UNIVERSITATEA SAPIENTIA DIN CLUJ-NAPOCA
FACULTATEA DE ȘTIINȚE TEHNICE ȘI UMANISTE, TÎRGU-MUREȘ
SPECIALIZAREA CALCULATOARE

COMANDA ROBOTULUI BAXTER FOLOSIND GESTURILE MÂINILOR

PROIECT DE DIPLOMĂ

Coordonator ştiinţific:
Dr. ing. Márton Lőrinc
Dr. ing. Szántó Zoltán

Absolvent:
Vágási Zsolt-Ferenc

2023

UNIVERSITATEA „SAPIENTIA” din CLUJ-NAPOCA **Viza facultății:**
Facultatea de Științe Tehnice și Umaniste din Târgu Mureș
Specializarea: Calculatoare

LUCRARE DE DIPLOMĂ

Coordonator științific:	Candidat: **Vágási Zsolt-Ferenc**
Prof. dr. ing. Márton Lőrinc,	Anul absolvirii: **2023**
Ș. l. dr. ing. Szántó Zoltán	

a) Tema lucrării de licență:
Comanda robotului Baxter folosind gesturile mâinilor
b) Problemele principale tratate:
- Comanda la distanță a robotului Baxter prin protocolul MQTT
- Metode de recunoaștere a gesturilor folosind senzorul Kinect
c) Desene obligatorii:
- Schema bloc a aplicației
- Diagramele UML a aplicației
- Măsurători experimentale – Mișcarea robotului.
d) Softuri obligatorii:
- Program Python pentru comunicare la distanță cu robotul Baxter.
- Program Python pentru recunoașterea gesturilor folosind senzor Kinect
e) Bibliografia recomandată:
[1] Fairchild, Carol, and Thomas L. Harman. ROS Robotics By Example: Learning to control wheeled, limbed, and flying robots using ROS Kinetic Kame. Packt Publishing Ltd, 2017.
[2] Aggarwal, Love & Gaur, Varnika & Verma, Puneet, Design and Implementation of a Wireless Gesture Controlled Robotic Arm with Vision. International Journal of Computer Applications. Vol 79, pp. 39-43, 2013.
[3] Yi Li, Hand gesture recognition using Kinect, IEEE International Conference on Computer Science and Automation Engineering, pp. 196-199, 2012.

f) Termene obligatorii de consultații: săptămânal
g) Locul și durata practicii: Universitatea „Sapientia” din Cluj-Napoca,
Facultatea de Științe Tehnice și Umaniste din Târgu Mureș, laborator 243.
Primit tema la data de: 31.03.2022.
Termen de predare: 27. 06. 2023.

Semnătura Director Departament

Semnătura coordonatorului

Semnătura candidatului

Semnătura responsabilului
programului de studiu

Model tip a.

Declaraţie

Subsemnata/ul Vágási Zsolt-Ferenc, absolvent(ă) al/a specializării calculatoare, promoţia 2019-2023 cunoscând prevederile Legii Educaţiei Naţionale 1/2011 şi a Codului de etică şi deontologie profesională a Universităţii Sapientia cu privire la furt intelectual declar pe propria răspundere că prezenta lucrare de licenţă/proiect de diplomă/disertaţie se bazează pe activitatea personală, cercetarea/proiectarea este efectuată de mine, informaţiile şi datele preluate din literatura de specialitate sunt citate în mod corespunzător.

Localitatea,

Data: Absolvent

Semnătura……

Model tip b.

Declaraţie

Subsemnata/Subsemnatul ....., funcţia……,
titlul ştiinţific…… declar pe propria răspundere că ……Numele şi prenumele absolventului……, absolvent al specializării ……Numele specializării conform HG..... a întocmit prezenta lucrare sub îndrumarea mea.

În urma verificării formei finale constat că lucrarea de licenţă/proiectul de diplomă/disertaţia corespunde cerințelor de formă şi conținut aprobate de Consiliul Facultăţii de Științe Tehnice și Umaniste din Târgu Mureș în baza reglementărilor Universităţii Sapientia. Luând în considerare şi Raportul generat din aplicaţia antiplagiat „Turnitin” consider că sunt îndeplinite cerințele referitoare la originalitatea lucrării impuse de Legea educației naționale nr. 1/2011 şi de Codul de etică şi deontologie profesională a Universităţii Sapientia, şi ca atare sunt de acord cu prezentarea şi susținerea lucrării în fața comisiei de examen de licență/diplomă/disertaţie.

Localitatea,

Data: Semnătura îndrumătorului

Ide kerül a Turnitin similarity report

Comanda robotului Baxter folosind gesturile mâinilor

Extras

În ultimii ani, roboții industriali au cunoscut o evoluție semnificativă, având aplicații tot mai diverse. Această lucrare de licență se concentrează pe controlul roboților industriali, având ca obiectiv dezvoltarea unui sistem care să înregistreze mișcările utilizatorului și să le utilizeze pentru a controla un robot industrial. Accentul primar al acestei cercetării a fost pe controlul la distanță al unui robot industrial cu două brațe, prin intermediul mișcării brațului uman.

Pentru a realiza controlul în timp real, am utilizat un senzor Kinect de a doua generație, capabil să urmărească mișcările și să le măsoare distanța. Datele obținute de acest senzor au fost convertite în semnale corespunzătoare pentru a controla brațul robotului. Pentru a controla mișcarea brațelor robotului industrial, am utilizat unghiurile articulațiilor utilizatorului.

Pentru implementare, am ales un robot cooperativ Baxter, care este un robot industrial montat pe o platformă fixă cu două brațe, fiecare având șapte grade de libertate. Sistemul dezvoltat permite controlul robotului prin gesturi ale mâinii.

Sistemul permite utilizatorului să controleze instrumentele de prindere ale brațului robotic prin gesturi ale mâinii, pe lângă mișcarea propriu-zisă a brațului. În funcție de starea mâinii, instrumentele de prindere ale robotului pot fi deschise sau închise.

Cuvinte-cheie: roboti industriali, control la distanță, gesturi, Kinect

SAPIENTIA ERDÉLYI MAGYAR TUDOMÁNYEGYETEM
MAROSVÁSÁRHELYI KAR
SZÁMÍTÁSTECHNIKA SZAK

Baxter robot kézgesztus alapú vezérlése

DIPLOMADOLGOZAT

Témavezető:
Dr. Márton Lőrinc
Dr. Szántó Zoltán

Végzős hallgató:
Vágási Zsolt-Ferenc

2023

Kivonat

Az elmúlt időszakban jelentős növekedést tapasztalhattunk az ipari robotok terén és felhasználási területük is folyamatosan bővül. Ezek a robotok egyre több funkcióval rendelkeznek és egyre szélesebb körben képesek emberi munkákat elvégezni. A kutatásunk során az ipari robotok vezérlésével foglalkoztunk, célunk egy rendszer kifejlesztése volt, amely rögzíti a felhasználó mozgását és ezek alapján számítja ki az ipari robot vezérléséhez szükséges jeleket.

Diplomadolgozatomban egy távoli vezérlésű kétkarú ipari robot vezérlésére alkalmas rendszer megoldását terveztük, amely az emberi kar mozgását használja alapul. A robot mindkét karját valós időben kellett vezéreljük, ezért a második generációs Kinect érzékelőt alkalmaztuk, amely alkalmas a mozgás követésére és távolság mérésére. A Kinect szenzor által nyújtott adatokat olyan formába alakítottuk át, ami alkalmas egy robotkar vezérlésére. Az ipari robot karjainak vezérléséhez a felhasználó végtagjainak és ízületeinek szögeit használtuk.

A kivitelezéshez egy fejlesztési célra tervezett Baxter kooperatív robotot használtunk, amely egy fix platformra rögzített robot, amely két, egyenként hét szabadságfokkal rendelkező, robotkarral rendelkezik. A rendszer lehetővé teszi a robot kézmozdulatokkal való vezérlését.

A rendszer a robotkar mozgatása mellett lehetővé teszi a felhasználónak, hogy a robotkar megfogó eszközeit kézmozdulatokkal vezérelje. A kéz állapota alapján a robot megfogó eszközei kinyílnak vagy bezáródnak.

Kulcsszavak: ipari robot, gesztusok, távvezérlés, Kinect

Abstract

In recent years, industrial robots have witnessed significant growth, with expanding applications and increasing functionality that enables them to perform a wider range of human tasks. In this thesis, our research focused on developing a system that records user movements and utilizes them to calculate the necessary control signals for an industrial robot. Specifically, we designed a solution for the remote control of a two-armed industrial robot based on human arm movements.

To achieve real-time control of both robot arms, we utilized a second-generation Kinect sensor, renowned for its movement tracking and distance measurement capabilities. The data captured by the Kinect sensor was subsequently transformed into a suitable format for controlling the robot arm. The control mechanism relied on the angles of the users' limbs and joints to manipulate the arms of the industrial robot.

Our implementation was centered on employing a Baxter cooperative robot. Baxter is a fixed platform robot, equipped with two robotic arms, featuring seven degrees of freedom each. The system provides control of the robotic arms through hand gestures.

The system enables the user to control the gripping tools of the robot arm through hand gestures, in addition to moving the robot arm itself. Based on the state of the hand, the robot's gripping tools can open or close.

Keywords: industrial robot, remote control, gestures, Kinect

Tartalomjegyzék

Ábrák, táblázatok jegyzéke

1. Bevezető

Az ipari robotok először az 1960-as években jelentek meg és egészen az 1990-es évekig ezek jelentették a robotika legnagyobb százalékát. Főleg az autó iparban voltak használatosak és ez az ipar diktálta ezek fejlődését. Egy ipari robot áll egy vagy több robotkarból. A robotkar az az egység, ami csuklókból, szegmensekből és végberendezésből áll, célja, hogy egy adott feladatot elvégezzen.

Ezen robotok vagy robotkarok előnye az emberi erőforrással szemben az, hogy nem fáradnak és monoton, ismétlődő munkafolyamatokat konzisztens precizitással tudnak elvégezni. Ezek a munkafolyamatok elvárják a fenntartott figyelmet, amit sok esetben mi emberek nem tudunk biztosítani. A robotok a legtöbb iparágban megtalálhatóak. Használatosak nehéz tárgyak mozgatására, kiemelt pontosságot igénylő munkafolyamatok elvégzésére vagy olyan munkákra, amik esetlegesen veszélyesek lennének az emberre nézve.

A robotok felhasználása, napjainkban több ágra is kiterjedt, használatosak például az orvostudományban, a kiszolgálóiparban, rehabilitációs központokban. A robotok használhatóak emberek kiszolgálására, különböző műtétek elvégzésére, terápiás célokra, vagy különféle környezetben kisebb feladatok elvégzésére (például tárgyak szállítása vagy adott területek felderítése).

