő

A felhő alapú számítástechnika az elmúlt években világszerte ismert fogalommá vált. A legfontosabb funkciója, hogy a komplex számítási feladatokat a felhő végzi el. A *NIST* (*National Institute of Standards and Technology*) definíciója szerint a felhő alapú számítástechnika egy mindenütt jelenlévő, igény szerinti kényelmes hozzáférést jelent az elosztott számítási erőforrásokhoz [1]. Erre a célra három szolgáltatásmodellt határoztak meg:

- *SaaS* (*Software as a Service* - Szoftver, mint szolgáltatás)
- *PaaS* (*Platform as a Service* - Környezet, mint szolgáltatás)
- *IaaS* (*Infrastructure as a Service* - Infrastruktúra, mint szolgáltatás)

Ezen felül a *NIST* 4 tervezési modellt is definiál, melyek különböző tulajdonságokkal rendelkeznek kiterjedés és biztonság szempontjából:

- Privát felhő
- Közösségi felhő
- Publikus felhő
- Hibrid felhő

Az első két modell nagyon hasonlít egymásra, hiszen lokális, magán úton fenntartott felhőszolgáltatás jellemzi őket. A biztonsági követelmények könnyen teljesíthetőek, hiszen a külvilágtól elzárt, külső behatolásoktól védett a rendszer. A harmadik típus nyílt hozzáférést biztosít a felhő erőforrásaihoz és szolgáltatásaihoz. Ebben az esetben a biztonsági mechanizmusok sokkal komplexebbek. A negyedik tervezési modell pedig ötvözi az előbbi eseteket, tehát több privát felhőt összeköt publikus úton. Ez skálázhatóvá teszi a rendszert, valamint a biztonsági elvárások kielégítése és a szolgáltatások erőforrás-menedzsmentje is hatékonyabban működik.

A felhő alapú számítástechnika feltörekvő részterülete a felhő robotika, amely számos kutatás tárgyát képezi. Alkalmazza a mindenütt jelenlévő felhő előnyeit, mint a végtelennek tekinthető számítási kapacitás, a skálázható felépítés, valamint a hibatűrő, robosztus működés.

2.1.2. M2M és M2C kommunikáció

A közvetlen *M2M* kommunikáció nagy méretű csomópont-hálózatok esetén nehézkes, ami számos kihívást állít a fejlesztések elé. A felhő robotika megoldást jelenthet ezen problémák jelentős részére, hiszen egy mindenütt jelenlévő hálózati komponens jelenik meg a rendszerben, a felhő. Ennek következtében megjelenik a *M2C* (*Machine-to-Cloud* - Gép-felhő) kommunikáció, amelyre már készült implementáció, a *RoboEarth* nevű rendszer [2] [3]. Ez a tervezési modell egyszerűsíti a robotok együttműködését mind egymással, mind pedig a felhasználóval. Továbbá lehetővé teszi olyan szoftverek fejlesztését, melyek moduláris felépítésűek. Ennek következtében a szoftverkomponensek fizikai és földrajzi elhelyezkedése kötetlen, tehát akár elosztott virtuális gépeken is futhat, megvalósítástól függően. Erre a célra alkalmas lehet a feltörekvő *NFV* (*Network Function Virtualization* - Hálózati funkciók virtualizációja) technológia [4]. A szigorú hálózati követelmények azonban továbbra is megmaradnak, amely főleg a vezetéknélküli technológiákat érinti. Ehhez persze biztosítani kell a megfelelő hálózati kapacitást, amely lehetővé teszi a valós idejű kommunikációt a résztvevők között. Ennek megvalósítását igéri a szabványosítás alatt álló ötödik generációs mobil cellás technológia (*5G*), amely képes lesz minimalizálni a késleltetést, valamint többszörösére növelni az átviteli sebességet.

2.1.3. Vezetékes és vezetéknélküli megoldások

A felhő robotikai rendszer kulcsfontosságú komponense a hálózati egység. Különböző alkalmazási területek különböző technológiát igényelnek. A követelmények főbb paraméterei az adatátviteli sebesség, a késleltetés és a késleltetés-ingadozás, melyek a *QoS*-t meghatározzák. Az előbbi paraméterek nagymértékben függenek az elérhető sávszélességtől, valamint a közeget jellemző *SNR* (*Signal-to-Noise Ratio* - Jel-zaj viszony) értékétől. A közeget általánosan két részre oszthatjuk: vezetékes és vezetéknélküli. Rögzített helyű robotok esetén alkalmazható vezetékes kapcsolat, amely természetesen jobb paramétereket eredményez. Hátránya, hogy a robot mozgatásakor gondoskodni kell a kábeles ellátásról. Ezzel szemben a mobil robotok vezetéknélküli technológiák

segítségével kommunikálnak. A skála, amelyből választhatunk, itt szélesebb körű, mint vezetékes esetben. Az *ISM* sávokban helyet foglaló *Bluetooth*, *NRF24* és *ZigBee* alkalmas lehet szenzoradatok továbbítására. Ezek azonban viszonylag kicsi átviteli kapacitással rendelkeznek. Video továbbítására így a *WiFi* és az *LTE* hálózatok használatosak. Természetesen a vezetéknélküli megoldások érzékenyebbek a zajra, viszont nagy előnyük, hogy a hardver nincs helyhez kötve kommunikáció közben.

2.1.4. Koncepciók

A [5] cikk 3 típusú felhő robotikai koncepciót definiál:

- Peer-alapú modell
- Proxy-alapú modell
- Klón-alapú modell

A Peer-alapú rendszerek esetén minden robotra egy önálló számítási entitásként tekintünk, amely a fedélzetén végez el komplex számításokat, a felhő ekkor csak egy közvetítő közegként funkcionál. A Proxy-alapú modell egy hierarchikus, kétrétegű hálózati modell, ahol a robotok csoportja rendelkezik egy vezetővel, akivel minden más robot kommunikál. Ebben az esetben csak a vezető tartja a felhővel a kapcsolatot. A Klón-alapú modellben pedig minden robot egységről készül egy virtuális klón, melynek tulajdonságai megfelelő pontossággal közelítik a valós eszközét.

A [6] 3 általános szemléletet ír le egy robotikai rendszer vezérlési folyamatára.

- Autonóm, önálló gondolkodású, proaktív
- Reaktív
- Hibrid

2.1.5. Ember-gép interfész

A felhő robotika további fontos eleme a felhasználói interakció, elvégre a robotikai feladatok haszonélvezői az emberek. Az ember-gép interakció gyakori példája a teleoperáció, melynek során a felhasználó parancsát valós időben juttatjuk el a céleszközhöz, ami a lehető legkisebb késleltetéssel végrehajtja azt. Másik lehetőség, hogy a felhasználó csak felügyeli a folyamatot, valamint visszacsatolást kap a feladat sikerességéről. Ez a folyamat a robot eszközök autonóm működését kívánja.

2.2. Szolgáltatás-központú architektúra

A *SOA* (*Service Oriented Architecture* - szolgáltatás-központú architektúra) egy általános szoftverarchitektúra, melynek koncepciója szerint a rendszer moduláris felépítésű. A *SOA* alapon megvalósított rendszert modulok, komponensek alkotják, melyek önálló egységenként futnak. A szoftverkomponensek szolgáltatásokat biztosítanak az egymás

közötti, valamint más rendszerekkel való kommunikációra. Ez az architektúra skálázható, rugalmas és robusztus rendszerek felépítését teszi lehetővé. Továbbá a *SOA*-alapú szoftverek segítségével heterogén rendszerek is képesek egymással kommunikálni [7]. A *SOA* alapú rendszerek számos kommunikációs protokollt támogatnak, így többek között az *MQTT*, a *SOAP*, a *RESTful API* is alkalmazható a fejlesztés során. A szolgáltatások egymáshoz lazán csatoltak így abban az esetben, ha bármelyik fél meghibásodik, egy másik léphet a helyébe. Az együttműködő robotok vezérlésekor ez kiemelkedően hasznos funkció, hiszen a robotok mozoghatnak környezetükben. Tehát egyes eszközök megjelenhetnek, míg mások távoznak a rendszerből. Ez önszervező szolgáltatás-felépítést kíván maga után, melyet a *SOA* ugyancsak támogat. Egyes szolgáltatások használata igény esetén történik, valamint a komponensek újrahasznosítása is biztosított.

A [8] cikk szerzői leírják, hogy a *SOA* szerepe egyre inkább növekszik napjainkban, hiszen számos fejlesztői szervezet által elfogadott paradigmáról beszélünk. Ennek következtében a szerzők létrehoztak egy *SOA* alapú rendszert, valamint bevezették a *RaaS* (*Robot As A Service* - Robot, mint szolgáltatás) fogalmat. A [9] egy moduláris rendszert mutat be, ahol a felhőben lévő vezérlőegységet komponensekre bontják, majd bevezetik erre a *Robot Control As A Service* (Robotvezérlő, mint szolgáltatás) kifejezést. A rendszert virtuális környezetben és valós roboton is tesztelik. A [10] alkotói létrehoztak egy webes szolgáltatást tartalmazó alkalmazási réteget a *ROS* (*Robot Operating System*) fölé, ahol a *SOAP* segítségével kommunikáltak a vezérlővel.

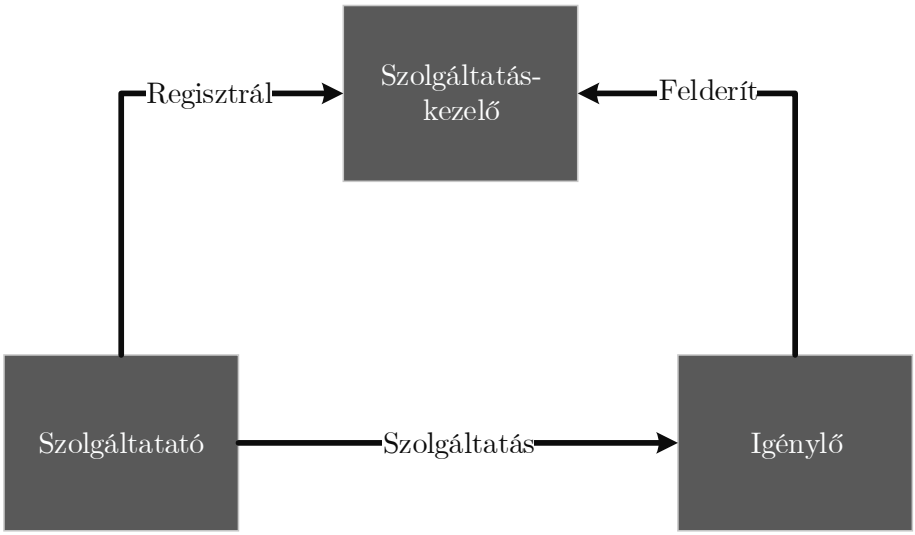
A *SOA* koncepciója szerint 3 fő szoftverkomponens létezik, melyet a fejlesztés során ajánlott alkalmazni:

- **Szolgáltatás-kezelő:** A szolgáltatásokat tárolja, egy központi adatbázisnak tekinthető
- **Szolgáltató:** A szolgáltatást nyújtja
- **Igénylő:** A szolgáltatást igényli és használja

A szolgáltatások igénybevétele az alábbi egyszerűsített módszert követi (2.1. ábra). A szolgáltatást először regisztrálni kell a szolgáltatás-kezelőnél, amely eltárolja a szolgáltatás adatai. Minden szolgáltatás rendelkezik egy előre definiált leírással, amely jellemzi a működését. Az igénylő ezután felderíti a szolgáltatást, melyre a szolgáltatás-kezelő válaszol a kért szolgáltatás tulajdonságaival. Ezt követően pedig kiépülhet a kapcsolat a szolgáltató és az igénylő között.

Az *Arrowhead* projekt kidolgozott egy rendszer-architektórát, amely alkalmazza a *SOA* koncepcióját [11]. Lokális felhőkre bontja a rendszert (*System-of-Systems* - Rendszerek rendszere architektúra), valamint számos biztonsági követelményt is támaszt, melyben az *AAA* (*Authentication*, *Authorization* and *Accounting* - Hitelesítés, jogosultságkezelés és naplózás). Az *Arrowhead* továbbá megoldást ajánl a *QoS* biztosítására a lokális felhőn belüli, valamint

A [12] egy konkrét példán mutatja be az általuk megvalósított *SOA*-alapú rendszert, ahol képadatok tömörítését végzik. A [13] bemutat egy rendszertervet, amiben létrehoztak



2.1. ábra. *A SOA alapvető mechanizmusa*

egy együttműködő szolgáltatási réteget.

2.3. Kommunikációs protokollok

2.3.1. REST API

A webes szolgáltatások egyik leggyakoribb megvalósítása *REST* (*Representation State Transfer*) *API* segítségével történik [14]. Az ilyen szolgáltatásokat *RESTful Service*-ként emlegeti a szakmai terminológia. A kommunikáció *HTTP* üzenetekkel zajlik, közülük a *GET*, *POST* és a *PUT* használatos a legtöbb esetben. A kommunikáció szerver-kliens arcshitektúrát követ, ahol a kliens kérést indít a szerver felé, majd ezt követően a szerver válaszol. A kliensnek specifikálnia kell az *URI*-t (*Uniform Resource Identifier*), amely az adott webes szolgáltatást azonosítja. A kommunikáció általában *JSON* vagy *XML* formátumú üzenetekkel történik, a szabvány ezt nem definiálja.