Ezek a robotok vagy robotkarok sok esetben kell emberek között dolgozzanak, így fontos, hogy megfelelő biztonsági kritériumoknak megfeleljenek, hogy minimálisra lehessen a baleset veszélyét csökkenteni. Ezekre az esetekre fejlesztették ki a kooperatív robotokat. A kooperatív robotok vagy robotkarok lényege, hogy abban az esetben, amikor azt érzékeli, hogy az elvártnál nagyobb visszaható erő hat rá, abbahagyja a folyamatot, amit épp végez. Ezzel nem csak biztonságosabb az emberekre nézve, hanem saját magát is óvja.

A kooperatív robotok (cobot) 1996-tól léteznek, ezeknek célja, hogy emberekkel egy térben, biztonságosan tudjanak működni. Ahhoz, hogy ezt elérje, a robot ellenőrzi a rá ható külső erőket, arra az esetre, hogyha valamivel érintkezésbe kerülne a környezetében. A folyamatot csak akkor hajtja végre, ha ez az érték megfelelő tartományon belül van, ha a robotra ható erő meghaladja a biztonságos értéket, akkor a robot leáll abban az állapotban, amiben épp van.

A fentiek alapján adott volt a feladat, hogy megvalósítsuk egy robot, vezérlését távolról. A fejlesztésre használt robotot hálózatra kellett lehessen kötni, hogy ne kelljen közvetlenül a robotra csatlakozni. A rendszer vezérléséhez a felhasználó karjának mozgatásából kinyert

értékeket alakítottuk át, a robot vezérléséhez szükséges paraméterekké és ezzel tudtuk a robotunkat irányítani.

A robotra való fejlesztésben egy felmerülő probléma volt, a kompatibilitás. Mivel ipari robotról van szó, ezeknek a fejlesztése nincsen szinkronban a modern fejlesztési technológiákkal, így ahhoz, hogy minél hatékonyabban lehessen dolgozni, ezt a problémát át kell hidalni. A célunk az volt, hogy az robot operációs rendszerétől függetlenül, akármilyen eszközről, bármilyen technológiát használva lehessen arra fejleszteni.

Hasonlóképpen a robot vezérlésének pontossága is egy felmerülő probléma. A dolgozat során több hibalehetőség is fennáll, amikkel végig számolni kell és ezek után hangolni az adott paramétereket.

2. Célkitűzések

A cél egy olyan rendszer kialakítása, ami képes a felhasználó mozdulatai alapján vezérelni egy kétkarú robot végtagjait. A felhasználó elhelyezkedésétől függően a robotnak tudnia kell tükörben és szimmetrikusan is mozognia. A rendszernek emellett a felhasznált kamera képet is tudnia kell közvetíteni a felhasználó felé, egyfajta visszajelzésként. Fontos, hogy a robot mozgása a felhasználó mozgását tükrözze, a lehető legpontosabban.

A fent említett rendszer kialakításához az első cél, hogy a felhasználó végtagjaiból hasznos adatokat tudjak kinyerni. Ilyenek a felhasználó kezének a pozíciója, szögei, állapota. Ezeket az információkat pedig fel kell dolgozni, értelmezni.

Egy másik cél, a rendszer kialakításában, hogy a használt ipari robotot vezérelni tudjam. Ezt magába foglalja a robotkarok mozgatását, a megfogó eszközök gesztus alapú vezérlését és a követési folyamat leállítását, szintén egy gesztus használatával.

A fentiek alapján pedig, a robot karjaira kell illeszteni a felhasználót figyelő szenzorból kinyert adatokat. A felhasználó mindkét karját kell folyamatosan követni, hasonlóképen ezeket a mozdulatokat a robot karjaira rá kell illeszteni.

A komponensek között, célom megtervezni és megvalósítani a robot és számítógép közötti kommunikációt. Ez magába kell foglaljon olyan információkat, mint a felhasználó mozgásából kinyert mozgásadatok, adott vezérlési parancsok, a felhasználó mozgása alapján. A robot felől pedig olyan információk, amik a robot aktuális helyzetét leírják.

3. Irodalomkutatás

A robotikában már használatos módszer a gesztusvezérlés. Ezzel egy pontosabb irányítást tudunk a robotkaroknak biztosítani, mint a hagyományos vezérlési módokkal, mint például billentyűzet vagy esetleg kontroller. Mivel a robotkarok használata széles körben elterjedt (mind ipari mind kutatási célokból), többféleképpen közelítették meg a gesztusvezérlés problémáját.

Az [1] -es cikkben, a robotkar vezérlésére előre meghatározott kézmozdulatokat használtak. Egy webkamera segítségével rögzítették a felhasználó által végzett kézmozdulatot, amit egy számítógép feldolgozott és hasonlóság alapján megállapította, hogy a felhasználó melyik gesztust hajtotta végre. Ezt azután hálózaton keresztül elküldték a robotkarnak azt a feladat sorozatot, amit az adott mozdulathoz kötöttek. Ezt ezután a robotkar végrehajtotta.

A [2] -es cikkben hasonló megközelítést használtak, viszont itt a robotot egy mobilis alapra helyezték és így nem csak mozdulatokat volt képes végrehajtani, hanem térben is tudott mozogni. Itt a kézmozdulat felismerésére gyorsulásmérőket rögzítettek a felhasználó kezeire és lábaira, így tudták meghatározni, hogy milyen mozdulatokat kell a robotkar elvégezzen.

A [3] -as cikkben az első megoldáshoz hasonlóan webkamerát használtak a gesztus felismerésére és OpenCV segítségével feldolgozta azt. Az innen kinyert információt küldte Bluetooth-on keresztül a robotkart irányító mikrovezérlőnek.

A gesztusok felismerésére más megközelítéseket is használtak, mint az egyszerű kamera vagy a gyorsulásmérő. Az egyik ilyen megoldás, hogy a felhasználó kézmozdulatait a Kinect kamera segítségével vesszük és dolgozzuk fel. Az [4] -es cikkben az első generációs kinect kamerát használták erre a célra. A mélységi kamera képét használva elkülönítették a felhasználót a háttértől, a képkockák közötti különbségekből pedig ki fel tudták dolgozni annak mozgását.

A [5]-ös számú publikációban szintén Kinect kamerát használtak a jelnyelvben vagy egyéb nagy pontosságot igénylő helyzetekben használatos kézmozdulatok szöveggé vagy paranccsá alakításához. Ebben a tanulmányban a Kinect kamera már meglévő funkcióit bővítették ki, mivel itt pontosan kellett a felhasználó ujjait felismerni és feldolgozni. Az emberi ujjak felismerésére az első lépés az volt, hogy olyan három pontot rögzítsenek, amit a felhasználó ujjait meg tudják határozni, ezután pedig ezek alapján tudva az ujjak helyeit, szintén elmentett kézmozdulatokhoz, jelzésekhez hasonlították az észlelt gesztusokat.

A gesztusokkal való vezérlés esetében több szempont is változó lehet felhasználási esetenként. Ilyen változó paraméterek lehetnek a fényviszonyok, a felhasználó elhelyezkedése az őt érzékelő eszközhöz képest, a felhasználó testi tulajdonságai. Ahhoz, hogy a rendszer megfelelően működőképes legyen, szükség van adott esetben kalibrációra. Ezt a [5] -ös tanulmányban úgy

oldották meg, hogy a program előír adott mozdulatokat, amit a felhasználónak meg kell tenni, így tudja az felismerni a felhasználót és lekövetni azt.

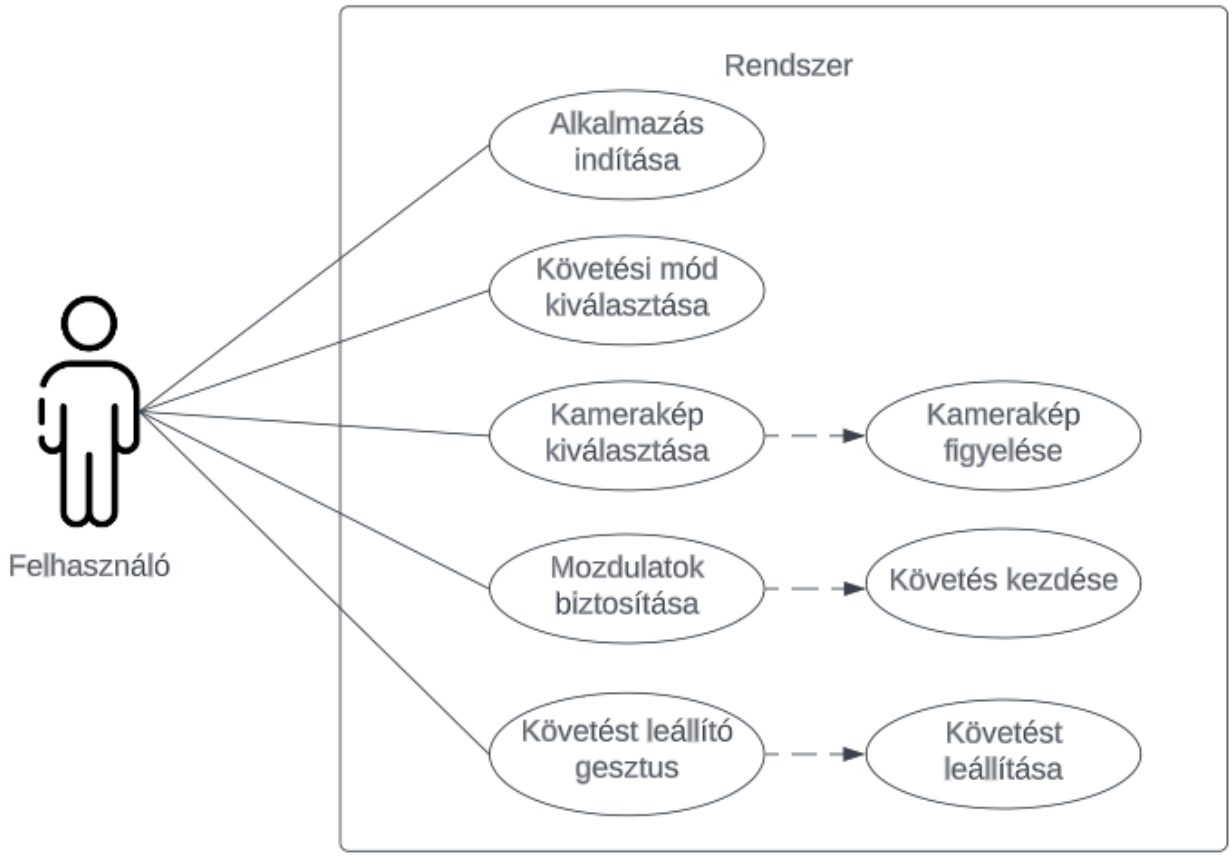
Egy másik megközelítés a robotkar és a felhasználó mozgásának koordinálása. Ebben az esetben a felhasználó pillanatnyi helyzetéből kell információt kinyerni, és azt olyan formába alakítani, amivel lehet a robotkart vezérelni. Ezt két féle módszerrel lehet megvalósítani.