2.3.2. MQTT

Az *MQTT* (*Message Queue Telemetry Transport*) a *TCP/IP* rétegszerkezet tetején elhelyezkedő, alacsony erőforrás-igényű, üzenetküldésre alkalmas protokoll [15]. A kommunikáció modellje a *publish-subscribe*, ahol egy *broker* köti össze a klienseket a kommunikáció során. A kliensek *topic*-okra küldenek üzenetet, melyet csak azok kapnak meg, akik feliratkoztak a *topic*-ra. Az üzenetek formátumát a *REST API*-hoz hasonlóan itt sem határozza meg a szabvány. Alacsony erőforrás-igénye miatt gyakran alkalmazzák az *IoT* (*Internet of Things*) és *M2M* kommunikációban.

2.3.3. SOAP

A *SOAP* (*Simple Object Access Protocol*) az előző kettőhöz hasonlóan egy üzenetküldő protokoll [16]. A szabvány ebben az esetben szigorú előírást ad az üzenetformátumra, így kizárólag csak az *XML* a megengedett. A *SOAP* platformfüggetlen, általában *HTTP* vagy *SMTP* felett használják adatok továbbítására.

2.4. Robotikai szolgáltatások

A robotikai alkalmazások számos szolgáltatási lehetőséget nyújtanak. A leggyakrabban az alábbiak forulnak elő a gyakorlatban:

- Szenzorok: hőmérséklet, nyomásmérő, páratartalom-mérő, fénymérő, *LIDAR*, *Radar*, *Sonar*, tapintás-érzékelő, akkumulátorszint-mérő, kamera, odométer
- Beavatkozók: motorok, hidraulikus rendszerek, hőmérséklet szabályzók
- Magasabb szintű szolgáltatások: útvonal-tervezés, felderítés, *SLAM*, számítógépes látás, hangfeldolgozás, inverz kinematika

A hálózatra kötött szenzorokkal és az érzékelt adatokkal az *IoT* (*Internet of Things* - Tárgyak internete) [17] foglalkozik, ahol az adatok feldolgozása ugyancsak a felhőben zajlik. A felhő robotika ezzel analóg módon elhozza az adatokat a robotok fedélzeti szenzoraiból, majd a felhőben történő adatfeldolgozást követően döntést hoz, és megtörténik a visszacsatolás is. A visszacsatolás általánosan egy beavatkozó hardverelemmel történik, melyek valós-idejű mikrokontrollereket igényelnek közvetlenül a fedélzeten.

A felhő robotika szemlélete szerint a robot fedélzetén csak a hardverközeli vezérlés foglal helyet a legtöbb esetben. A magasabb szintű tervezési feladatok, melyek komplexebb számításokat igényelnek, a felhőben helyezkednek el. Az útvonal-tervező algoritmusok, a felderítés és a *SLAM* [18] a mobil robotok esetén használatos. A számítógépes látás [19] és hangfeldolgozás a humanoid robotok esetén gyakori. Az önvezető autók [20] fejlesztésekor az előző területek együtt szerepelnek. A robotkarok vezérlésekor pedig inverz kinematikai eljárásokat alkalmaznak.

A felhő alapú robotika könnyen lehetővé teszi a már említett együttműködő rendszerek létrehozását, amely hatékony és optimális feladatmegoldást eredményezhet. A robotok ekkor a felhőben történő vezérlés hatására kiegészítik egymás fizikai tulajdonságait az eredményesebb munkavégzés érdekében. Ilyen módon végezhető például egy terület felderítése, és a terület 3D modelljének elkészítése, melyben az együttműködésnek köszönhetően minden robotnak csak egy bizonyos részterületet kell lefednie, a végeredmény összeadódik. A trajektória-tervezéskor pedig az a cél, hogy a robotok ne üztközzenek egymásnak, amely a felhőben történő számításokkal ugyancsak egyszerűsödik.

2.5. Moduláris robotika

A moduláris robotika egyre inkább elterjed az ipari, felderítő, mentőszolgálati alkalmazásokban. Lényege, hogy a fizikai alkatrészek újrakonfigurálható építőelemekként

készülnek. Ennek következtében a robot mérete, alakja és más fizikai paraméterei is változtathatóak. Ez a tulajdonság nagy előnye a moduláris robotoknak az előbb említett alkalmazási területeken.

Moduláris robotokra létező példa a *SuperBot*, melyet a [21] cikk alkotói hoztak létre, valamint a *HEBI Robotics* [22] is gyárt moduláris robotokat.

2.6. Robot Operating System

A *Robot Operating System* (*ROS*) egy nyílt forráskódú szoftvercsomag, amely kifejezetten a robotikai szoftverek fejlesztésére összpontosít [23]. A rendszer felépítése moduláris, amely rugalmas fejlesztést tesz lehetővé, valamint a *SOA* alapú rendszerekbe való integrációt is egyszerűsíti. Széles körben elterjedt és erősen épít az együttműködő fejlesztők eredményeire. A rendszer legfőbb eleme a *roscore*, amely a rendszer alapvető funkcióiért felelős. A *ROS* számos előre elkészített modult tartalmaz, melyek között útvonal-tervező algoritmusok, számítógépes látást támogató szoftverek és robotkar koordinátorok is megtalálhatóak. A kész modulok mellé saját komponenseket (*node*) is készíthetünk, melyek implementációjára *C++* és *Python* programnyelveken van lehetőségünk. A *node*-ok önálló funkcióval rendelkeznek, valamint képesek kommunikálni egymással. A kommunikációnak két típusa van:

- **Publish-Subscribe:** a *node* egy *topic*ra küldhet üzenetet, melyet azok a *node*-ok kapnak meg, akik feliratkoztak rá
- **Szolgáltatások:** szerver-kliens architektúra, ahol a kliens kér egy szolgáltatást, majd a szerver válaszol rá.

A fejlesztés során továbbá a parancssort is alkalmazhatjuk a modulokkal való kommunikációra, amely esetén a fejlesztő közvetlenül tájékozódhat a rendszer aktuális állapotáról. A kommunikáció előre definiált üzenetstruktúrákkal zajlik.

A *ROS* egyik legfontosabb modulja a *tf*, vagy más néven *transform*, amely egyszerűen hozzáférhető interfészt biztosít térbeli koordináta-transzformációk végrehajtására. A *tf* felépítése hierarchikus, ahol az úgynevezett frame-ek leszármaznak egymásból. A fa gyökere a *world_frame*, amelyből minden más frame leszármazik. A *tf* képes ezek között a *frame*-ek között oda-vissza transzformálni.

2.7. Gazebo

A *Gazebo* egy önállóan is működőképes, valamint a *ROS*-sal könnyen integrálható 3D-s grafikus környezet [24]. Az *OGRE* nevű 3D-s motor hajtja a rendkívül erőforrás-igényes szoftvert, melyet főként robotikai szimulációk végrehajtására fejlesztettek. Számos előre definiált objektumot tartalmaz: robotokat és környezeti elemeket egyaránt. Saját leíró nyelvének segítségével - amely leginkább az *XML*-re hasonlít - egyedi objektumok is készíthetőek. A szoftver erőssége, hogy képes a valósághoz nagyon közeli fizikai környezetet

szimulálni. Ennek köszönhetően a virtuális szenzorok működése során az egyenletes spektrális eloszlású Gauss zajt is figyelembe veszi.

Jelen fejezetben röviden összefoglaltam azon technológiák jellemzőit és működését, melyek egy felhő robotikai keretrendszer alapköveit adják. Átfogó képet kaptunk szolgáltatás-központú architektúráról, valamint a kommunikációs protokollokról. Továbbá ismertettem néhány példát a robotikai rendszerek szolgáltatásairól, valamint bemutattam a *ROS* és a *Gazebo* jellemzőit. A következő fejezetekben kiderül, hogyan lehet az említett fogalmak alapján megtervezni egy ilyen rendszert.

3. fejezet

Rendszer specifikáció

Ebben a fejezetben bemutatom a tervezett felhő robotikai rendszer felépítését, valamint konkrét példákkal demonstrálom az elvárt működést. Kiemelt részletességgel írok a rendszer működésének alapját képező szolgáltatásokról és a rajtuk végezhető műveletekről. Továbbá leírom a teszt eseteket, melyeket a komponensekkel végeztem. Végül pedig ajánlást adok néhány alapvető hibakezelő megoldásra.

3.1. A rendszer terve

A rendszer tervezésekor a *SOA* (*Service Oriented Architecture*) koncepcióját alkalmaztam. A *SOA*-alapú szoftver moduláris felépítésű, ahol a szoftverkomponensek szolgáltatásokat biztosítanak az egymás közötti, valamint más rendszerekkel való kommunikációra. Ez az architektúra skálázható, rugalmas és robusztus rendszerek felépítését teszi lehetővé. Továbbá a *SOA*-alapú szoftverek segítségével heterogén rendszerek is képesek egymással kommunikálni. Ezen szolgáltatások egymáshoz lazán csatoltak, tehát egy szolgáltatás később is csatlakozhat a rendszerhez. A rendszer terve a 3.1. ábrán látható.

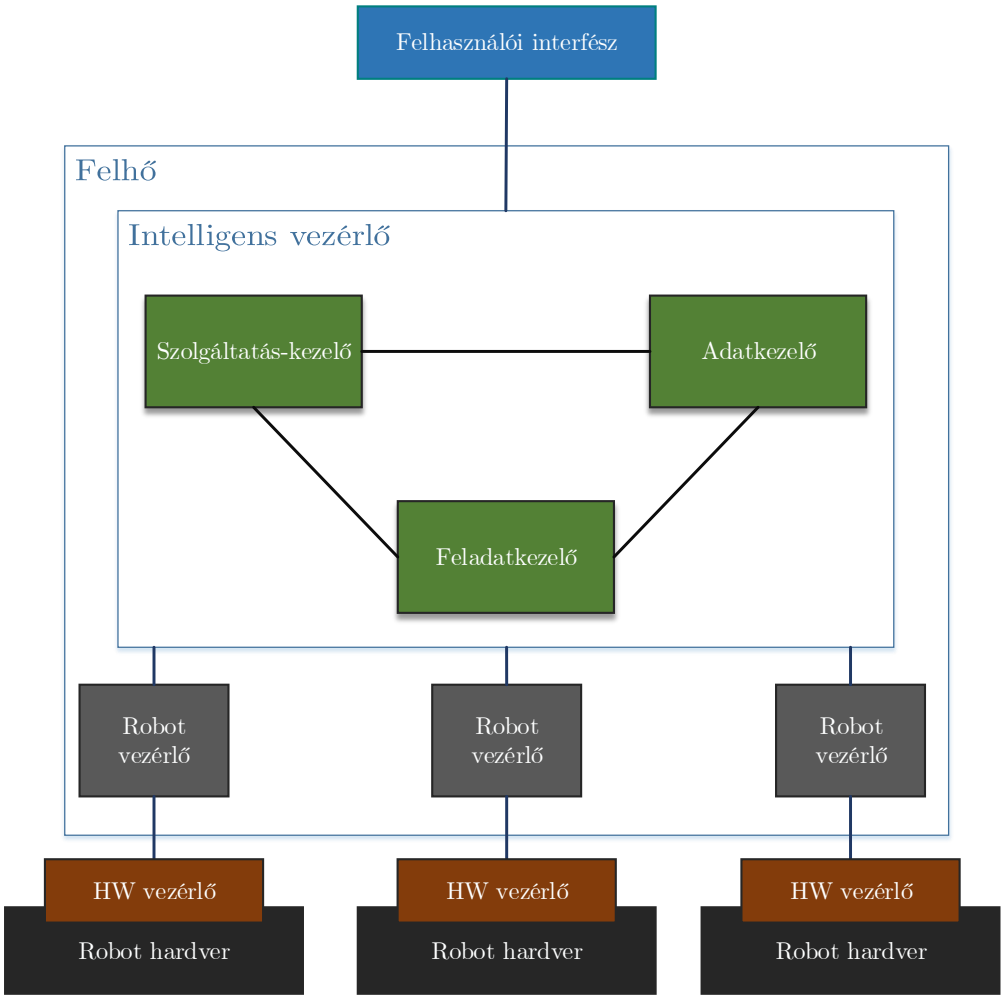
Ahogy a *SOA* elméleti alapjaiban korábban leírtam (2.2 fejezet), a legfőbb komponensek a Szolgáltatás-kezelő, a Szolgáltató és az Igénylő. A rendszer makroszkópikus szemléletében a Szolgáltató megfelel a Robotnak, az Igénylő pedig a felhasználó ebben az esetben. A Robot tehát képes szolgáltatásokat nyújtani, melyet a Szolgáltatás-kezelő modulban regisztrálhat. A felhasználó pedig felderítheti ezen szolgáltatásokat, majd igénybe veheti azokat.

A rendszer architektúrájában szereplő modulok tehát az alábbiak:

Felhasználói interfész: A rendszer és a felhasználó között létesít kapcsolatot. Ez lehet akár egy parancssori alkalmazás, vagy egy grafikus felület.

Felhő: Egy összefoglaló név azon komponensek halmazára, melyek a felhőben helyezkednek el. Ezen modulok fizikailag kötetlen helyen vannak, valamint hálózati technológiák segítségével elérhetőek.

Intelligens vezérlő: Ez a modul tekinthető a rendszer „agyának”. Feldolgozza a felhasználó által küldött kéréseket, valamint értelmezi a komplex feladatokat. A feladatokat részekre bontja, megtervezi ezek körülményeit, majd ezt végrehajtatja a Feladatkezelővel.



3.1. ábra. *A rendszer terve*

Feladatkezelő: A feladatok konkrét részét hajtja végre. Parancsokat ad a Robot vezérlő modulnak, valamint visszacsatolást kap a Robot állapotáról.

Szolgáltatás-kezelő: A szolgáltatások regisztrációját és egyesítését kezeli, valamint biztosítja a tárolt szolgáltatások állandó elérhetőségét.

Adatkezelő: Adatjegyző szoftverkomponens, amely szenzoradatokat tárol. Erre hivatkozhat az Intelligens vezérlő a feladat tervezésekor.

Robot vezérlő: A Robotot vezérlő magas szintű szoftverkomponens. Ez a modul képes szolgáltatásokat nyújtani. Szolgáltatások lehetnek a szenzorok aktuális adata, vagy a beavatkozók állapotának módosítása.

Robot hardver: A felhő robotikai rendszer hardvereleme, amely lehet szimulált, vagy valós eszköz.

Hardver (HW) vezérlő: A Robot hardver fedélzetén lévő alacsony szintű vezérlőegység (például mikrokontroller).

3.2. Szolgáltatás-kezelő

A Szolgáltatás-kezelő általános feladata a szolgáltatások és ezek tulajdonságainak tárolása. A rendszer szempontjából egy központi adatbázisnak tekinthető. A Szolgáltatás-kezelő elérhetőségét minden komponens ismeri, ellentétben például egy robot esetével, ahol dinamikus szolgáltatásról beszélünk, amit a többi modul csak a Szolgáltatás-kezelőn keresztül ismerhet meg.

A szolgáltatások a rendszer indulását követően bármely időpontban csatlakozhatnak a rendszerhez. Új szolgáltatás megjelenésekor regisztráció szükséges, melynek során a Szolgáltató eljuttat minden szükséges információt a Szolgáltatás-kezelőhöz, amely egy előre definiált struktúrát követ, benne az alábbi elemekkel:

- Név
- Típus
- Hálózati cím
- Szolgáltatásréteg
- Tulajdonságok
- URI

A szolgáltatások lokális címkiosztásáért egy saját fejlesztésű modul felel. A szolgáltatást általánosan a neve azonosítja, melyhez egy típus is társul. A szolgáltatások felderítése során erre is hivatkozhat az Igénylő modul, amennyiben ez bizonyos fajtájú szolgáltatást keres. Továbbításra kerül a szolgáltatásréteg (SL) száma, melyet a 3.5 fejezetben részletesen ismertetek. A szolgáltatások $REST\ API$-n keresztül hívhatóak, a 2.3.1 fejezetben tárgyalt módszerrel.

3.2.1. Teszt eset 1: Szolgáltatás regisztrációja

A teszt során egy UGV robot szolgáltatatását volt célom regisztrálni a Szolgáltatás-kezelő modulban. Első lépésben elindítottam az Intelligens vezérlő Szolgáltatás-kezelő modulját. Minden robot rendelkezik egy Felhő-beli vezérlővel, ez jelen esetben a Robot vezérlő komponens. A következő lépésben Robot vezérlő modult indítottam, amely regisztrációs kérelmet küldött a Szolgáltatás-kezelőnek, az alábbi elemekkel:

- Név: Robot 1
- Típus: UGV robot
- Hálózati cím: 192.168.0.20:9001
- Szolgáltatásréteg: SL1
- Tulajdonságok: sugár: 0.2m; maximális sebesség: 0.5 m/s

- URI: /move; /getOdometer

A szolgáltatás neve tehát: Robot 1, típusa pedig: *UGV*. Az *UGV* robot kétdimenzióban képes elmozdulni, ilyen például egy robotporszívó. A regisztrációs üzenetben megtalálhatóak továbbá a szolgáltatásra jellemző *URI*-k. A „/move" például egy mozgatóparancsot küld a robotnak, a „/getOdometer" pedig kinyeri az odométer adatait. A Szolgáltatás-kezelő jóváhagyta a regisztrációs kérelmet és tárolta a szolgáltatást. A folyamat végeztével a szolgáltatás felderíthetővé vált.

3.3. Feladatkezelő

Felhasználói kérés esetén az Intelligens vezérlő részekre bontja az adott komplex feladatot. A feladat egy szálát a Feladatkezelő hajtja végre. A folyamat az alábbi szekvenciát követi:

1. Kérés érkezik a feladat végrehajtására a Intelligens vezérlőhöz, aki részekre bontja azt.

2. Az Intelligens vezérlő kiválasztja a megfelelő feladattípust, megszabja a résztvevő robotok számát, majd továbbítja a kérést a Feladatkezelő felé.

3. A Feladatkezelő ekkor listázza a szükséges szolgáltatásokat, valamint lekéri ezek állapotát a Szolgáltatás-kezelő modultól. Előfordulhat, hogy az adott szolgáltatásból több példány van, azonban ebből csak a szükséges mennyiséget választja ki. A kiválasztás során a feladatvégzés szempontjából optimális példányokat választ, melynek elérése érdekében az adott feladatra releváns tulajdonságokat vizsgálja.

4. Ezt követően elvégzi a szolgáltatások addícióját, kiterjesztését, melyet a 3.5 fejezetben részletezek.

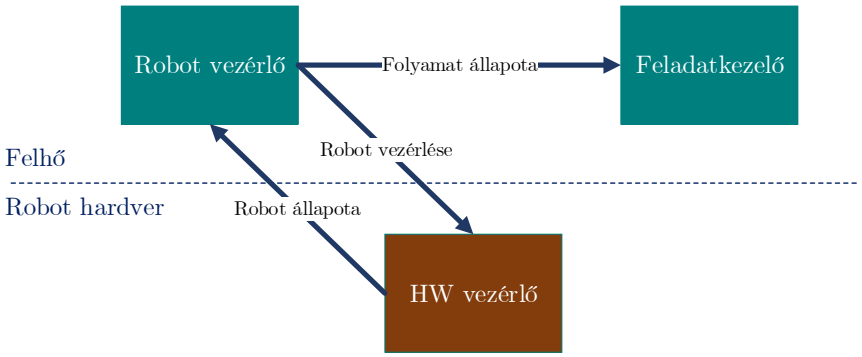
5. A Feladatkezelő kiadja a konkrét parancsot a Robot vezérlő részére.

6. Majd a konkrét feladat végrehajtásra kerül. A 3.2. ábra mutatja, ahogy Robot vezérlő modul kommunikál a Robot fedélzeti hardver vezérlőjével, valamint a Feladatkezelő modult is tájékoztatja a folyamat állapotáról.

7. Az Intelligens vezérlő kiértékeli a feladat eredményeit.

3.3.1. Teszt eset 2: Egyszerű feladat végrehajtása

Ezen teszt célja, hogy egy robot menjen egyik pontból a másikba. Az egyszerűség kedvéért folytatom az előző példát (3.2.1 fejezet), ahol a Robot vezérlő regisztrálta magát a Szolgáltatás-kezelő modulban, mint Szolgáltató. A Robot hardverét a *ROS* és a *Gazebo* segítségével szimuláltam virtuális környezetben. A szimulált hardvert elhelyeztem egy minimális 3D-s környezetben, ahol csak egy világítást biztosító modell, és a padló modellje szerepel a roboton kívül. A robot kezdeti koordinátája ($x = 0m, y = 0m$), a cél pedig eljutni az ($x = 1m, y = 1m$) pozícióba. A folyamat az alábbiak szerint történik: A felhasználó



3.2. ábra. *A Feladatkezelő modul működése*

kérést indít az Intelligens vezérlőhöz, hogy juttassa el a robotot a kívánt koordinátára. A feladat ebben az esetben még nem túl összetett, az Intelligens vezérlő így egy konkrét feladatra ad kérést a Feladatkezelőhöz. A Feladatkezelő felderíti a Robot vezérlő által regisztrált szolgáltatást, ezen keresztül adhat parancsokat a robotnak, valamint kaphat valós idejű szenzoradatokat. Az egyszerűség kedvéért a robot pozíciójának meghatározását a *Gazebo* végzi. A Feladatkezelő így a valós idejű visszacsatolt adatok segítségével a kívánt pozícióba juttatja a robotot. A feladat végeztével az Intelligens vezérlő tárolja a feladat költségeit, paramétereit későbbi felhasználás céljából.

3.4. Szolgáltatások leképezése

A különböző hardverek modellje és a szoftveres szolgáltatások leképezhetőek egy absztrakciós szintre, ami a rendszer szempontjából egyértelmű leírással rendelkezik. A leképezést követően a rendszerben szereplő hardverelemek rendelkeznek egy szoftvermodellel, amely szolgáltatásokon keresztül biztosít hozzáférést az eszközhöz. A rendszer tervében szereplő Robot vezérlő modul egy ilyen egységnek tekinthető.

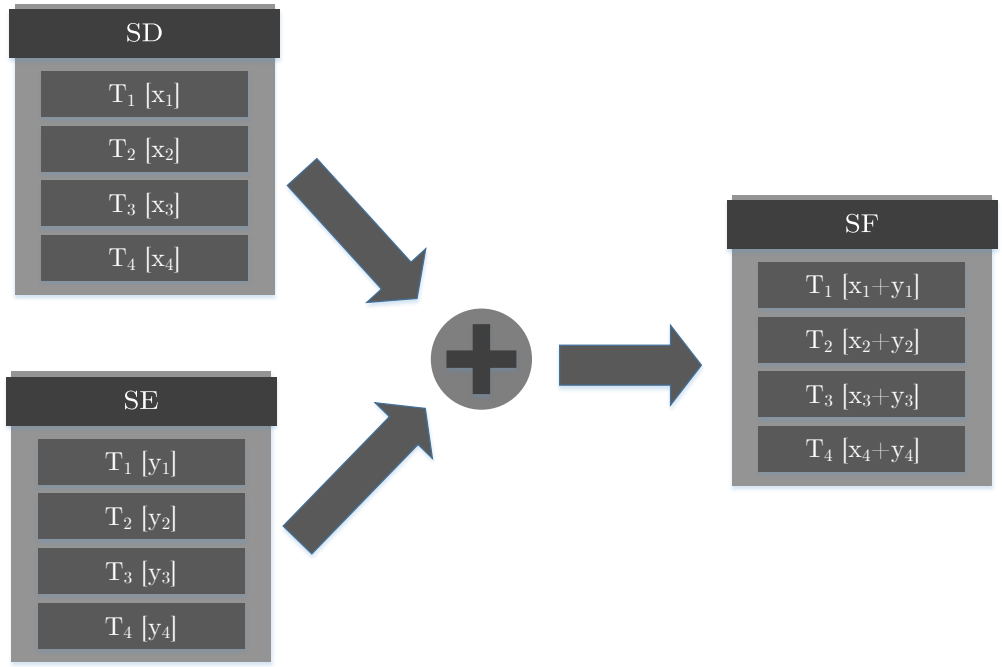
A leképezett szolgáltatások teszik lehetővé az additív tulajdonságok megjelenését, melyek a rendszer alapját jelentik. Ezen tulajdonságok definiálása a Szolgáltatás-kezelő modul feladata, amely generikus típusokat rendel az adott szolgáltatáshoz. Egyes feladatok végrehajtásához szükség van több szolgáltatás egyesítésére, amelyre két lehetőséget szolgáltat a rendszer:

- Additívitás
- Kiterjesztés

3.4.1. Szolgáltatás additivitása

A szolgáltatások additivitása azt jelenti, hogy a rendszer minden egyes tulajdonságot külön-külön összegez. Ekkor egy vagy több új szolgáltatás jön létre, amely örökli az ős tulajdonságait. A szolgáltatások addícióját a 3.3. ábrázolja. A folyamatban három szolgáltatás szerepel (SA, SB, SC), melyek mindegyike k darab tulajdonsággal rendelkezik

(jelen példában $k = 4$). A tulajdonságok paraméterei a 3.1 egyenlet szerint adódnak össze, ahol T_i^s maga a tulajdonság értéke, $i \in [1, k], i \in \mathbb{N}$ és $s \in \{SA, SB, SC\}$. A művelet során az SA és az SB szolgáltatások addíciójaként jön létre SC szolgáltatás.