Az egyik ilyen megközelítés, az inverz kinematikát használja, amikor a robotnak csak a végberendezését akarjuk irányítani. Megadjuk a robotkar végberendezésének a koordinátáit és adott képletek alapján kell kiszámolni a robot csuklóinak a szögeit. Ezt a módszert alkalmazták a [6] -os publikációban, is, ahol a cél egy robot kar vezérlése volt, szintén kézmozdulatokkal. A felhasználó kézmozdulatainak felismeréséhez Kinect kamerát használt, amivel meg tudta határozni a saját kezének (jelen esetben a tenyerének) a koordinátáit. Miután meghatározta a konverziós mátrixát a robotkarnak, ki tudta számolni a szögeket, amikre a robot be kell a saját csuklóit állítsa.

A másik megközelítés a robotkar vezérlésére az, hogy direkt kinematikai feladatként tekintünk rá, vagyis direkt a szögeket adjuk meg, és az alapján vizsgáljuk, hogy a végberendezés koordinátája miként változik. Az inverz kinematika mellett, ezt a megközelítést is tanulmányozzák a [7] -es tanulmányban. Itt a Baxter robot egy kutatásra szánt változatát vezérelték meg mindkét módszerrel. Az egyik módszer a vektoros megközelítés volt (direkt kinematikai feladat). Ennek a módszernek a lényege, hogy a felhasználó karjainak mozgását vetítették át a robot karjaira, mintát véve az emberi kéz mozdulataiból. Egy Kinect kamera segítségével a felhasználó ízületeit vektorokkal kötötték össze és azokból szögeket számoltak. Így összesen négyféle szöget tudtak kinyerni, amit a robot karjára lehetett vetíteni. Ezek a szögek a váll RPY (Roll, Pitch, Yaw), Euler féle szögei és a könyök elhajlásának szöge. Így összesen négy szabadságfokon tudták a robotkart mozgatni a hétből.

Mind a két megközelítésnek megvannak a maga előnyei és hátrányai. Hogyha a robot vezérlésére inverz kinematikai feladatként tekintünk, pontosabb követést és több szabadságfokon való mozgást tudunk elérni, viszont sok esetben ez a módszer nem biztosítja, hogy a robot mozgása az emberéhez hasonló lesz. Hogyha a másik megközelítést alkalmazzuk, a robot mozgása jobban fog a felhasználóéra hasonlítani, viszont kevesebb ízületet tudunk vezérelni (a lehetséges hétből csak négyet).

4. Követelmény specifikáció.



Ábra 1: A rendszer felhasználói esetdiagramja

A rendszerhez tartozó applikáció egy Windows rendszerre fejlesztett WPF applikáció. Miután a felhasználó elindítja az applikációt, egy ablakon kiválassza a követéshez és kamerakép közvetítéséhez szükséges opciókat, majd miután a robot kezdeti állapotba helyezi magát, elkezdődik a felhasználó követése. Ezt később egy gesztussal meg lehet állítani és a robot visszaáll kezdeti állapotába.

4.1.1. Felhasználói követelmények

A fenti, 1. ábrán látható az applikáció használati eset (use case) diagramja. Az aktor azokat a felhasználókat jelöli, aki az alkalmazást használni szándékoznak. A fontosabb használati esetek:

- Alkalmazás indítása: A felhasználó, ahhoz, hogy az alkalmazást használni tudja, egyszer el kell azt indítania

- Követési mód kiválasztása: Amikor az alkalmazás elindul, előjön egy ablak, ahol a felhasználó ki tudja választani, hogy a robot szimmetrikusan kövesse le a mozgását, vagy tükrözve
- Mozdulatok biztosítása: A kamerakép és a követés típusának kiválasztása után a felhasználó el tudja a követést indítani. Ekkor a robot kezdeti állapotba helyezi magát és elkezdi a karjait a felhasználó mozdulatai után mozgatni. Mivel a robot karjai, a felhasználó mozgását követik, ezért a felhasználónak biztosítania kell a vezérléshez a kézmozdulatokat
- Követés leállítása: Amikor a felhasználó a kezeit „X" alakba helyezi, a felhasználó mozdulatainak követése leáll és a robot visszaáll a kezdeti pozíciójába.

4.1.2. Rendszer követelmények

a.) Funkcionális követelmények

A rendszer alapvető funkcionalitása, hogy az ipari robot karjai lekövetik a felhasználó karjainak mozgását. Ezt a robotnak tudnia kell kétféleképpen. Abban az esetben, ha a felhasználó és a robot egyirányba néz, a robotnak szimmetrikusan kell követnie a felhasználót, hogyha pedig egymással szemben állnak, akkor a követésnek tükrözöttnek kell lennie. A rendszernek biztosítania kell egy felületet, ahol ezt a felhasználó ki tudja választani.

A rendszernek képesnek kell lennie a robot végberendezéseinek irányítására, a felhasználó kezeinek állapota alapján: hogyha a felhasználó keze zárt állapotban van, a robotkar végberendezésének is zárt állapotba kell kerülnie és fordítva.

A rendszernek rendelkeznie kell egy vészleállítási gesztussal, amivel le lehet a mozgás követését állítani biztonságosan, anélkül, hogy a felhasználó a robotban, vagy saját magában kárt tenne.

A felhasználónak a Kinect kamerától legalább egy méterre kell állnia és egyedül lennie, mint emberi alak a kamera terében, hogy az alkalmazás a lehető legpontosabban tudja lekövetni.

b.) Nem funkcionális követelmények

A rendszer működéséhez a felhasználónak tudnia kell futtatni egy WPF applikációt, amihez szükséges legalább egy Windows 8-as operációs rendszer. A felhasználó kézmozdulatainak

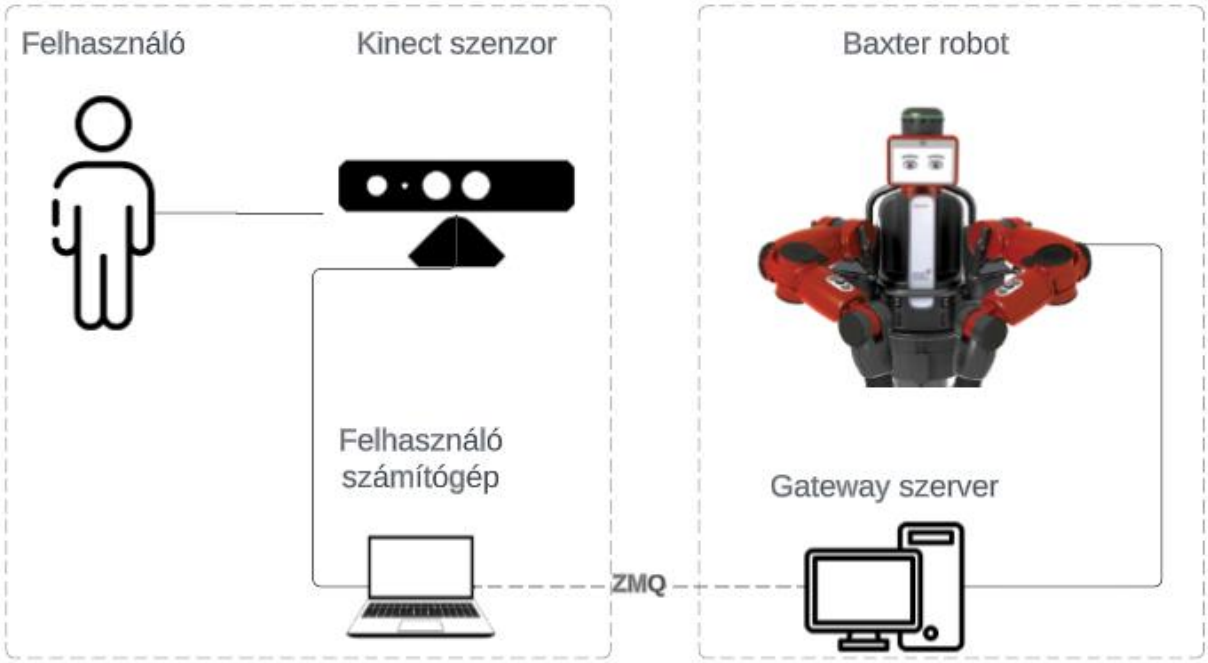
felismerésére szükséges egy Kinect v2 szenzor, aminek kezeléséhez szükséges a Kinect for Windows (version 2.0) fejlesztői csomag.

A rendszer fejlesztéséhez szükséges egy fejlesztői környezet használata, ami ebben az esetben a Microsoft Visual Studio 2022 és a Visual Studio Code volt. A rendszer C# és Python nyelven volt fejlesztve. Emellett verziókövetésre a Github rendszerét használtam. A rendszer működéséhez szükséges, hogy a rendszerkomponensek egy hálózaton legyenek (például a robot és a felhasználó számítógépe).

A rendszerben használt robotnak biztonságosnak kell lennie. Nem tehet kárt sem saját magában sem a körülötte lévő eszközökben vagy emberekben. Emellett a robotnak kell rendelkeznie egy vészkapcsolóval, amivel bármikor le lehet biztonságosan állítani a robotot.

5. A rendszer architektúrája

A rendszer követelményeiből adódóan a Kinect kamera és a robot párhuzamosan kell működjenek és egymással kommunikáljanak. Itt ütköztem bele a rendszer elemeinek kompatibilitási problémájába. Ahhoz, hogy az általunk használt Baxter ipari robottal kommunikálni tudjunk, szükségünk volt egy Python könyvtárra, amivel a robotot vezérelni lehet és egyre, amivel adatokat lehet kivenni belőle. A Kinect kamerából, pedig a Kinect SDK felhasználásával nyertem ki adatot, ami pedig egy C# applikációnak része. Ezt egy köztes gép beiktatásával oldottam meg. Ez a köztes gép direkt kommunikál a robottal és a rendszer megmaradt komponenseivel.



Ábra 2: A rendszer tervezett architektúrája

Ahogy az a fenti ábrán (2.) látható, a rendszer két részből áll. Az első, a felhasználói interfész, ami a felhasználóval van közvetlen kapcsolatban. A rendszernek ebbe a részébe tartozik a Kinect kamera és a felhasználó számítógépe, ami össze van kötve az előbb említett Kinect kamerával. A rendszer ezen része úgy van elképzelve, hogy a felhasználó mozdulatait a Kinect szenzor rögzíti, majd a hozzá kötött számítógép feldolgozza a szenzorból jött információkat. Ebben a részben kerülnek a kinyert információk átalakítása. Az így kapott információkat a felhasználó eszköze elküldi a rendszer másik részének.

A második rész a robot interfész. Ez áll magából a Baxter ipari robotból és egy köztes gépből, ami egyfajta hidat képez a robot és egyéb eszközök között (a továbbiakban gateway szerver). A gateway szerver szerepe, hogy kommunikálni tudjon a robottal és a felhasználói interfésszel párhuzamosan. Ez a szerver gyűjti be a felhasználótól érkező információkat és annak megfelelően parancsokat küld a robot felé. Amikor a robot sikeresen elvégezte a feladatot és visszajelez a szervernek, a szerver visszaigazolja a felhasználói interfész felé, hogy a feladat el lett végezve.