3.3. ábra. *A szolgáltatások additivitása*

$$\begin{aligned} T_1^{SC} &= T_1^{SA} + T_1^{SB} \\ T_2^{SC} &= T_2^{SA} + T_2^{SB} \\ T_3^{SC} &= T_3^{SA} + T_3^{SB} \\ T_4^{SC} &= T_4^{SA} + T_4^{SB} \end{aligned} \tag{3.1}$$

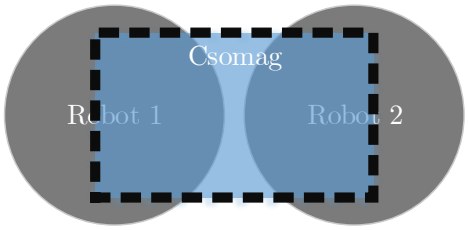
A művelet tulajdonsága, hogy $SA + SA = SA$, tehát ha egy szolgáltatáson önmagával végezzük el, az eredeti szolgáltatást kapjuk eredményként.

3.4.2. Teszt eset 3: Rakomány szállítása szolgáltatás-addícióval

A 2. teszt eset (3.3.1 fejezet) folytatásaként hozzáadtam egy másik robotot a *Gazebo* szimulációs környezethez, melyet közvetlenül az első mellé helyeztem el (3.4. ábra).

A teszt során szállítani szeretnék egy M tömegű csomagot az $(x = 0m, y = 0m)$ pontból az $(x = 0m, y = 1m)$ pontba. A feladat végrehajtására a 2 robot áll rendelkezésre, ahol az első robot m_1 tömeget bír el, a második pedig m_2-t. Tegyük fel, hogy $M > m_1, m_2$ és $M \leq m_1 + m_2$. Ekkor M tömegű tárgy szállításához mindkettő robot szükséges, így képesek lesznek eljuttatni az adott objektumot a célponthoz.

A folyamat kezdetén az felhasználó kérést indít az Intelligens vezérlőhöz, hogy el szeretné juttatni az adott szállítmányt a $(x = 0m, y = 1m)$ pontba. Az Intelligens vezérlő felderíti az elérhető Robot vezérlő modulok szolgáltatását a Szolgáltatás-kezelő modulnál. Az



3.4. ábra. *A csomag elhelyezkedése az UGV robotokon*

Intelligens vezérlő két robot tulajdonságai és a csomag tömege alapján tudja, hogy csak ketten bírják el az adott csomagot. Ennek következtében részekre bontja a feladatot, valamint igényli a szolgáltatások addícióját a Szolgáltatás-kezelő modultól. A két robot szolgáltatásának addíciója ekkor úgy történik, hogy a maximális teherbíró-képességek, mint tulajdonságok összeadódnak. A két robot az addíciónak köszönhetően végre tudja hajtani a feladatot.

A teszt során egy egyszerű példát mutattam be, ahol két UGV szállít egy tárgyat. A szállítmány rögzítése történhet dinamikusan, amely esetben a rakomány rögzítési pontjai szabadon elfordulhatnak. Ekkor a útvonal tervező algoritmus nagyobb szabadsággal bír. Azonban előfordulhat olyan eset is, amikor a két rögzítési pont fix, melynek következtében egy merev testként mozognak együtt.

Az új szolgáltatást a Szolgáltatás-kezelő modul hozza létre igény esetén, majd eltárolja ezt, így az Intelligens vezérlő a későbbi feladatok tervezése során is igénybe veheti. Ennek persze feltétele, hogy az adott hardver és szoftver is működőképes legyen a későbbiekben is, ezért a Szolgáltatás-kezelő modul felelős, hiszen periodikusan ellenőrzi a regisztrált szolgáltatások elérhetőségét.

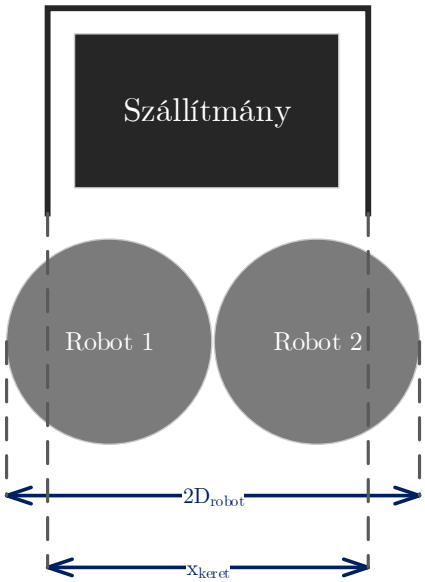
3.4.3. Teszt eset 4: Szolgáltatás-egyesítés esetén jelentkező anomáliák

A szolgáltatás-addíció pozitív hatásai mellett lesznek hátrányos következmények. Az előző példát figyelembe véve a két robot maximális teherbíró-képessége ugyan megnőtt, azonban a fizikai kiterjedés is összeadódik. Így előfordulhat, hogy nem férnek el egymás mellett a környezet adta akadályok miatt (3.5. ábra), amely növeli a tervezés komplexitását. Az ábrán a D_{robot} jelöli egy robot átmérőjét, az x_{keret} pedig a környezet adta akadály „ajtajának" szélességét.

Az Intelligens vezérlő a környező elemek pozíciójának tudatában ezt is figyelembe veszi a feladat tervezése során. Megoldás lehet például a két robot megfelelő pozícionálása egymáshoz viszonyítva.

3.4.4. Teszt eset 5: Rakomány szállítása szolgáltatás-addícióval az energia becslésével kiegészítve

A teszt célja ebben az esetben is, hogy UGV robotok segítségével kell eljuttatni egy adott tárgyat egyik pontból a másikba, melynek tömege $m_t = 10kg$. Két típusú UGV-vel



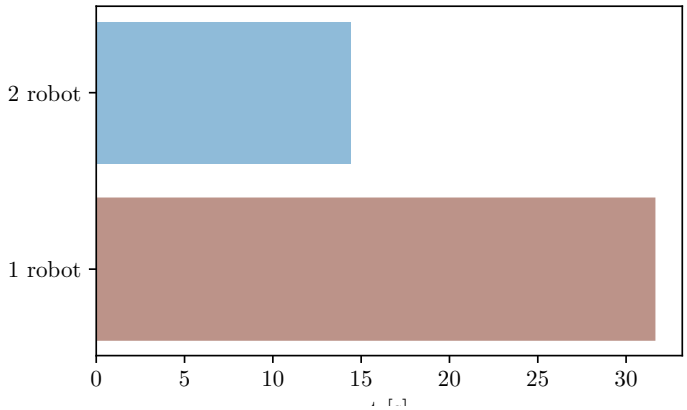
3.5. ábra. *A szolgáltatás-egyesítés velejárói*

hajtottam végre a tesztet (3.1.. táblázat). A teszt összeállítása a 3.4.2 fejezetben található 3.teszt esethez hasonló, annyi kiegészítéssel, hogy a robotok energiaszintjét is becslem. Az első típusú robotok szolgáltatását egyesítette a rendszer, ennek következtében a maximális teherbírása összeadódik, viszont a sebesség és az energiaszint küzül a minimumot kell venni, hiszen abban az esetben, ha az egyik akkumulátor lemerül, a robot nem lesz képes mozogni. A két robot fizikai kiterjedése is összeadódik továbbá. Fontos, hogy a robotok a feledat megkezdése előtt felvegyék a megfelelő alakzatot, hogy a szolgáltatások addícióját követően hatékony legyen a munkavégzés. A második típusú robot erősebb fizikai paraméterekkel rendelkezik, viszont jóval távolabb van a célponttól.

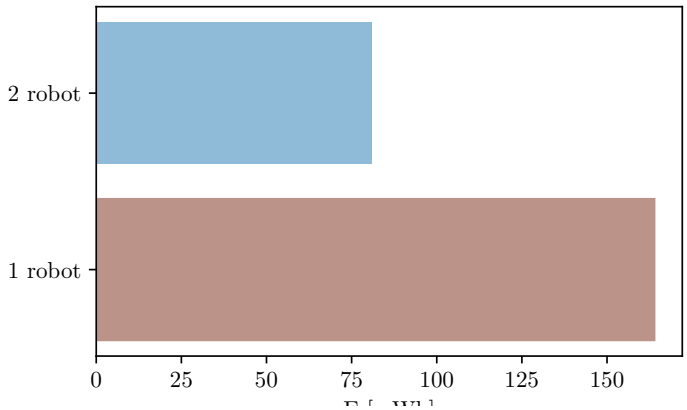
	Típus 1	**Típus 2**
Maximális teherbíró-képesség	$6\,kg$	$10\,kg$
Akkumulátor kapacitása	$30\,Ah$	$60\,Ah$
Motor feszültség	$12\,V$	$12\,V$
Motor áramfelvétele	$10\,A$	$10\,A$
Maximális sebesség	$0.5\,\frac{m}{s}$	$1\,\frac{m}{s}$
Távolság a célpontól	$2\,m$	$8\,m$
Példányszám	2	1

3.1. táblázat. *A tesztben szereplő UGV robotok paraméterei*

Készítettem továbbá egy egyszerűsített akkumulátor-szimulátort, melynek célja, hogy becsülje egy általános akkumulátor töltöttségi szintjét, adott fizikai paraméterek alapján. A szimulátor figyelembe veszi az időben konstans fogyasztást, amely a fedélzeti vezérlő hardver és a szenzorok fogyasztásából ered. Valamint a motor energiahasználatát is modellezi, amely időben változó értékeket ad. A szimulátor szolgáltatásként jelenik meg a rendszerben, melyet hozzá lehet rendelni egy robot eszközhöz. A folyamat eleján meg kell



3.6. ábra. *A teszt időtartama*



3.7. ábra. *A teszt során felhasznált energia*

adni a fizikai paramétereket (feszültség - U, maximális áram - I, töltéskapacitás - Q_{max}, lineáris fogyasztás), ami alapján szimulálja az energiahasználatot. Az energiahasználatot a $P = U \cdot I$, $\Delta E = P \cdot \Delta t$ és a $\Delta Q = \frac{\Delta E}{U}$ egyenletek alapján becslem.

A teszt folyamán összehasonlítottam a szolgáltatás-egyesített robotokat és egy önállóan mozgó robotot. Célom az volt, hogy megvizsgáljam a szolgáltatás-egyesítés hatását egy konkrét, mindennapi helyzetből vett példára. A vizsgáltatot két paraméter alapján végeztem: az idő és az felhasznált energiamennyiség. A második típusú robot tesztjének eredményei a 3.6. és a 3.7. ábrákon láthatóak.

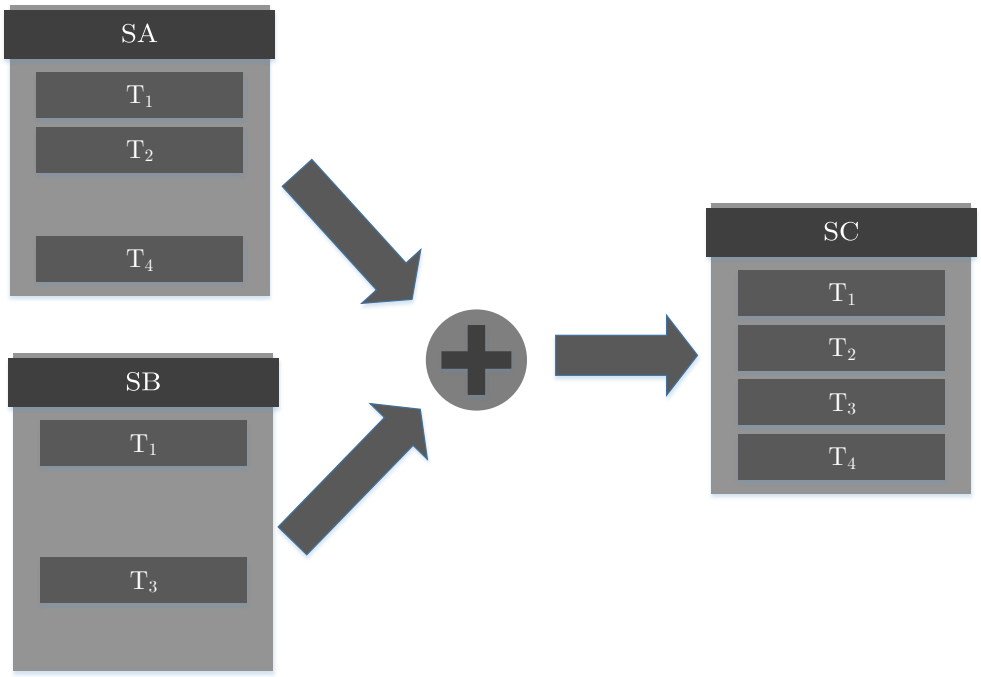
A robotok motorjának elektromos tulajdonságai mindkét esetben megegyeznek, így az egy időegységre eső fogyasztás is valószínűleg közel egyenlő lesz. Az 1. típusú robotokból viszont két példány szerepel a szimulációban, melyek fele akkora sebességgel mozognak, mint a 2. típusú. Az akkumulátor kapacitása a 2. típusú robot esetén kétszer akkora, mint az 1. típusú robotnál, ami a munkavégzési képesség maximális időtartamára hatással van. A célponttól vett távolság a 2. típusú robotnál négyszer akkora, mint az 1. roboté, így az enegia használat és az idő is ezzel arányosan növekszik.