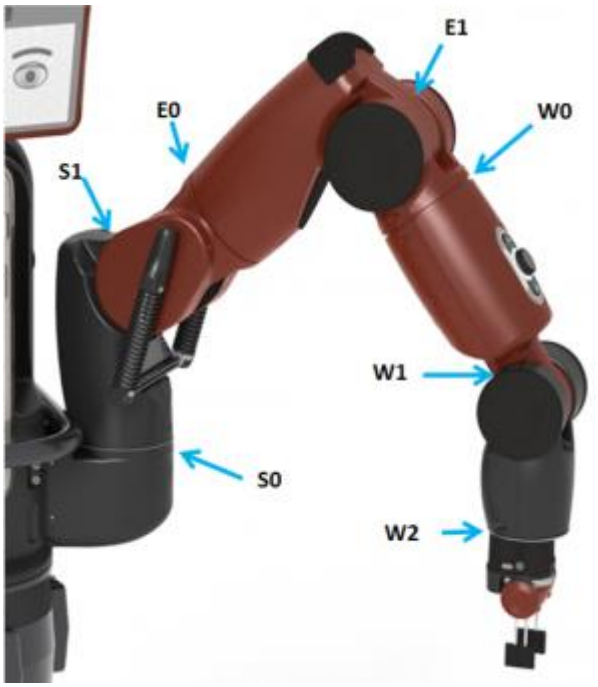
Emellett a szervernek publikáló szerepe is van. A robottól érkező publikált adatokat, állandóan továbbítja a feliratkozott kliensek felé.

5.1.Alkalmazott eszközök

5.1.1. Baxter Kooperatív robot

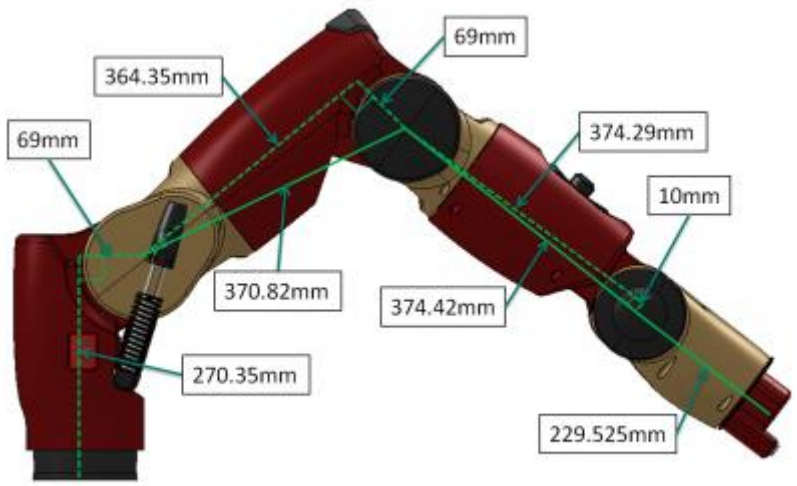
A Baxter egy két karral rendelkező kooperatív robot, amit a Rethink Robotics cég fejlesztett ki 2012-ben. Mindkét kar hét szabadságfokkal rendelkezik és minden csukló rendelkezik erő, nyomaték és pozíció érzékelőkkel. Ezek segítségével lokalizálni lehet a robotkar összes csuklóját és a csuklókra ható külső erők mérésével a robot tudja érzékelni, hogy ha a körülötte lévő eszközök az útjába lennének, vagy azokkal esetleg ütközne. Ezeknek köszönhetően a Baxter robot az emberek között is tud dolgozni, mivel hogyha kontaktba kerülne valakivel, ezt érzékelni tudja és leáll.

A Baxter robot rendelkezik egy szolid törzzsel, amire rá van illesztve két mobilis robotkar és egy „fej”, amin egy állapotjelző LED csík, egy kamera és egy kijelző található [8]. A robot mindkét karja megegyező tulajdonságokkal rendelkezik. Mindkét kar végén megtalálható egy kamera és egy végberendezés, az lehet vákuumos, vagy motorikus.



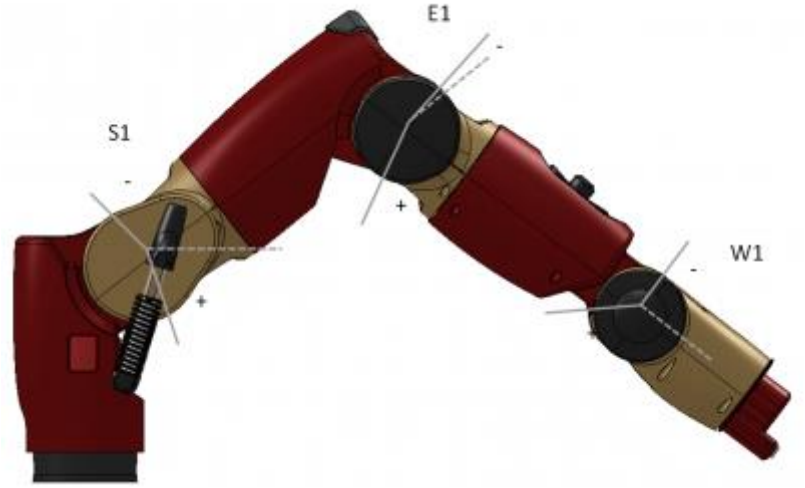
Ábra 3: A Baxter csuklóinak nevei [8]

Az 3. ábrán láthatóak az egyik karnak a csuklói és annak az elnevezései. Mindkét kar az emberi végtagokról vannak mintázva, ezért a csuklók nevei is ahhoz köthetőek (S0, S1 – shoulder, E0, E1 – elbow, W0, W1, W2 – wrist). Minkét karon megtalálható egy vezérlésre szolgáló kontroller is, amivel a robot saját kezelőfelületét tudjuk irányítani.



Ábra 4: A Baxter szegmenseinek hossza [8]

A 4.ábrán látható, a karok szegmenseinek a hossza, a megfogó szerkezet nélkül.



Ábra 5: A Baxter elhajló csuklói [8]

A kar két típusú csuklóból áll. Az egyik típus a hajló csuklók, ezek az S1, E1 és W1. Ezek a 5. ábrán láthatóak, ezeknek paraméterei pedig az 1. táblázatban vannak feltüntetve.

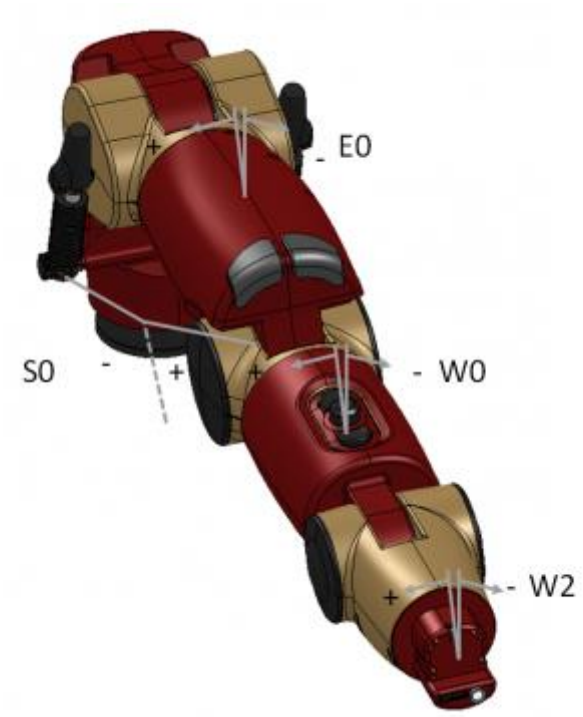
Csukló neve	Minimális elfordulás (szögben)	Maximális elfordulás (szögben)	Lefedett tartomány
S1	-123	+60	183
E1	-2.864	+150	153
W1	-90	+120	210

Táblázat 1: A Baxter elhajló csuklóinak paraméterei [8]

A másik típus, a forduló csuklók. Ezek a S0, E0, W0 és W2.

Csukló neve	Minimális elfordulás (szögben)	Maximális elfordulás (szögben)	Lefedett tartomány
S0	-97.494	+97.494	194.998
E0	-174.987	+174.987	349.979
W0	-175.25	+175.25	350.5
W2	-175.25	+175.25	350.5

Táblázat 2: A Baxter elforduló csuklóinak paraméterei [8]

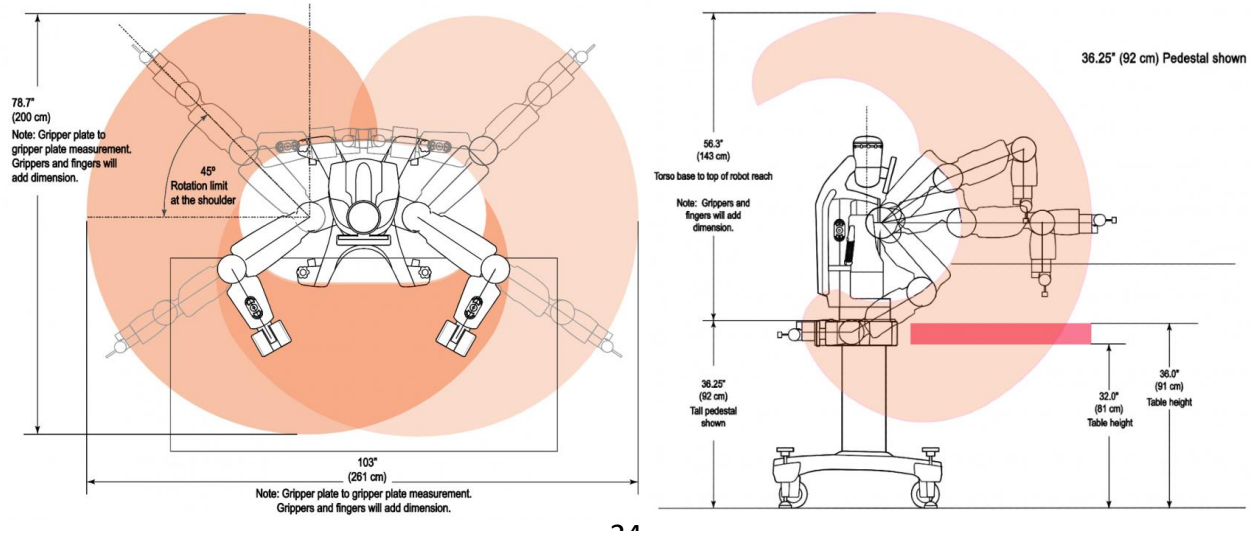


Ábra 6: A Baxter csavar csuklói [8]

Ezek a csuklók az 6. ábrán láthatóak, a paramétereik pedig a 2. táblázatban vannak feltüntetve.

A Baxter robot többféle szenzorral rendelkezik, amikkel érzékelni tudja a körülötte lévő környezetet. Ezek közé tartozik a mindkét robotkar végberendezése mellett található kamera és infravörös szenzor távolságmérésre, minden csuklón nyomaték mérő, gyorsulásmérő, egy kamera a robot fején mozgásérzékelők, amik meg tudják határozni a közelben lévő embereket.

A robot teljes szélessége, ha minkét kar ki van nyújtva, 261 cm (a megfogó szerkezetek nélkül) és 185 cm magas. A Baxter robotnak a teljes munkatere minimum 200*261 cm [9], ez az 7. ábrán van szemléltetve.

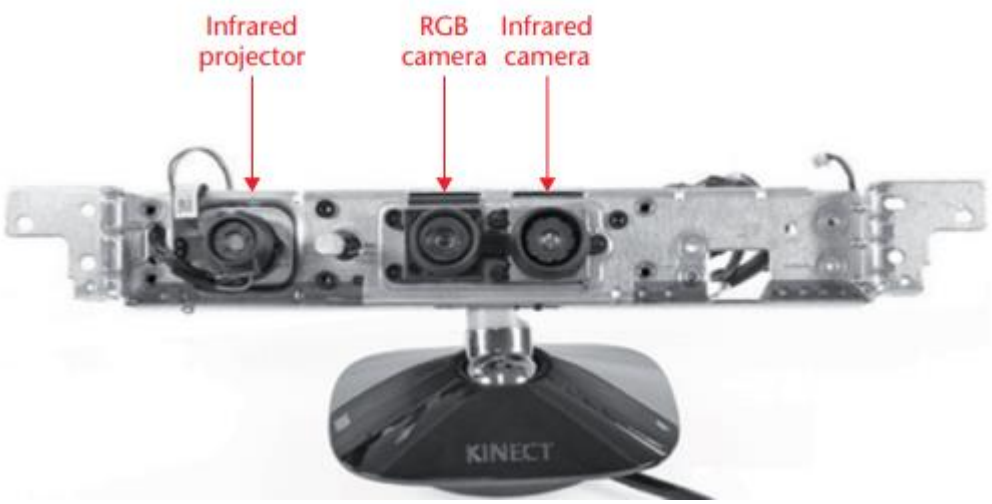


Ábra 7: A Baxter munkatere oldal- és felülnézetből [9]

5.1.2. Kinect Kamera

A Kinect kamera, a Microsoft által fejlesztett 3D szenzor. Az első generációt 2010 november 4.-én adták ki [10], elsősorban az Xbox 360 játékkonzolra. A szenzor megjelenése akkoriban nagy port kavart a fejlesztők közrében, egyre nagyobb lett az érdeklődés iránta.

A Kinect első generációja több érzékelővel felszerelt eszköz volt. Ahogy az 1.árbán látható, az RGB kamera mellett rendelkezett infravörös kamerával és egy infravörös projektorral. A kamerák mellett fellelhető volt a Kinect kamerán egy négy mikrofonból álló rendszer (8. ábra) [10].



Ábra 8: A Kinect szenzor kamerái [10]

A Kinect szenzor RGB kamerája egy 30 Hz-es és 640*480 pixel felbontású képet tudott nyújtani. Emellett támogatott nagyobb felbontást is. 1280*1024 pixeles felbontás esetében a nyújtott kép frissítési rátája 10 Hz-re csökkent. A Kinect 3D mélységi kamerája egy infravörös lézer projektorból és egy infravörös kamerából állt. Ez a két eszköz segítségével a Kinect képes volt egy mélységi térképet alkotni az érzékelt objektumokról. Ennek a szenzornak a felbontása 640*480 pixel volt, másodpercenként 30 képkockával. Ez a szenzor 0.8 m és 3.5m között tudott pontos mélységi adatokkal szolgálni.

A meglévő szenzorokkal a Kinect képes volt 3D-s képeket rögzíteni, képes volt mozgásérzékelésre, arcfelismerésre és a fejlett mikrofonrendszernek köszönhetően a hangvezérlést is támogatta. Ezek mellett számos másik funkciója is volt. A Kinectet használni lehetett kézmozdulatok érzékelésére, felismerésére, tárgyak felismerésére és követésére vagy akár belső terek és tárgyak 3D modellezésére [17].

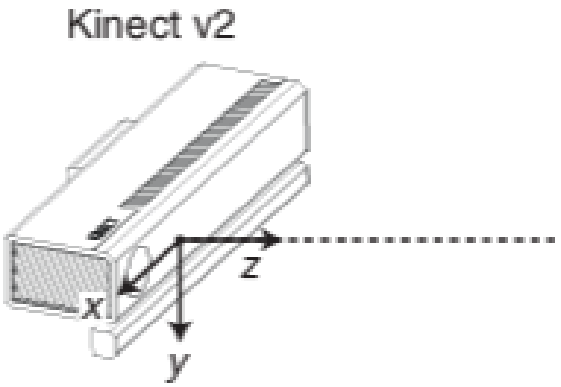
A Kinect kamerának viszont az egyik legnagyobb funkciója az emberi csontváz követése volt (továbbiakban skeletal tracking). A skeletal tracking egy összetett folyamat, amely során a rögzített kép alapján az emberi alakot felismerjük és annak különböző végtagjait és részeit jól meghatározva felismerjük és elkülönítsük. Ennek egyik alapvető feltétele, hogy a felhasználó testi adottságaitól függetlenül ugyanolyan pontosan kell működnie. [10]

A Kinect szenzor második generációja 2013-ban jelent mg és 2014 júliusa óta használják kutatási célokra. Az új generáció kinézetében hasonlított elődjére, viszont annál egy kicsit magasabb és szélesebb lett. A Kinect v2 szenzor az előző generációhoz képest fejlődött felbontásban. Az RGB kamera felbontása itt 1920*1080 pixel, a mélységi kamera felbontása pedig 512*424 pixelre nőtt és a mélységi képet nagyobb tartományban, 0.5m és 4.5m között tudta pontosan szolgáltatni.

A legnagyobb újítása viszont, hogy a mélységi kép megalkotásához nem az előző Kinect-ben használt strukturált fény technológiáját használták, mivel annak erős fényben romlott a pontossága. A Kinect v2-ben már ToF (Time of Flight) szenzorokat használtak az infravörös projektor és kamera helyett [18].

A ToF szenzor áll egy infravörös kibocsájtóból és mátrixosan elhelyezett infravörös szenzorokból. Ezek egy áramkörre vannak kötve, ami begyűjti a vissza érkező jeleket és azoknak a visszaérkezési idejükből kiszámolja, hogy az érzékelt tárgy mekkora távolságra van [19].

A Microsoft biztosított a kutatók számára egy könyvtárat a Kinect fejlesztéséhez. A Kinect Software Development Kit (SDK) először 2011-ben jelent meg. A Kinect SDK tartalmaz több funkciót, amivel a Kinect eszközökre való fejlesztést segíti. Ilyen a kamera kalibráció, az emberi test álló helyzetből való felismerése, az emberi test ülő pozícióból való felismerése, test, illetve kéz gesztusok felismerése, feldolgozása vagy 3D szkennelés [17]. A Kinect SDK 20 különböző emberi ízületet meg tud különböztetni. A Kinect SDK képes a térben pontokat meghatározni három koordináta alapján [18]. Az x-tengely a szélességet, az y-tengely a magasságot, a z-tengely pedig a mélységet adja meg (9. ábra).



Ábra 9: A Kinect SDK koordináta rendszere [18]

5.1.3. Vitruvius könyvtár

A Vitruvius [12] egy a LightBuzz által, a Kinect kamerákra fejlesztett szoftverkönyvtár, ami kiegészíti a Kinect szenzor lehetőségeit. AVitruvius könyvtár a Kinect SDK-val együtt működik. Ez a könyvtár felhasználja a Kinect kamera adatait és megkönnyíti annak feldolgozását. A Vitruvius könyvtár funkciói közé tartoznak:

- mozdulatok és testtartások felismerése

- test érzékelés, kirajzolás, magasság számítás

- végtagok szögeinek kiszámítása szögben vagy radiánban

- gesztusok felismerése

- arcjegyek felismerése, arckifejezések

- jelenet elemzés, háttér feldolgozás, objektumok kiválasztása

5.1.4. Robot Operating System

A Robot Operating System (a továbbiakban ROS) egy nyílt forráskódú keretrendszer, amely széles körben elterjedté vált a robotika területén, sokoldalúsága és moduláris architektúrája miatt. A Stanford Egyetem kutatói által fejlesztett, majd 2007-ben bevezetett keretrendszer egy köztes szoftverként működik (middleware) [11]. A ROS egy olyan rendszer, amely bonyolult robot alkalmazások fejlesztését segíti elő, rugalmas és elosztott architektúrájával.

A ROS lévén egy a robotikában használt keretrendszer, többféle könyvtárat és eszközt gyűjt egybe, amivel többféle típusú és felépítésű robotra lehet ennek segítségével alkalmazásokat

fejleszteni [15]. Emellett több eszközt is biztosít a szenzorok használatára, különböző komponensek irányítására, folyamatok monitorizálására [15].

A ROS egy publish/subscribe üzenetátviteli modellt használ, aminek használatával a rendszerelemek hatékonyan tudnak egymás között kommunikálni. Ez az architektúra használata nagyban járul hozzá, a keretrendszer moduláris felépítéséhez, így könnyebben lehet szoftver, illetve hardver elemeket integrálni [11][16].

Az üzenetátviteli rendszer, a ROS egyik legfontosabb jellemzője. A rendszerben egymással kommunikáló komponensek, csomópontokként (node) vannak tekintve. Ezek között számos üzenettípus támogatott, mint például a szenzoradatok, vezérlő parancsok, egyedi üzenetek. Így biztosítva a kompatibilitást különböző robot platformok, szenzorok vagy aktuátorok között [16].

A ROS peer-to-peer csatlakozáson alapul, a rendszerben lévő entitások között [11]. Egy ROS-ra épült rendszerben több folyamat fut több gazda eszközön és ezek peer-to-peer topológia szerint vannak összekötve. Ennek előnye, hogy a rendszer erőforrásai jobban el vannak osztva és nincsen szükség egy központi szerverre, hogyha több ROS-t használó gép (például robot) van egy hálózatba kötve.

Egy ROS hálózatot a számítási gráf (Computational Graph) ír le. Ez a ROS folyamatok peer-to-peer hálózata [16]. Egy ROS hálózatban a fő elem a ROS Master, ami megadja a hálózatban levő csomópontoknak a név regisztrációt és biztosítja a keresést a csomópontok között. A hálózatban a csomópontok tudnak egymással direkt kommunikálni az „üzeneteken” (messages) keresztül. Ezek adatstruktúrák, amiket a csomópontok egymásnak küldenek. Továbbá a hálózat támogat publish/subscribe mintájú adatcserét is, a „témákon” (topic) keresztül. Emellett a ROS hálózat támogat many-to-many mintájú kommunikációt is a „szolgáltatások” (services) révén.

A ROS keretrendszernek kiadtak egy újra gondolt, második generációt is, ami az elő generáció biztonsági és strukturális kompromisszumait hivatott kijavítani. Mivel az első generáció főként kutatási célból lett fejlesztve, idővel megmutatkoztak annak tervezéséből fakadó hátrányai [15], megkötései. Erre megoldásként került kiadásra a ROS 2, először 2015-ben [20] az Open Robotics szervezet által.