Az eredmények azt mutatják, hogy az 2. típusú robot esetén a feladat elvégzésének időtartama és az energiafogyasztás több, mint kétszer akkora, mint az 1. típus esetén.

Ez megfelelel az elvárásoknak, valamint a rendszer szolgáltatás-egyesítő szemlélete szempontjából validációs jelentőséggel bír.

3.4.5. Szolgáltatás kiterjesztése

A szolgáltatás kiterjesztése esetén pedig olyan tulajdonságok halmazával bővítünk, amely eddig nem volt része az alap szolgáltatásnak. A szolgáltatások kiterjesztése a 3.8. ábrán látható, valamint a 3.2 egyenletrendszer alapján történik. Az addícióhoz hasonlóan a folyamatban három szolgáltatás szerepel (SD, SE, SF), a tulajdonsággal száma itt is $k = 4$. T_i^s maga a tulajdonság értéke, $i \in [1, k], i \in \mathbb{N}$ és $s \in \{SD, SE, SF\}$. A művelet során az SD és az SE szolgáltatások kiterjesztéseként jön létre SF szolgáltatás.

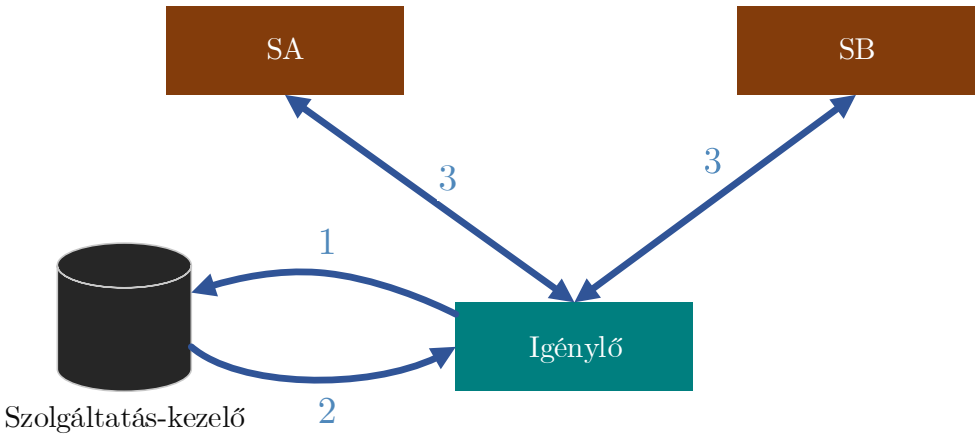


3.8. ábra. *A szolgáltatások kiterjesztése*

$$\begin{aligned} T_1^{SF} &= T_1^{SD} + T_1^{SE} \\ T_2^{SF} &= T_2^{SD} \\ T_3^{SF} &= T_3^{SE} \\ T_4^{SF} &= T_4^{SD} \end{aligned} \tag{3.2}$$

3.4.6. Teszt eset 6: Szinkronizált mozgás

a. eset: A gyakorlatban számos olyan példával találkozhatunk, amely esetén két robotnak szinkronizáltan kell mozogniuk. Ez a feladat megoldható a robotok szolgáltatásainak kiterjesztésével. A mérés során 3. teszt esetben (3.4.2 alfejezet) használt összeállításban szereplő 2 robotot alkalmazom. Az egyik robotot az ($x = 0m, y = 0m$), a másikat az ($x = 0m, y = 4m$) kezdeti pozíciókban helyeztem el. A feladat célja, hogy a két robot



3.9. ábra. *A szolgáltatások hívása hagyományos (1.) esetben*

szinkronizáltan eljusson az $(x = 10m, y = 2m)$ koordinátájú találkozási pontra.

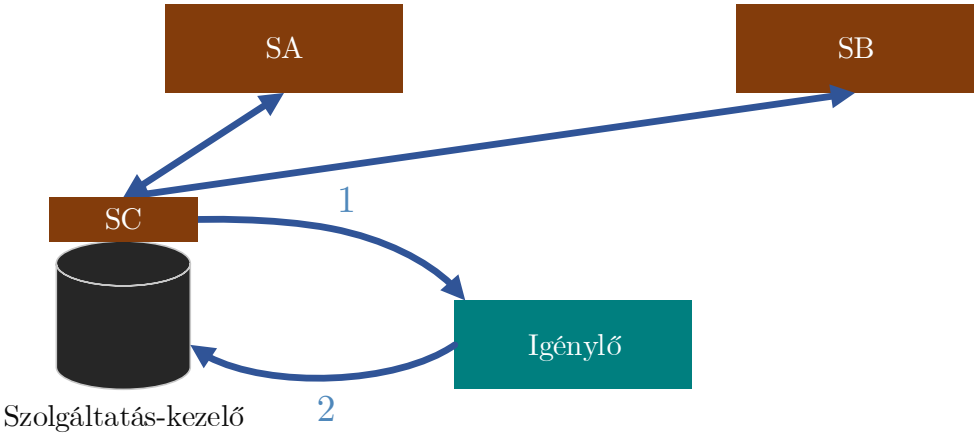
A folyamat ebben az esetben is a felhasználó parancsára indul, amikor az Intelligens vezérlő felméri a körülményeket és a tervezés során kérvényezi a két Robot vezérlő szolgáltatásainak kiterjesztését. A Szolgáltatás-kezelő ekkor a robotok pozíciójára vonatkozó tulajdonságok alapján kiterjeszti a szolgáltatásokat. A generált szolgáltatás a Feladatkezelő parancsára az adott pontba navigálja a szimulált robotok modelljét, melyek megvárják egymást az elvárt működés szerint.

b. eset: A szolgáltatás kiterjesztésére egy további egyszerű eset, mikor két robotkarnak kell felemelni egy tárgyat, majd megfordítani azt, és visszahelyezni a kiindulási helyére. Ekkor a Szolgáltatás-kezelő generál egy szolgáltatást, ami csakis azért felel, hogy a két robot helyzete szinkronban maradjon. A két robot ugyanolyan erővel kell, hogy megfogja az objektumot, viszont az időbeli szinkron ennél is fontosabb. Biztosítani kell, hogy minden időpillanatban legalább egy robot tartja a tárgyat.

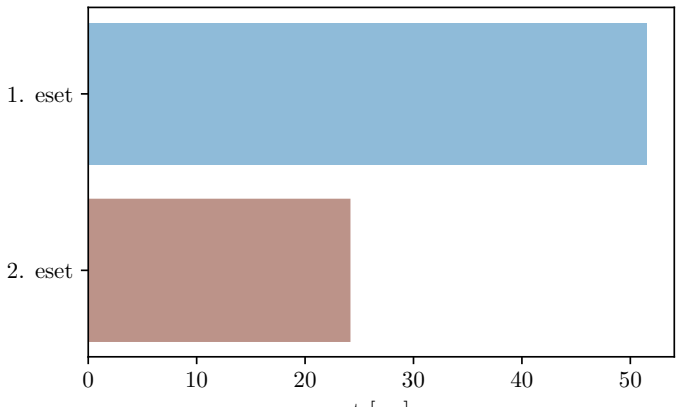
3.4.7. Teszt eset 7: Egyesített szolgáltatás igénybevétele a hagyományossal szemben

A szolgáltatás egyesítést teljesítmény szempontjából is teszteltem. A mérés első fázisában a két szolgáltatást külön-külön hívtam meg, majd tároltam az eredményét. Második lépésben pedig elvégeztem a szolgáltatás kiterjesztését, így létrejött egy harmadik, absztrakt szolgáltatás, amely örökölte a két ős-szolgáltatás tulajdonságait. A mérés eredményét a 3.11. grafikon mutatja.

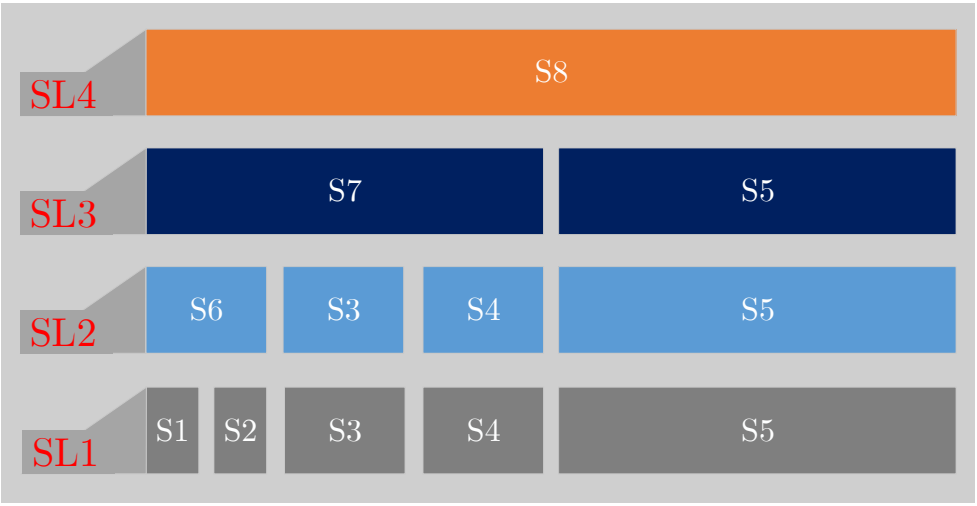
Az elvárásaim szerint abban az esetben, ha egyesével hívjuk meg a szolgáltatásokat, több időt vesz majd igénybe, mint az egyesített szolgáltatás igénybe vétele. Ez a teszt analóg azzal a folyamattal, ahol a szekvenciális és a párhuzamos feldolgozást hasonlítjuk össze. A párhuzamosítás mindenképpen a teljesítmény javulását okozza a folyamat bizonyos szálú felbontásáig. A feldolgozás gyorsulása a szálak számának függvényében viszont nem feltétlenül lesz lineáris.



3.10. ábra. *A szolgáltatások hívása egyesített esetben (2.) esetben*



3.11. ábra. *A hagyományos (1.) és egyesített (2.) szolgáltatások hívásának időtartama*



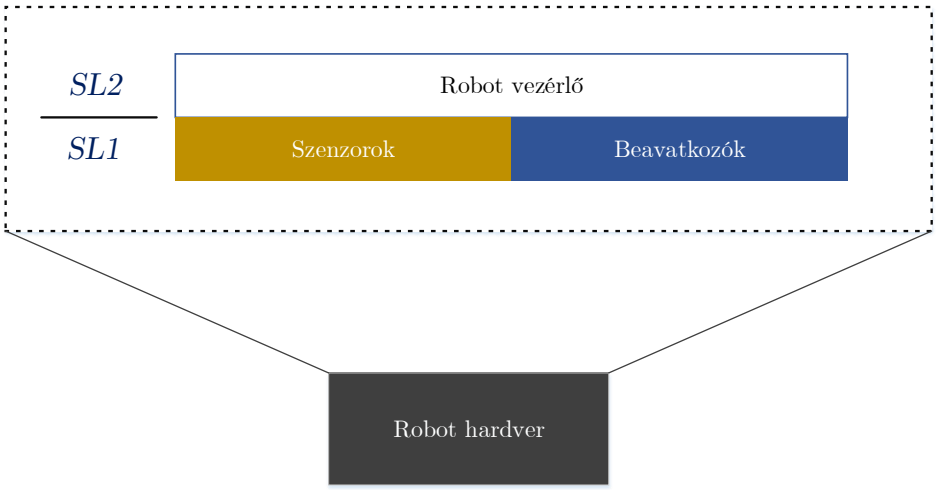
3.12. ábra. *Szolgáltatásrétegek*

Az teszt eredményei azt mutatják, hogy az egyesített szolgáltatás hívása kevesebb, mint 25 ms, a hagyományos pedig több, mint 50 ms időt vett igénybe. Ez tehát több, mint kétszeres teljesítményjavulást jelent a működés szempontjából. A szogáltatás-egyesítés tehát a folyamatok gyorsítására is alkalmas.

3.5. Generikus szolgáltatásrendszer

A rendszer magja a *SOA*-ra épít, így minden szoftverkomponens rendelkezik legalább egy szolgáltatással, mindegyiknek megvannak a saját tulajdonságai. A szolgáltatások, mint önálló entitások léteznek. Az arra jogosult szoftverkomponensek és felhasználók igénybe vehetik őket. Az együttműködő felhő robotikai szoftver szempontjából viszont nagyon fontos, hogy ezek rendszerszinten kompatibilisek legyenek egyással, valamint adaptívak legyenek a változásokra. Ennek elérése érdekében létrehoztam egy generikus szolgáltatásrendszert. A tervezés alapjait az *OSI* kommunikációs szabvány [25] alkalmazási rétegére helyeztem (7. réteg), melyet további részegységekre bontottam, a szolgáltatásrétegekre. A tervezési mintát a 3.12. ábrán szereplő példa demonstrálja.