5.1.5. ZMQ

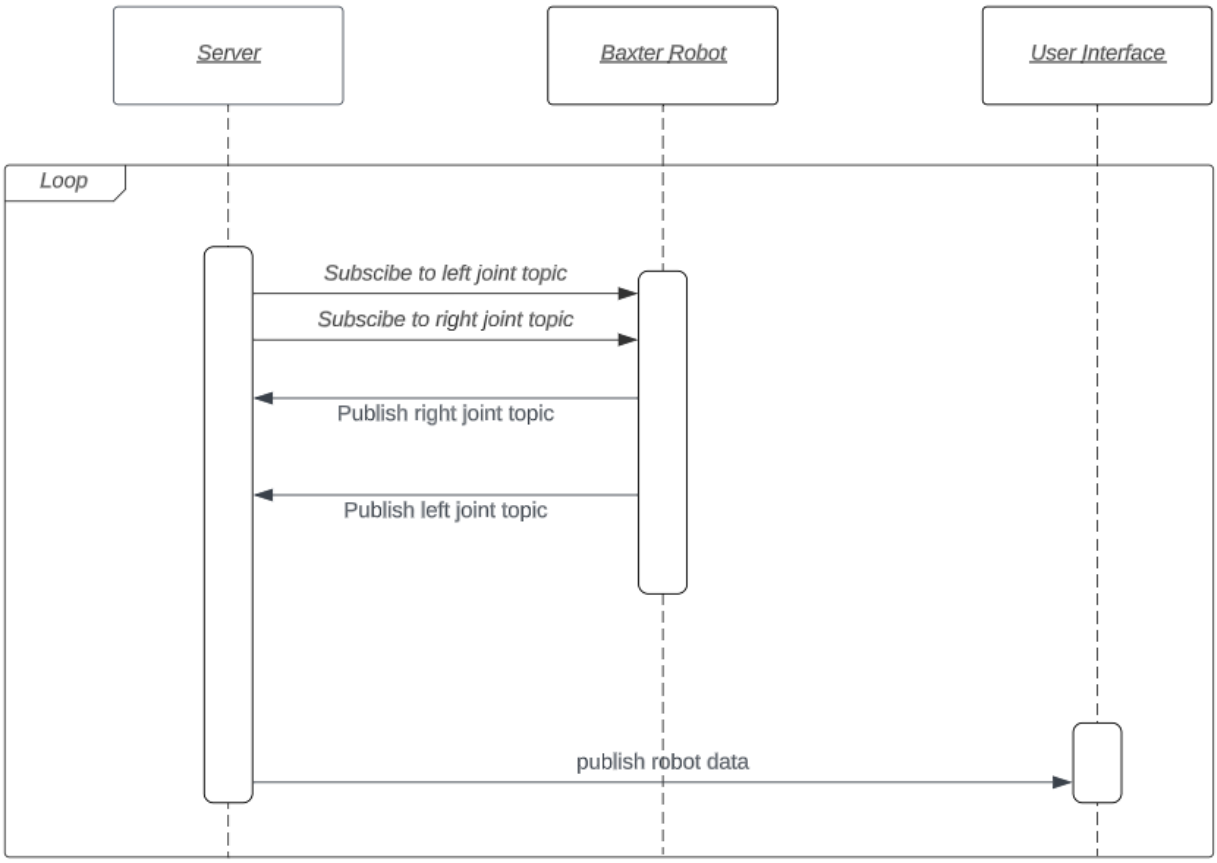
A Zero MQ (ZMQ), egy nyílt forráskódú kommunikációs technológia, ami lehetővé teszi az alkalmazások és folyamatok közötti aszinkron üzenetküldést [13]. A ZMQ célja, hogy egy

hatékony kommunikációs eszközt nyújtson, elosztott rendszerek fejlesztéséhez, különböző programozási nyelveket támogatva. A ZMQ különböző kommunikációs mintákat támogat, ilyen a publish/subscribe, a request/reply, push/pull vagy a many-many multicast. A ZMQ alapvető kommunikációs egysége a socket, [21] ami a kommunikációban a végpontot jelenti. Az alkalmazások socketeket használnak az üzenetek küldésére és fogadására.

A ZMQ több kommunikációs protokollt támogat, mint például Transmission Control Protocol (TCP), User Datagram Protocol (UDP) vagy Inter-Process Communication (IPC). A ZMQ aszinkron és esemény vezérelt működésű, ami lehetővé teszi az alkalmazásoknak vagy folyamatoknak, a párhuzamos üzenet kezelést és az üzenetek folyamatos feldolgozását, a folyamat blokkolása nélkül. A ZMQ kommunikációs technológia előnyei közé tartozik a hibatűrés, a többféle programozási nyelv támogatása és a skálázhatóság.

5.2. Elemek közti kommunikáció

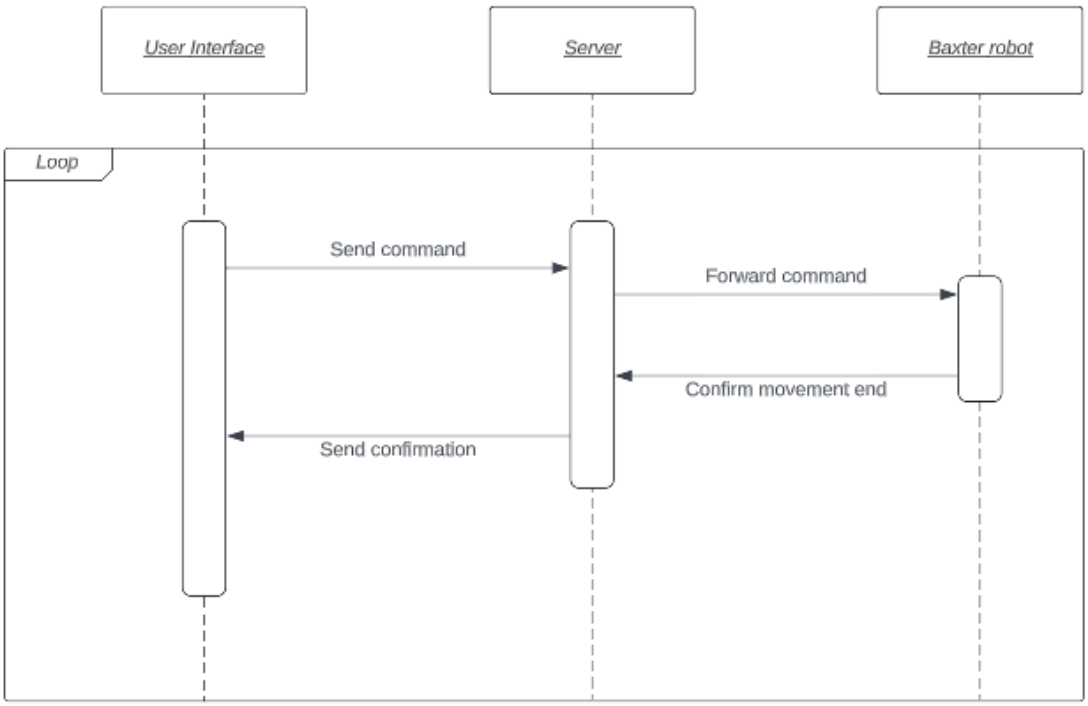
A rendszerkomponensek közötti kommunikációt a ZMQ nyílt forráskódú keretrendszer (lásd 5.5. fejezet) használatával valósítottam meg. A gateway szerveren párhuzamosan két socket-en történik a kommunikáció. Az egyik socket publish/subscribe mintát követ, a másik request/reply. Ezt két párhuzamos szálon oldottam meg. A publish/subscribe típusú socket-en folyamatosan meg van osztva a robot karjainak adatai. Ilyenek a végtagok ízületeinek szögei, gyorsulásai, illetve a végberendezés pozíciója, állapota és orientációja.



Ábra 10: Az adatokat folyamatosan publikáló szál szekvencia diagramja

A 10. ábrán látható a publikáló szál szekvencia diagramja. Ez először lekéri a Robottól a szükséges adatokat, amit az visszaküld. Ezután innen publikálódik az összes feliratkozott kliens felé az adat.

A második szál, a klienseket hallgatja. Ez egy Request/Reply típusú szál, mivel fontos, hogy a kliens visszaigazolást kapjon, hogy a feladat elvégződött és csak azután tudjon újabb parancsot küldeni.



Ábra 11: A parancsokat fogadó és továbbító szál szekvencia diagramja

Ahogy az a 11. Ábrán látható, a második szál szekvencia diagramja. Ez a szál begyűjti a felhasználótól érkező parancsokat és aszerint vezérli a robot mozgását. Amikor a robot az aktuális mozdulatot befejezi, visszajelez a szervernek. Ezt követően a szerver visszaigazolást küld a felhasználó felé, így az tud újabb parancsot küldeni.

A komponensek közötti kommunikációban fontos elem volt, egy üzenetküldési protokoll megfogalmazása. Ezt a protokollt kellett azután használni a komponensek közötti információ cserére.

A tervezett protokollban az üzenetek JSON formátumot követnek. A szervernek küldött parancs első eleme határozza meg, hogy melyik robotkarra vonatkozik a parancs, jobb vagy bal. Az ehhez társított érték, megint egy JSON értékpá. Ennek az első eleme a parancsot jelöli. Az ehhez társított érték (vagy értékek) a paramétereket adják meg a parancs elvégzéséhez.

```
"right_arm",
{
    { "move_to",
        {
            {"right_w0", 0.172},
            {"right_w1", -0.039},
            {"right_w2", -0.156},
            {"right_e0", -0.069},
            {"right_e1", 2.172},
            {"right_s0", -0.682},
            {"right_s1", -0.643}
        }
    }
}
```

Ábra 12: Egy üzenet felépítése a felhasználó esetében

A 12. ábrán látható annak a parancsnak a felépítése, amelyik megadott szögekre állítja be a Baxter robot jobb karjának az csuklóit. Az első paraméter adja meg a robot karját, a második a parancsot, az ez utáni értékpárok pedig a szögeket adja, amikre be kell állítani a csuklókat.

A szerverről érkező adat felépítése is hasonló. A különbség itt az, hogy a szerver nem parancsot küld. A küldő/fogadó szál esetében csak egy visszaigazolás, hogyha sikerült vagy nem a parancs végrahajtása, a publikáló szál esetében pedig a parancs helyett az adott kar adatainak típusa van. Egy a jobb kar adatait tartalmazó üzenet a 13. ábrán látható.

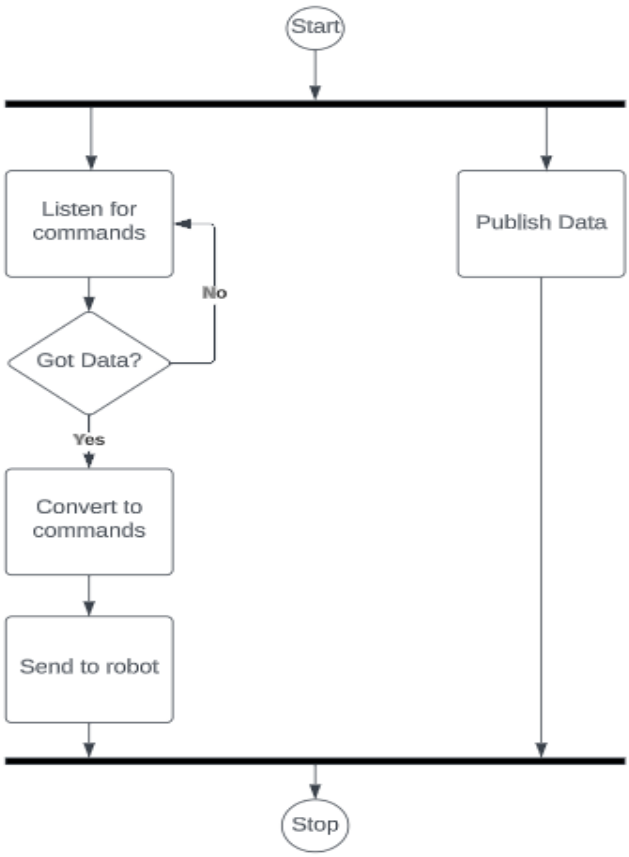
```
'arm_right',
{
    'velocities':
    {
        'right_s0': -0.0023561944897503642,
        'right_s1': 0.010995574285501701,
        'right_w0': 0.0035342917346255468,
        'right_w1': -0.009817477040626518,
        'right_w2': -0.01060287520387664,
        'right_e0': 0.01021017612225158,
        'right_e1': 0.01531526418337737
    },
    'angles':
    {
        'right_s0': -0.8156942839580688,
        'right_s1': 0.46096122675956686,
        'right_w0': 0.34476218207724674,
        'right_w1': 1.0737865515197895,
        'right_w2': 2.8290440680576743,
        'right_e0': 0.47016511148687934,
        'right_e1': 0.4233786974563742
    }
}
```

Ábra 13: A szerverről jövő üzenet formája

5.3. A rendszer megvalósítása

5.3.1. Gateway szerver

Az első lépés, hogy a robottal kommunikálni tudjunk. Ez azt feltételezi, hogy nem csak információkat kell tudjunk lekérni, hanem parancsokat küldeni a robotnak. Ebből kifolyólag olyan eszközre volt szükségünk, amit be tudunk ékelni a felhasználó eszköze és maga a robot közé, így gyakorlatilag megoldva a felmerülő kompatibilitási problémákat, ugyanis így a megvalósítani kívánt applikációtól függ, hogy a felhasználónak milyen eszköz kell, nem kell a robottal összeilleszteni. A szerver lényegében egy híd a végpontok között. A feladata, hogy felismerje a robotot és a kliens (jelen esetben a felhasználó) oldal felé továbbítson minden információt, ami a robottól érkezik. Emellett persze a kliens oldalt is figyelnie kell, hogy az innen érkező parancsokat feldolgozza és átalakítsa olyan formába, amit a robot megért és végre tud hajtani.



Ábra 14: A gateway szerver működési elve

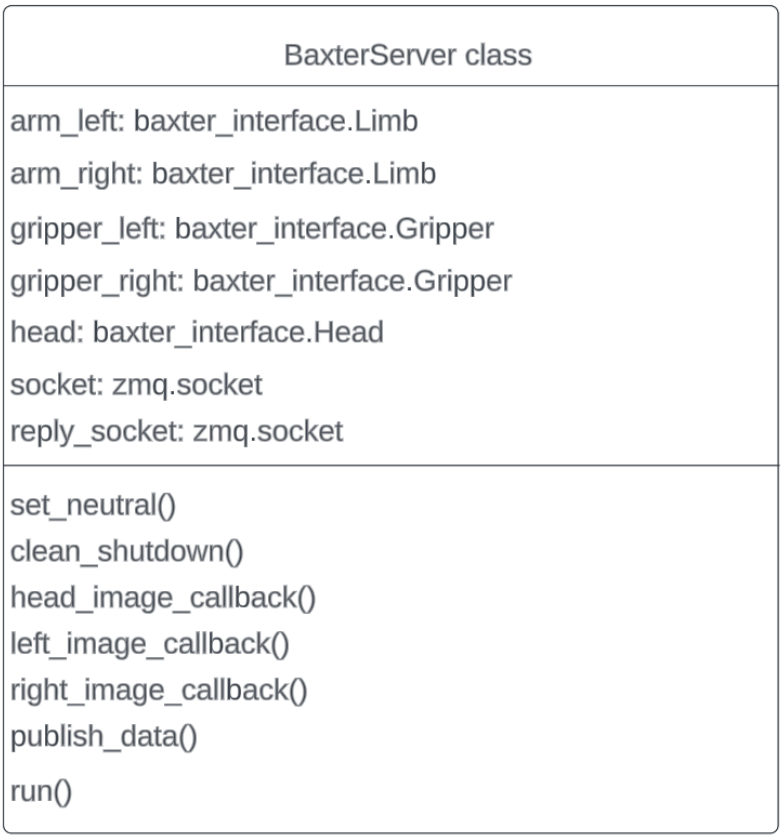
Ahogy a 14. ábrán látható, a szervernek egyszerre két szálon kell működnie. Az első szál feladata, hogy folyamatosan közzé tegye a robot által nyújtott információkat. Ilyenek a csuklók aktuális szögei, pozíciói, minden csuklón lévő nyomaték, a kamerák képe vagy a megfogó szerkezetek állapota (nyitott vagy zárt).

A második szál szerepe, hogy a kliens/ kliensek üzeneteit fogadja. Ennek a szálnak a feladata hallgatni (listening), hogyha a klienstől érkezik-e üzenet. Abban az esetben, hogyha érkezett üzenet, azt fel kell dolgoznia. Ezután pedig a parancsot elküldi a robotnak. Amikor a robot elvégezte a feladatot, az visszajelez a szervernek, ami majd visszajelez a felhasználónak.

A szervernek emellett fel kell a robotot ismernie. Mivel az általunk használt robotnak, a Baxternek az operációs rendszere ROS alapú, ezért kézenfekvő volt ennek a technológiának a lehetőségeit alkalmazni. A ROS egy robot programozásra fejlesztett, nyílt forráskódú keretrendszer (lásd 5.1.4 fejezet). Ennek segítségével hozzá fértem a robot által publikált adatokra. Egy ROS hálózaton belül az entitások csomópontokként (node) vannak tekintve és a csomópontok tudnak egymás között kommunikálni. A ROS egyik kommunikációs protokollja, a publish/subscribe elvet követi. Ez azt jelenti, hogy a robot (jelen esetben a Baxter), állandóan publikál adatokat úgynevezett „topicok”-ban. A szervernek először inicializálnia kell egy ROS csomópontot, ezután fel kell iratkoznia azokra a topic-okra, amelyeket továbbítani akar.

5.3.2. Gateway szerver komponensei

A 15. ábrán látható a BaxterServer osztálydiagramja. Az osztály rendelkezik egy jobb és bal végtag adattaggal, amik a robot jobb és bal karját vezérlik. Hasonlóképpen vannak elérve a karok végberendezései és a robot feje. Ezek mellett vannak a socket illetve reply_socket paraméterek, amiken keresztül majd a felhasználóval tud kommunikálni.



Ábra 15: A Gateway szerver osztály diagramja

A szerver osztályt először inicializáltam, ez a 16. ábrán látható. Itt az első lépés volt, egy ROS csomópont inicializálása. Ez, azt a célt szolgálja (lásd 5.1.4 fejezet), hogy a robot által publikált információkat el tudja a szerver érni. Ezt követően a robotot engedélyezett állapotba kellett helyezni, hogy lehessen vezérelni. Ezután inicializáltam a robot elemeire mutató adattagokat és feliratkoztam a robot „topic-jaira”. Ezt követően meghívódik az init_stream() függvény, ahol a socket-ek inicializálódnak.

Ezután következett, hogy a robotot vezérelni lehessen. Erre a baxter_interface python könyvtárat használtam, ami egy a Rethink Robotics által fejlesztett, kifejezetten a Baxter robot irányítására szolgáló könyvtár. Ebben implementálva vannak a robot különböző elemei, mint a jobb és bal

```
33      def __init__(self):
34
35          print("Initializing node... ")
36          rospy.init_node("streamer_node")
37          self._rs = baxter_interface.RobotEnable(CHECK_VERSION)
38          self._init_state = self._rs.state().enabled
39          print("Enabling robot... ")
40          self._rs.enable()
41
42          self.arm_left = baxter_interface.Limb('left')
43          self.arm_left.set_joint_position_speed(1.0)
44
45          self.arm_right = baxter_interface.Limb('right')
46          self.arm_right.set_joint_position_speed(1.0)
47
48          self.gripper_left = baxter_interface.Gripper('left')
49          self.gripper_right = baxter_interface.Gripper('right')
50
51          self.head = baxter_interface.Head()
52          self.bridge = CvBridge()
53          head_image_topic = 'cameras/head_camera/image/'
54          rospy.Subscriber(head_image_topic, Image, self.head_image_callback)
55          left_image_topic = 'cameras/left_hand_camera/image/'
56          rospy.Subscriber(left_image_topic, Image, self.left_image_callback)
57          rigth_image_topic = 'cameras/right_hand_camera/image/'
58          rospy.Subscriber(rigth_image_topic, Image, self.right_image_callback)
59          self.init stream()
```

Ábra 16: A BaxterServer osztály inicializálása

```
107     def publish_data(self):
108         while not rospy.is_shutdown():
109             data = {
110                 'arm_left': {
111                                     'angles': self.arm_left.joint_angles(),
112                                     'velocities': self.arm_left.joint_velocities()
113                                 },
114                 'arm_right': {
115                                     'angles': self.arm_right.joint_angles(),
116                                     'velocities': self.arm_right.joint_velocities()
117                                 },
118                 'left_endpoint_position':{
119                                     'data':self.arm_left.endpoint_pose()
120
121                 },
122                 'right_endpoint_position':{
123                                     'data':self.arm_right.endpoint_pose()
124
125                 },
126                 'gripper_right': { },
127
```

Ábra 17: A publish_data() függvény

kar, adott kamerák, a jobb és bal megfogó eszközök vagy a robot feje. Ha ezek inicializálva vannak, a robotot egyszerűen tudjuk vezérelni.