A rétegszerkezetben az *L1-L7* az *OSI* szabvány rétegeinek felel meg. Az *L7* tartalmazza az *SL1-SL4* alrétegeket, ezek a szolgáltatásrétegek (*Service Layer*). Minden SL tartalmaz legalább egy szolgáltatást (*Service*). A szolgáltatásrétegekben alulról felfelé haladva szolgáltatás-kiterjesztés és szolgáltatás-addíció zajlik, majd egy felsőbb szintű szolgáltatás generálódik. Vegyük például az *SL1* szinten az *S1* és *S2* szolgáltatásokat. Ezek addíciója és kiterjesztése után egy *SL2* szinten lévő szolgáltatás generálódik, ez lesz az *S6*. Az *SL2* szinten az *S6*, az *S3* és az *S4* adódik össze, amiből az *S7* generálódik az *SL3*-ban. Az *S5* esetében annyiban különbözik a helyzet, hogy nem adódik össze, vagy akár úgy is fogalmazhatunk, hogy önmagával adódik össze, így ugyanazt a szolgáltatást kapjuk az *SL1*, *SL2* és *SL3* szinteken. Az *SL4*-ben pedig már az *S5* és *S7*-ből generálódik az *S8* szolgáltatás.



3.13. ábra. *A robot hardverének leképezése szolgáltatásokra*

A szolgáltatások külön-külön is igénybe vehetőek, ha az Igénylő úgy kívánja. A biztonsági követelmények ebben az esetben viszont kritikusak. Vegyünk egy példát az önvezető autókról. A járművet külön vezérelhető alkatrészek alkotják, melyeket a felhőből vezérlünk. Ez kifejezetten hasznos lehet például a távoli diagnosztikai eljárásokban, hiszen minden egységet külön tesztelhetnek. Ez azonban nem elhanyagolható támadási felületet nyit a külvilág felé. Ezért nagyon fontos elvégezni a megfelelő biztonsági intézkedéseket az emberi és anyagi károk elkerülése érdekében.

3.5.1. Teszt eset 8: Szolgáltatás egyesítés a szolgáltatásrétegekben

A robot hardverén található szenzorok és beavatkozók külön-külön elérhető szolgáltatásokat nyújtanak. A szolgáltatásrétegben ezek az *SL1* szinten helyezkednek el. Ezen szolgáltatások kiterjesztésekor egy *SL2* szintű szolgáltatás jön létre, ez a Robot vezérlő modul (3.13. ábra).

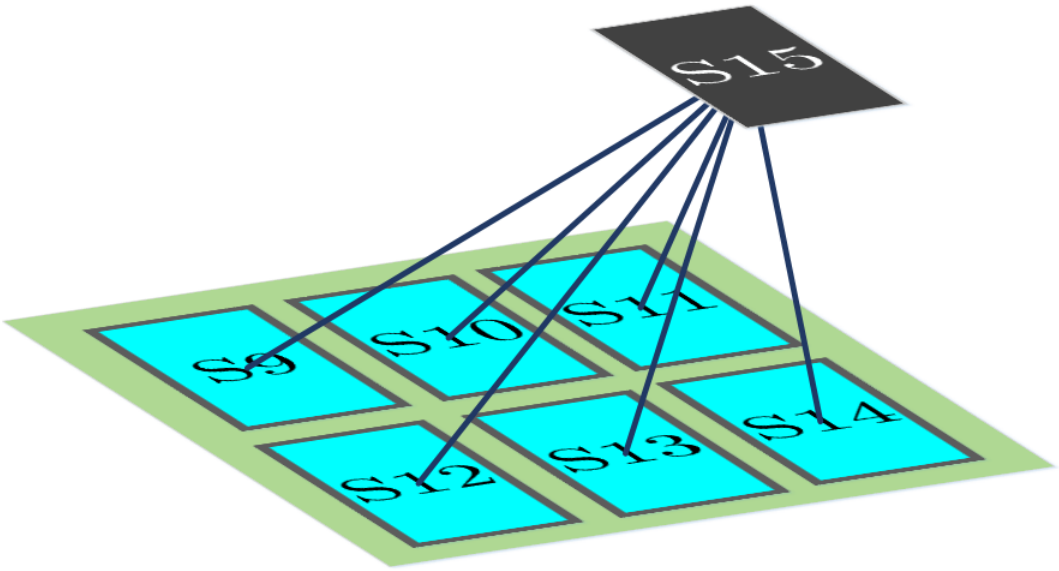
3.5.2. Teszt eset 9: Terület alapú szolgáltatás-kiterjesztés

Az architektúrát lényegét egy másik szemszögből is megközelíthetjük, melyet a 3.14. ábra demonstrál. Látható, hogy az *S9-S14* szolgáltatások egy síkban fekszenek, amely tekinthető például egy helyiség alapterületének. Ezekre hivatkozik az *S15*, amely logikailag felsőbb szinten van. Ugyanolyan szolgáltatás-egyesítés történik itt is, mint az előző esetben.

Erre egy egyszerű példa, ha egy helyiségben lévő hőmérséklet-szenzorok értékeit szeretnénk átlagolni. Az Intelligens vezérlő ezen szenzorok szolgáltatásainak (*S9-S14*) kiterjesztéseként létrehozza *S15*-öt.

3.6. Folyamatok kiértékelése

A rendszer adaptívan kezeli a szolgáltatások működését, a helyzettől függően kiválasztja az optimális megoldást. Ehhez minimalizálni kell a feladat végrehajtása során fellépő



3.14. ábra. *Példa: Területi szolgáltatásréteg*

veszteségeket. Ez egyszerű matematikai módszerrel leírható, tehát egy költségfüggvényt kell definiálni az adott paraméterekhez, feladathoz, körülményekhez, ami alapján képes metrikusan osztályozni a problémát. A költségfüggvény több valós és virtuális paraméter alapján is meghatározható, amely a felhasználási területtől függ. Az alábbi paramétereket alkalmazhatjuk például:

- Fizikai távolság
- Hálózati késleltetés
- Számítási kapacitás
- Energia fogyasztás
- Zajszint, zajszűrés, hibatűrés

A költségfüggvény (c) formális definícióját a (3.3) egyenlet írja le. Az X a paraméterek értékeit jelöli. Az f_i függvényt a felhasználó szabja meg attól függően, hogy milyen paraméterre szeretne optimalizálni.

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} \in \mathbb{R}^N \tag{3.3}$$

$$c(\mathbf{x}) = \sum_{i=1}^{N} f_i(x_i)$$

3.7. Hibakezelés

Robotikai rendszer lévén az egyik legfontosabb követelmény a robosztus működés. Ennek elésére érdekében gondoskodni kell arról, hogy a rendszer hibakezelő komponense a lehetséges hibák előfordulásának valüszínűségét minimalizálja. A működési hibáknak kólönbőző forrásai lehetnek, melyeket az alábbi sorrendben priorizálhatunk:

1. Hardver eredetű meghibásodás
2. Szoftver eredetű meghibásodás
3. Hálózati hibák

A robot élettartama növekedtével arányosan nő a hardver megibásodásának valószínűsége, amely akár a teljes robot cellát működésképtelenné teheti. Ebben az esetben az első és legfontosabb teendő a robot szervizelése. Ekkor meg kell találni a hiba pontos forrását, amelyre öndiagnosztikai megoldások alkalmazhatóak. Amennyiben a hiba azonosítása az előbb említett módszerrel nem lehetséges, felhasználói beavatkozás szükséges. Mobil robotok esetén a mozgásra képes robotokat önjáró módon kell eljuttatni a hozzá legközelebb eső szervíz üzembe. Ellenkező esetben, a mozgásképtelen robotot el kell szállítani, egy másik, megfelelő teherbíró-képességgel rendelkező robot segítségével.

A hiba továbbá lehet szoftveres, amely ugyancsak komoly problémát okozhat. Egyes szoftverkomponensek működése létfontosságú a robot szempontjából. A tervezés és fejlsztés során ezért minden szoftverkomponenst felügyelet alá kell vonni. Ehhez célszerű redundáns módszereket alkalmazni, valamint egy kiemelt prioriritású szoftverkomponenst elhelyezni a rendszerben, amely más fontos komponensek helyes futásáért felelős.

A harmadik hibalahetőség a hálózati meghibásodásokból ered. Ez a jelenség főként a vezetéknélküli technológiák során gyakori. Mobil sok esetben földrajzilag is nagy távolságot tesznek meg, melynek során hetergon hozzáférési hálózatokon keresztül kommunikálnak a felhővel. Ismert terep és vezetéknélküli lefedettség esetén törekedni kell arra, hogy a robot a lehető legjobb jelszinttel jellemzett útvonalon haladjon. Amennyiben ez nem lehetséges, gondoskodni kell arról, hogy a robot a hálózati kapcsolat megszakadása után is megfelelő magabiztossággal végezze feladatát, vagy keressen egy olyan helyet, ahol már elfogadható átviteli sebesség jellemzi a kommunikációt. A hiba továbbá kisebb helyiségekben és vezetékes környezetben is megjelenhet. Ekkor azonban jóbal kevesebb energiabefektetést igényel a javítás, amely például egy kábelcserével kivitelezhető.

A fejezetben felvázoltam a tervezett felhő robotikai rendszer felépítését, valamint konkrét példákkal támasztottam alá a koncepcióját. Részletesen bemutattam a rendszer működésének alapját képező szolgáltatásokat, valamint a rajtuk végezhető műveleteket: az addíciót és a kiterjesztést. Ismertettem továbbá a teszt eseteket, melyeket a komponensekkel végeztem. Végül pedig kitértem néhány hibakezelési módszerre.

4. fejezet

Implementáció és tesztelés

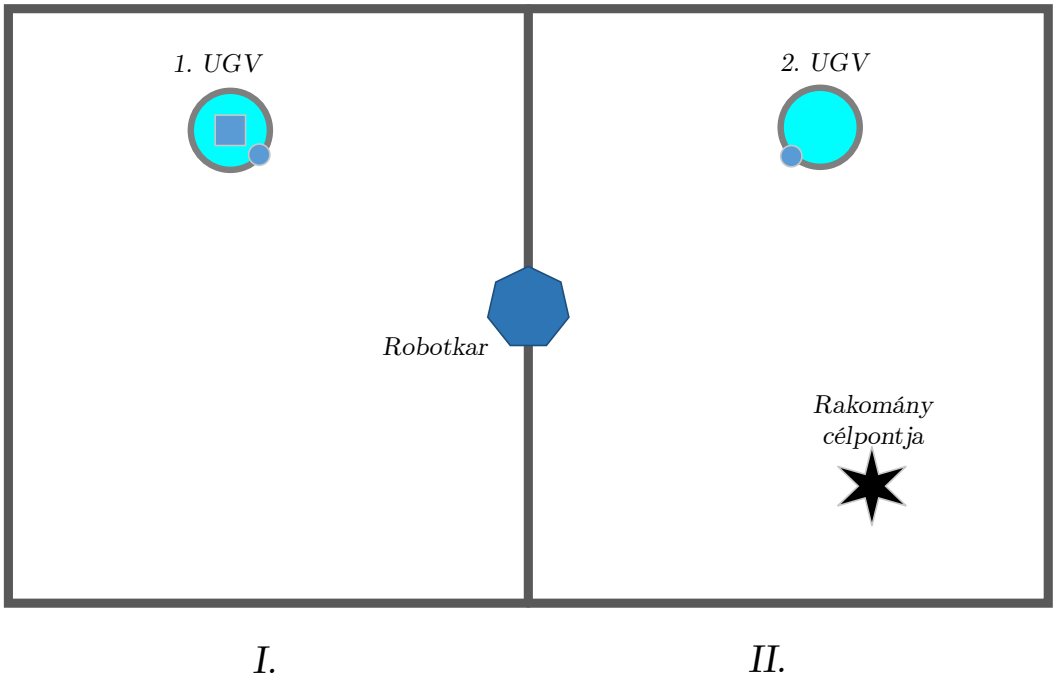
A rendszer komponenseinek működését tisztán virtuális környezetben teszteltem, amelyhez készítettem saját szimulációkat. A szimulációk során vizsgáltam a teljesítményt és a rendszer komponenseit, ezzel alátámasztva az elméleti működést. Továbbá segítségemre volt a *Gazebo* szimulációs környezet, amely valósághű méréseket tesz lehetővé. A tesztek eredményeinek kiértékelése során figyelembe vettem a dolgozat fő aspektusait: a szolgáltatásrétegeket és egyesített szolgáltatásokat. Az implementációt és a tesztelést számítógépen valósítottam meg, melynek paraméterei az alábbiak:

- Processzor: 2.6 GHz, 2 mag
- Ram: 8 GB
- Háttértár: SSD
- GPU: integrált

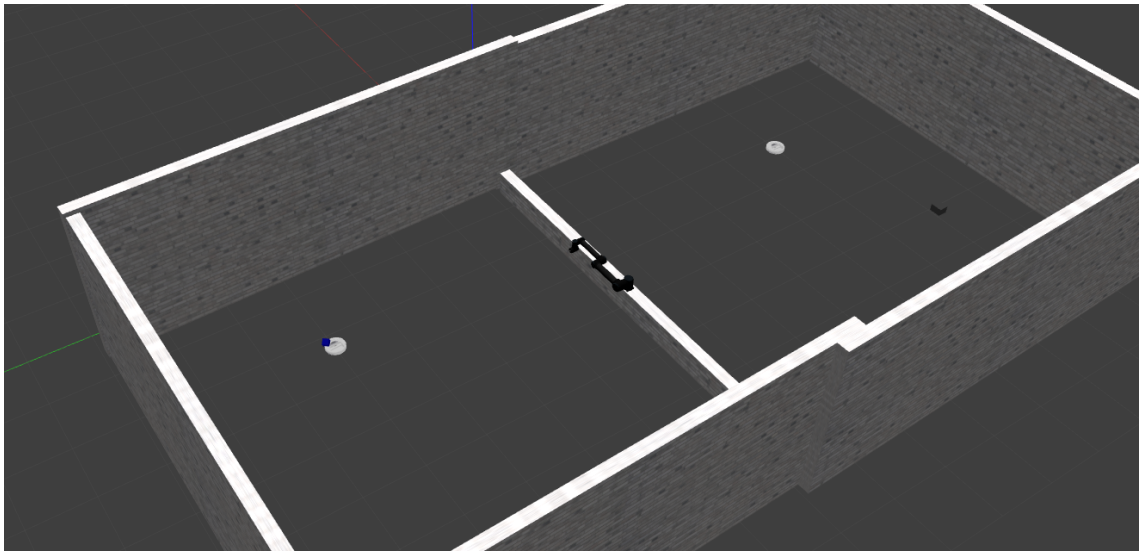
A tesztelés céljából a *Gazebo*ban létrehoztam egy virtuális arénát. Készítettem egy egyszerű épületet, 2 helyiséggel, melyek között alacsony választófal van. Elhelyeztem egy-egy *UGV*-t a két szobában, valamint egy robotkart az épület közepére. Az aréna elrendezésének vázlatát a 4.1. ábra mutatja, a 4.2. ábra pedig betekintést ad a 3D szimulációs környezetbe. Az elvégzendő feladat, hogy az I. helyiségben lévő robot eljuttassa a rakományát a II. helyiségbe. Itt lép életbe a robotok együttműködése, hiszen a robotkar képes átemelni a csomagot 1. robotról, és elhelyezni azt a 2. robotra. Mindehhez persze a 2. robotnak is meg kell közelítenie a robotkart.

A teszt során végrehajtottam a szolgáltatás-egyesítést, pontosabban a szolgáltatás-kiterjesztést, melynek segítségével a két robot egy *SL2*-es szolgáltatásként érhető el. Ezen a szolgáltatás hívásakor a két robot ugyanazt a célpontot tűzi ki, valamint oda is fog menni.

A rendszer indulásakor először a létfontosságú komponensek inicializálódtak: az Intelligens vezérlő, a Szolgáltatás-kezelő, a Feladatkezelő, és az Adatkezelő. A Szolgáltatás-kezelő ismert címen és porton fut, így a regisztrálni kívánó szolgáltatásoknak erre kell hivatkozniuk. Ezt követően indultak a Robot vezérlő modulok (2 *UGV* és 1 robotkar vezérlő), melyek regisztálták magukat a Szolgáltatás-kezelőben, egyedi portszámmal ellátva.



4.1. ábra. *A teszt aréna felülnézetből*



4.2. ábra. *A teszt aréna*

A végrehajtandó feladat indítása a felhasználó parancsával történt, melyet az Intelligens vezérlő feldolgozott, majd szálakra bontotta azt. Ebben a konkrét esetben a felhasználó azt kívánta, hogy a csomag eljusson a célponthoz. Az Intelligens vezérlő felderítette az elérhető szolgáltatásokat, így ezután már tisztában volt a helyiség alaprajzával, valamint robot eszközökkel tulajdonságaival. Az Intelligens vezérlő meghatározta a részfeladatokat, miszerint az 1. és 2. robotnak meg kell közelítenie a robotkart olyan távolságban, hogy a kar biztonságosan át tudja pakolni a csomagot. A harmadik szál természetesen a robotkar pakolása, melyet csak akkor kell végrehajtani, mikor mindkét robot odalért hozzá. Ezek alapján az Intelligens vezérlő parancsot küldött a Feladatkezelő modul részére a feladat szálainak végrehajtására. A Feladatkezelő ekkor elindította az előbb említett konkrét részfeladatok végrehajtását. Meghívta a Robot vezérlők szolgáltatását, így az UGV robotok mozogni kezdtek a célpont felé. Mindkét robot megközelítette a robotkart, a rakomány így eljutott a pakolási ponthoz. Ekkor a robotkar mozgatása következett, amely átpakolta a csomagot a 2. robotra. A Feladatkezelő ekkor parancsot adott a 2. robot vezérlő moduljának (Robot vezérlő), hogy menjen a II. helyiségben található célponthoz, amely végrehajtotta a feladatot.

5. fejezet

Következtetések

A rendszer felépítését SOA alapokra építettem, ami moduláris architektúrát eredményezett. A rendszer jól illeszkedik a felhő robotika alapkoncepciójához, hiszen a vezérlő egységet a felhőben helyeztem el. A felhő komponensei fizikai elhelyezkedésük kötetlen, a világ bármely részén lehetnek. A hálózati technológiák segítségével elméletben teljesen transzparens módon valósítható meg a kommunikáció közöttük. A rendszer tehát rendelkezik egy központi logikai egységgel, ez a Felhő. A felhasználó hozzá intézi kéréseit, valamint visszacsatolást is kap egy tetszőleges interfészen. A komplex feladatok végrehajtását az Intelligens vezérlő ütemezi és végrehajtást követel a Feladatkezelőtől. A Szolgáltatás-kezelő, mint SOA alapegysége, központi elemnek tekinthető és kulcsszerepe van abban, hogy a folyamatok hatékonyak és eredményesek legyenek. A szolgáltatások regisztációja egy fontos lépés a folymatok során, azonban az elérhetőség állapotának valós idejű ellenőrzése, valamint ez alapján a felderíthetőség még nagyobb prioritást élvez. A szolgáltatások egyesítése lehetővé teszi, hogy több független hardvert egy logikai elemként kezeljünk, valamint teljesítménybeli javulást is hoz.

A rendszer megvalósítása virtuális környezetben futott a *ROS* és *Gazebo* segítségével. A Gazebo valósághű fizikai szimulációkra képes, azonban valós hardveren is tesztelni kell a rendszert a gyakorlati megvalósíthatóság érdekében. A példák során azonban kiderült, hogy fontos gyakorlati haszna van a felhő robotikai keretrendszernek, hiszen rugalmas együttműködést tesz lehetővé a robotok között.

A rendszer ennek ellenére rendelkezik korlátokkal, hiszen a biztonsági funkciókat mindeddig nem definiáltam, amely kritikus lehet egy valós rendszer esetén. A rendszer - felhő robotikai szoftver lévén - továbbá feltételezi a hálózati kapcsolat állandó meglétét, így az ennek hiányában fellépő anomáliák egyelőre nem tisztázott jelenségek.

6. fejezet

További fejlesztési és kutatási tervek

6.1. Hierarchikus vezérlési modell

Néhány feladat elvégzését olyan környezeti tényezők is befolyásolhatják, melyek megakadályozzák a kommunikációt a lokális vezetéknélküli hálózati hozzáférési pontokkal. A körülmények változásának hatására a végrehajtandó részfeladat is változhat. Ekkor a rendszer egy eleme dönthet úgy, hogy létrehoz egy klasztert a feladat végrehajtására. A döntést végző szerv kiléte az adott helyzettől függ (6.1. táblázat). A döntést végző komponens kinevez egy klaszter vezetőt, aki koordinálja a munkavégzést, így hierarchikussá válik a vezérlés. A klaszter a munka végeztével visszatér, majd tájékoztatja a Feladatkezelőt a munka sikerességéről, részleteiről. Ekkor azt feltételezzük, hogy a klaszter vezető képes ellátni ezt a munkakört, képes végrehajtani egy feladatot az adott folyamat modellje alapján, azonban modellt alkotni nem, vagy nagyon alapvető mértékben tud. A hálózati jelszint megszűnésének két típusa lehet:

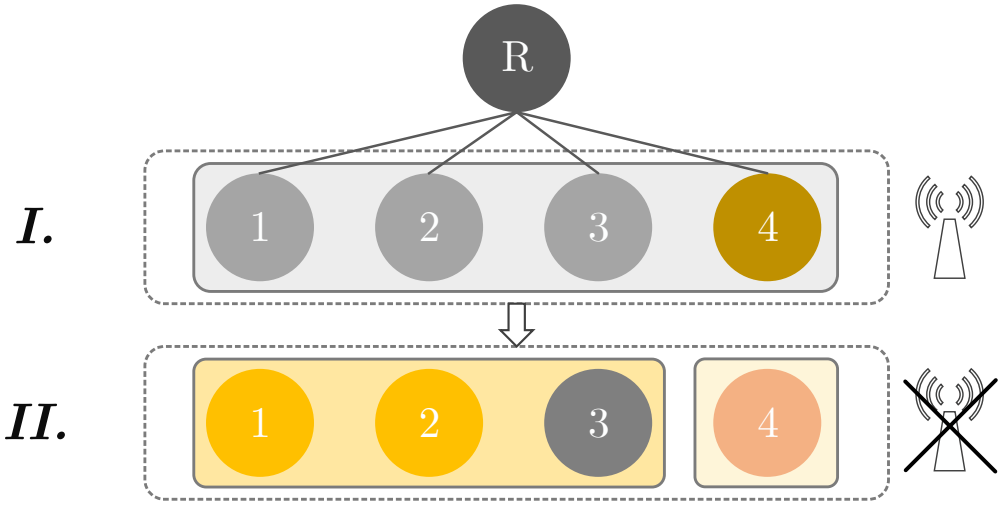
- Determinisztikus
- Sztochasztikus

Típus	Döntést végző komponens	Hibatűrés
Determinisztikus	Felhő	robosztus
Sztochasztikus	Robot	hibaérzékeny

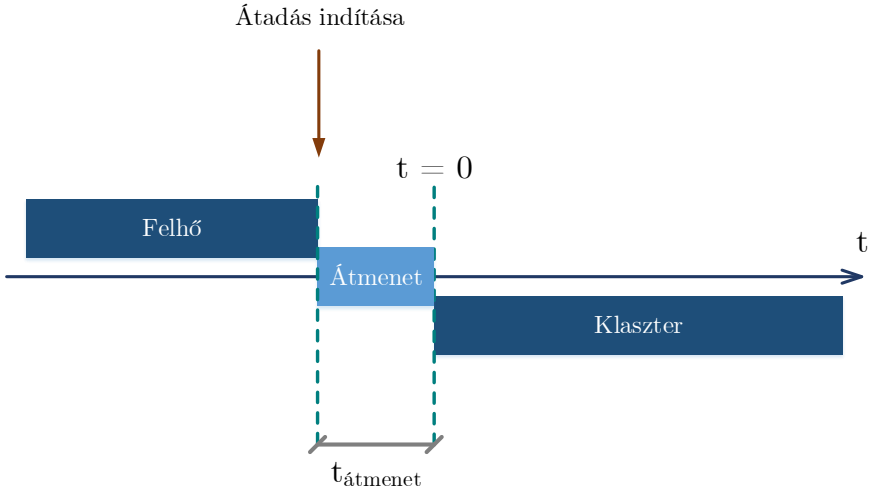
6.1. táblázat. *Hierarchikus vezérlési modell*

Determinisztikus esemény bekövetkeztekor a Felhő hozza meg a döntést a vezérlés átadásáról. A Feladatkezelő előre meghatározza azon területeket, ahol megszakadhat a kapcsolat a robot flotta és a felhő között Ugyancsak a Felhő hoz létre egy klasztert, valamint gondoskodik róla, hogy a klaszter vezető megkapja a Feladatkezelő releváns részének másolatát, így tudatában lesz az elvégzendő feladatnak. Természetesen ezt a műveletet az esemény bekövetkezése előtt kell megtenni, hogy biztosan megtörténjen a vezérlésátadás. Átadáskor a Felhő egy modellt szolgáltat a kiválasztott klaszter vezetőnek, aki ezt felhasználva képes a további vezérlésre. Nyilvánvaló, hogy a mobil robot eszköz kisebb számítási kapacitással rendelkezik, mint a Felhő, így a feladat komplexitása jelentősen

redukálódik arra az időtartamra, amíg a származtatott vezérlés tart. Az átadást követően a klaszter többi tagja a vezetővel kommunikál. A folyamat későbbi szakaszán is megszakadhat a kommunikáció a klaszter vezető és a többi robot között, így a vezérlés tovább öröklődik, abban az irányban, ahol a feladat leginkább releváns része található, így a hierarchikus vezérlési modell elméletben végtelenszer öröklődhet.



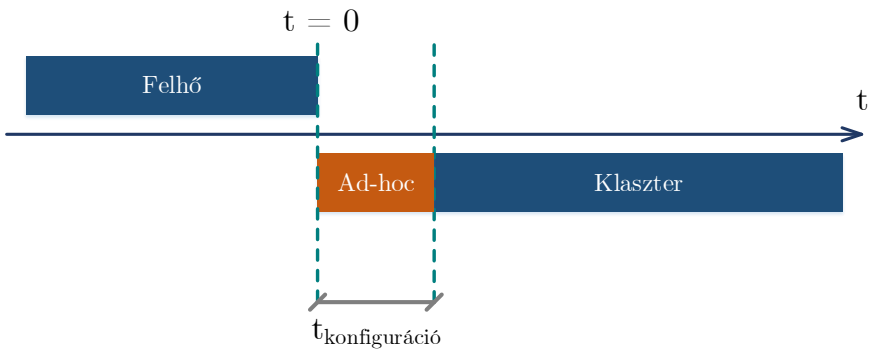
6.1. ábra. *A vezérlés öröklődése szükséges helyzetekben*



6.2. ábra. *Determinisztikus vezérlésátadás*

Sztochasztikus esetben a kapcsolat megszakadását okozó esemény váratlanul következik be, ami azt jelenti, hogy a Felhőnek nincs lehetőség a folyamat modelljét a robot fedélzetére továbbítani (6.3. ábra). Ekkor más megoldásra van szükség, ami megtartja a rendszer adaptivitását, valamint a szükséges robosztusságot. Ekkor persze a robotok célja, hogy minél hamarabb eljussanak arra a helyre, ahol már küszöbszint feletti a jelszint, így létesítve stabil kapcsolatot a felhővel.

Ilyen lehet például barlangi felderítés, mentés.



6.3. ábra. *Sztochasztikus vezérlésátadás*

A problémát egy heurisztikus matematikai függvényt használatával szemléltetem. A vezetéknélküli lefedettég - köztük a *802.11*-ben szabványosított *WiFi* - jelszintje általánosan leírható a *RSSI* (*Received Signal Strength Indicator*) mennyiséggel, melyet *dBm* arányszámmal adhatunk meg. Egy virtuális területen a sík minden egyes pontjához hozzárendeltem az arra jellemző *RSSI* értékeket. A vizualizációt igyekeztem olyan módszerrel elkészíteni, amely képes háromdimenziós függyvényt könnyen értelmezhetően megjeleníteni. A színtérkép (vagy másnéven hőtérkép) alkalmas erre a feladatra, ahol a kétdimenziós sík minden egyes (x, y) koordinátájához lehetőségünk van egy adott szint hozzárendelni, mégpedig a jelmagyarázatnak megfelelő jelöléssel. A jelmagyarázatot általában egydimenziós színskálával adják meg, folytonos értékekkel.

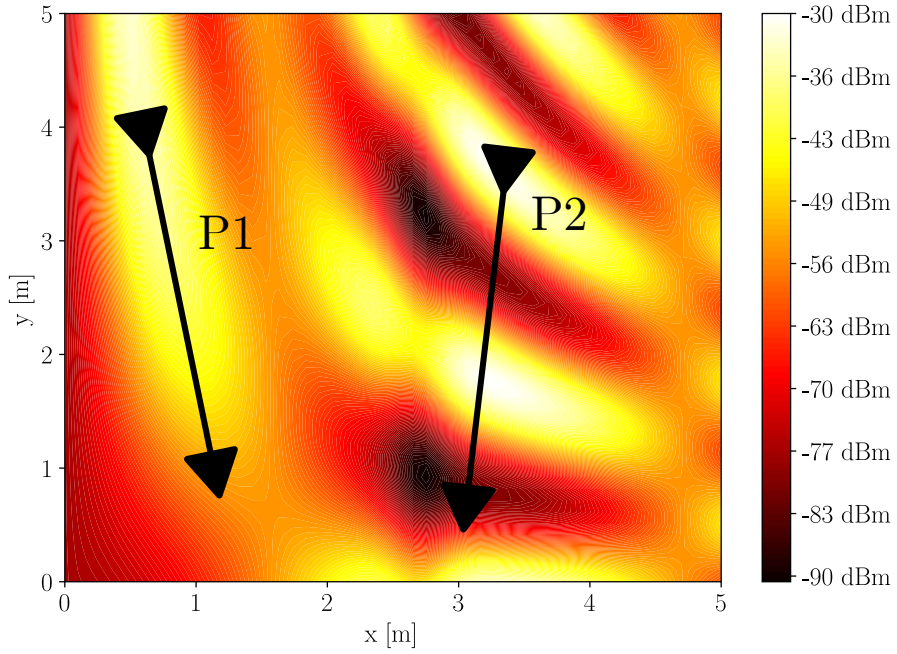
A síkból két egyenes útvonalat emeltem ki: az első esetén (P1) az *RSSI* érték nem csökken a megadott küszöbszint alá (-65 dBm). A második útvonalon (P2) két olyan szakasz is van, ahol a vezetéknélküli lefedettség nem megfelelő, itt nagy a valószínűsége, hogy megszakad a kapcsolat. A két útvonal és az *RSSI* értékek színtérképe a 6.4. ábrán láthatóak. Az elvárt működés, hogy a Felhő átad egy egyszerűsített modellt a robot részére t_{tmenet} idővel a szakadási időpont előtt. A robot megteszi az útjának egy részét, majd mikor a jelszint visszetért a küszöbszint fölé, újra a Felhő fog vezérelni.

6.2. Mesterséges intelligencia

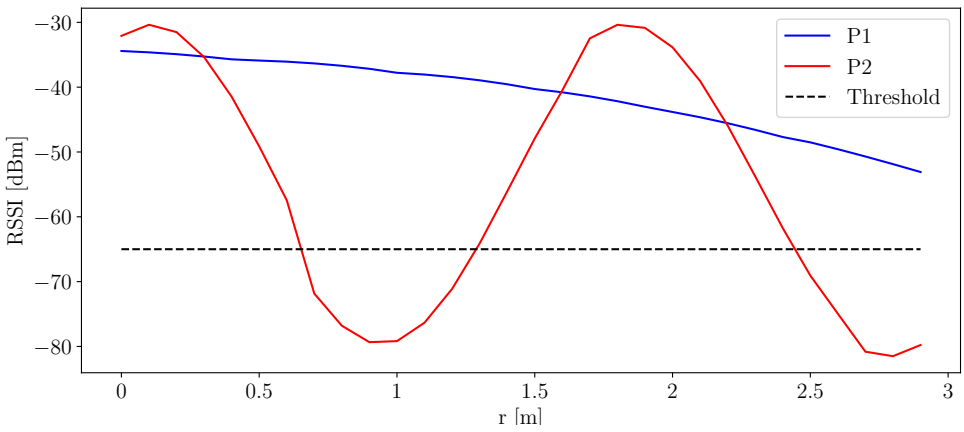
A rendszer sokat profitálhat egy olyan komponens fejlesztésétől, amely képes megtanulni azt, hogy mi az optimális megoldása az adott feladatnak. Ez a funkció most egy heurisztikus költségfüggvény alapján működik, azonban ezt kiválthatja egy megfelelő gépi tanuló algoritmus vagy egy neurális hálózat.

6.3. Digital Twin

A rendszer funkcióinak halmazát bővíteni lehetne a manapság feltörekvő Digital Twin technológiával, melynek lényege, hogy valós paraméterek alapján végeznek szimulációkat a robot virtuális modelljén. A modell viselkedését vizsgálva a mesterséges intelligencia segítségével optimális döntést hoz a vezérlés, amit aztén visszacsatol a valós hardverre.



6.4. ábra. *Vezetéknélküli lefedettség hőtérképe*



6.5. ábra. *A szimulációs útvonalakat jellemző jelszintek*

7. fejezet

Összegzés

A dolgozatomban bemutattam a felhő robotika alapkoncepcióját, ami szerint a vezérlés a felhőbe kerül, ezzel tehermentesítve a robotcellákat a komplex számítási feladatok alól. További célja a virtualizáció, amely földrajzilag teszi függetlenné a vezérlő elhelyezkedését, valamint hatékonyabb erőforrás-használatot eredményez. A felhő robotikában létre kell hozni a robot hardverének egy megfelelő leképezését, amellyel szoftveresen is számolhatunk. Manapság gyakran használt a szolgáltatás-központú architektúra, amely laza csatolást eredményez a komponensek között. Az elméleti áttekintést követően bemutatom az általam tervezett robotikai keretrendszert, amely *SOA* alapokra épít, valamint szolgáltatásrétegeket hoz létre generikusan. Ezen felül leírtam az általam tervezett és kutatott rendszer komponenseinek működését részletesen. Továbbá ezen modulokkal teszteket végeztem és megvizsgáltam ezek hatását több szempont szerint. Ezt követően integráltam az általam elkészített komponenseket, majd megvizsgáltam a teljes rendszer működését. Az eredmények alapján levontam a következtetéseket, valamint ajánlást tettem a jövőbeli fejlesztési feladatokra.

Köszönetnyilvánítás

Ezúton szeretném megköszönni Dr. Vidács Attilának a munkáját, mialatt támogatott engem és szakmai tanácsokkal látott el a szakdolgozatom készítéséhez. Köszönettel tartozom a családomnak és barátnőmnek kitartó biztatásukért, valamint barátaimnak, akik mindvégig mellettem álltak munkám során. Köszönöm mindenkinek!

Ábrák jegyzéke

Irodalomjegyzék

[1] Peter M. Mell and Timothy Grance. Sp 800-145. The NIST Definition of Cloud Computing. Technical report, Gaithersburg, MD, United States, 2011.

[2] Markus Waibel, Michael Beetz, Javier Civera, Raffaello d'Andrea, Jos Elfring, Dorian Galvez-Lopez, Kai Häussermann, Rob Janssen, J.M.M. Montiel, Alexander Perzylo, Bjoern Schiessle, Moritz Tenorth, Oliver Zweigle, and M.J.G. René Van de Molengraft. RoboEarth - A World Wide Web for Robots. *Robotics and Automation Magazine*, 18(2):69–82, June 2011.

[3] D. Hunziker, M. Gajamohan, M. Waibel, and R. D'Andrea. Rapyuta: The roboearth cloud engine. In *2013 IEEE International Conference on Robotics and Automation*, pages 438–444, May 2013.

[4] Ken Gray and Thomas D. Nadeau. *Network Function Virtualization*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2016.

[5] G. Hu, W. P. Tay, and Y. Wen. Cloud robotics: architecture, challenges and applications. *IEEE Network*, 26(3):21–28, May 2012.

[6] Paul D. Hestand. *A Service Oriented Architecture for Robotic Platforms*. PhD thesis, 2011. AAI3459179.

[7] Lucas Bueno Ruas de Oliveira. *Architectural design of service-oriented robotic systems*. PhD thesis, Instituto de Ciências Matemáticas e de Computação, 2015.

[8] Y. Chen, Z. Du, and M. García-Acosta. Robot as a service in cloud computing. In *2010 Fifth IEEE International Symposium on Service Oriented System Engineering*, pages 151–158, June 2010.

[9] A. Vick, V. Vonásek, R. Pěnička, and J. Krüger. Robot control as a service — towards cloud-based motion planning and control for industrial robots. In *2015 10th International Workshop on Robot Motion and Control (RoMoCo)*, pages 33–39, July 2015.

[10] Anis Koubaa. Ros as a service: Web services for robot operating system. In *Journal of Software Engineering for Robotics*, pages 1–14, December 2015.

[11] Jerker Delsing. *IoT Automation - Arrowhead Framework*. CRC Press, 1 edition, 2 2017.

[12] R. Doriya, P. Chakraborty, and G. C. Nandi. 'robot-cloud': A framework to assist heterogeneous low cost robots. In *2012 International Conference on Communication, Information Computing Technology (ICCICT)*, pages 1–5, Oct 2012.

[13] I. Jerstad, S. Dustdar, and D. V. Thanh. A service oriented architecture framework for collaborative services. In *14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise (WETICE'05)*, pages 121–125, June 2005.

[14] Web Services Architecture. W3C Working Group Note, World Wide Web Consortium (W3C), 2 2004. `https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#relwwwrest`.

[15] MQTT version 3.1.1 plus errata 01. OASIS Standard Incorporating Approved Errata 01, Organization for the Advancement of Structured Information Standards, December 2015. `http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/errata01/os/mqtt-v3.1.1-errata01-os-complete.html`.

[16] SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). W3C Recommendation, World Wide Web Consortium (W3C), 4 2007. `https://www.w3.org/TR/2007/REC-soap12-part1-20070427/`.

[17] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645 – 1660, 2013.

[18] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part i. *IEEE Robotics Automation Magazine*, 13(2):99–110, June 2006.

[19] T Huang. Computer Vision: Evolution And Promise. 1996.

[20] Sebastian Thrun. Toward Robotic Cars. *Commun. ACM*, 53(4):99–106, April 2010.

[21] B. Salemi, M. Moll, and W. Shen. SUPERBOT: A Deployable, Multi-Functional, and Modular Self-Reconfigurable Robotic System. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3636–3641, Oct 2006.

[22] HEBI Robotics. `https://www.hebirobotics.com/`. Utolsó letöltés: 2018.10.01.

[23] Robot Operating System. `http://www.ros.org/`. Utolsó letöltés: 2018.10.01.

[24] Gazebo. `http://gazebosim.org/`. Utolsó letöltés: 2018.10.24.

[25] H. Zimmermann. Innovations in internetworking. chapter OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection, pages 2–9. Artech House, Inc., Norwood, MA, USA, 1988.

[26] I. Horváth. Autonomous Robot Control in AR/VR Environment, 2017. Bachelor's thesis.