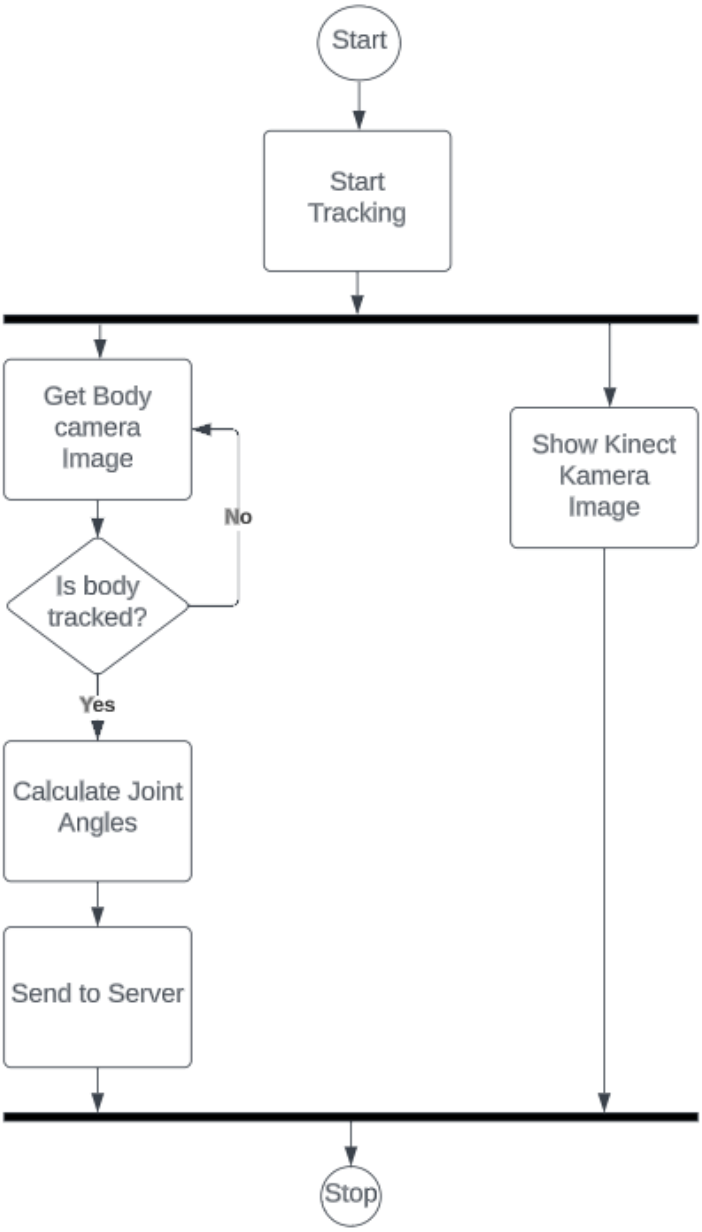
Az osztálynak a callback függvényei lekérik az adott kamera képét, amit a robot publikál. A publish_data() függvény pedig a felépített adatokat küldi ki a kliensek felé (17. ábra).

A parancsok fogadása és feldolgozása a self.run() függvényben történik. Itt elindul a szál, ami szórja az adatokat a későbbiekben és a szerver elkezdi figyelni a klienseit. Abban az esetben hogyha a szerver üzenetet kap, az üzenet tartalma alapján a megfelelő parancsot küldi a robotnak, ezután pedig visszaigazolást küld a kliens felé.

Ez a funkció egy, a fő szálon futó, ciklus, ami amíg a szerver le nem áll, addig figyeli az erre szánt kommunikációs végpontot (socketet).

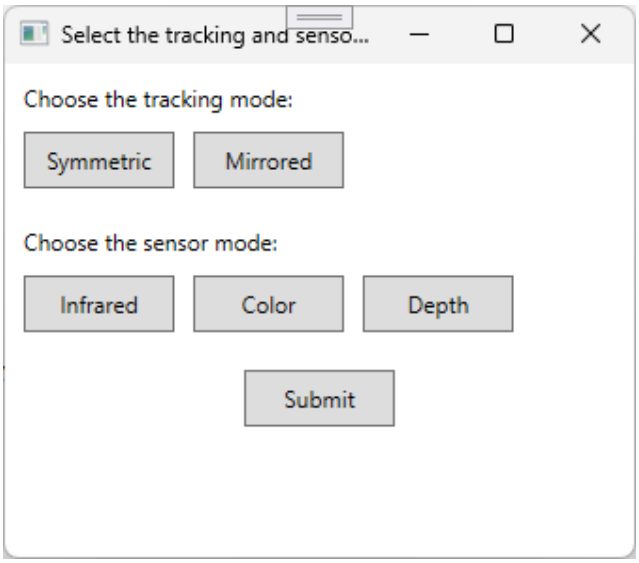
5.3.3. Felhasználói interfész

A második lépés, hogy létrehozzuk a rendszer azon részét, amellyel a felhasználó közvetlen interakcióban van. Ez a rész tartalmazza a Kinect szenzort, és az alkalmazást, ami azt feldolgozza. Ennek a csomópontnak az elméleti működési elve a 18. ábrán látható. Ennek is két szálon kell működnie. Az első szálon valósul meg a kamera kivetítése a képernyőre. A második szál gyűjti be a Kinect által rögzített felhasználónak az adatait. Az innen kinyert szögeket megfelelteti a robot értékeivel és a szervernek elküldi azokat. Fontos megjegyezni, hogy addig, az algoritmus blokkolódik, ameddig az egy testet nem érzékel a Kinect kamera érzékelési terében.



Ábra 18: A felhasználói interfész működési elve

Az alkalmazás rendelkezik egy egyszerű kezelőfelülettel, ami az alkalmazás indításakor feldob egy ablakot, ahol a felhasználó kiválasztja, hogy milyen formában szeretné, ha a robot lekövetné a mozgását és hogy a Kinect szenzor melyik kamerájának szeretné a képét látni. A követés típusára kettő lehetőség van: vagy szemben áll a robottal és akkor tükrözve kell a robot kövesse, vagy egy irányba néz a robottal, ez esetben a robot ugyanúgy kell mozogjon, mint a felhasználó (nem tükrözve). Három féle kameraképet lehet megjeleníteni: Infravörös (Infrared), mélységi (Depth) és RGB kép (Color). Az előugró ablak a lenti 19. ábrán látható.

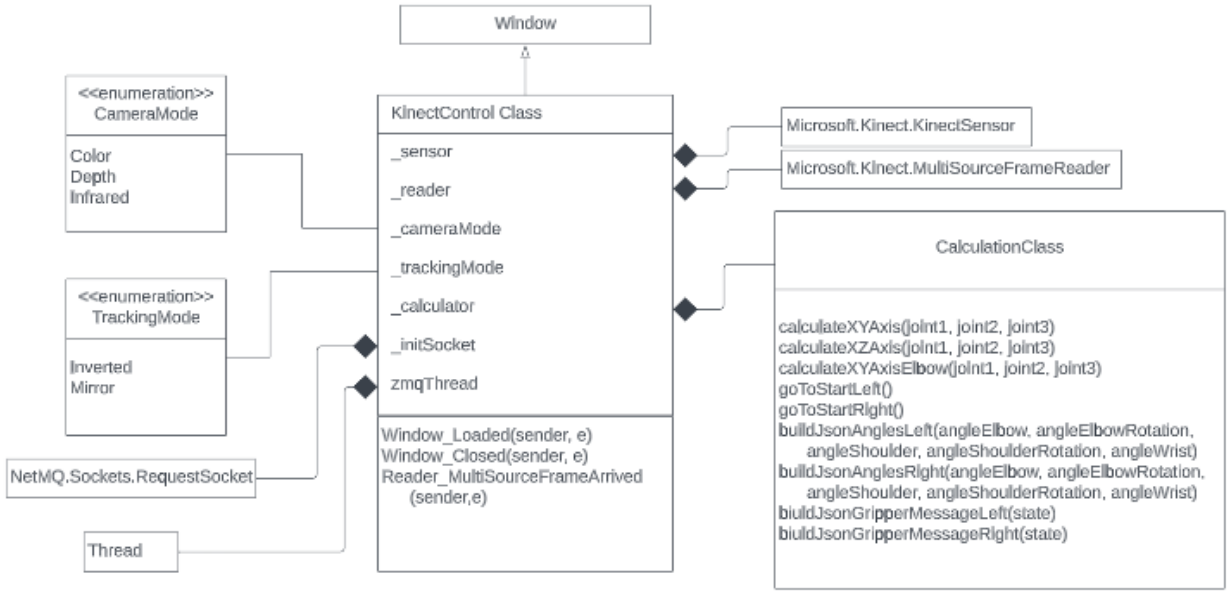


Ábra 19: Előugró ablak

Abban az esetben, ha a felhasználó nem választ semmit, hanem kilép, az alkalmazás leáll. Abban az esetben, ha ki lett valamelyik opció választva, előjön egy ablakban a Kinect szenzor RGB kamerájának a képe. Ekkor a robot egy kezdeti állapotba helyezi a karjait és a felhasználó követése elindul.

Ez után jelenik meg a kamera képét megjelenítő ablak. Így a felhasználó láthatja, amit a kamera lát. Ez az ablak egy 1920*1080 pixel felbontású kamera képből áll, amin a felhasználó által kiválasztott kamera kép jelenik meg.

5.3.4. A felhasználói interfész komponensei



Ábra 20: A felhasználói interfész osztálydiagramja

A 20. ábrán látható a felhasználói rész osztálydiagramja. Az applikáció alapjául egy nyílt forráskódú GitHubon megtalálható kódot használtam, ami a [14] -es linken megtekinthető. A rendszer ezen része egy WPF applikáció, aminek a vizuális része tartalmaz egy 1920*1080 méretű kép objektumot. Arra van a felhasználó által választott kamera képe kivetítve. A számítógéphez csatlakoztatott Kinect szenzorhoz a _sensor adattag tartalmazza a referenciát, ezzel lehet hozzáférni. A kamera aktuális képét a _reader adattagban mentettem el. Ebben mindig az aktuális képkocka van elmentve. Ezt a Reader_MultiSourceFrameArrived eseménykezelő metódussal oldottam meg. Ebben a függvényben a kiválasztott kameramód függvényében tettem ki a kamera képét a megjelenő ablakba. Emellett az osztály rendelkezik egy-egy eseménykezelő függvénnyel azokra az esetekre amikor az ablak betöltődik és amikor az bezáródik, a Window_Loaded és a Window_Closed függvények. Az osztálynak van egy zmqThread adattagja is, amin majd később a szerverrel való kommunikáció és a felhasználó mozgásának felismerése fog futni.

Amikor a KinectControl osztály inicializálódik, a szükséges paraméterek értéket kapnak, ilyen a _sensor vagy a _reader. Itt jön létre a felugró ablak is, amiben megtalálható öt gomb két sorban, itt lehet kiválasztani, hogy a Kinect kameráiból melyiket akarja a felhasználó látni és hogy hogyan szeretné, ha a robot követné a mozgását. Mielőtt a felhasználóról akármilyen mozgásadatot gyűjtenénk, a robotot mindkét karját egy előre meghatározott kezdő pozícióba

küldjük. Ehhez a CalculationClass osztályt használtam, amiben megvannak a számolásokhoz és az üzenetek felépítéséhez szükséges metódusok.

Az alkalmazás indításakor a robotot egy kezdeti állapotba kell helyezni, ahonnan biztonságosan el tudja kezdeni a követést. Erre a CalculationClass egyik metódusát használtam, amiben a szervernek küldök egy üzenetet a megfelelő szögekkel, amire a robotot be kell állítani. A függvény a 21. ábrán látható.

```
83      public string GoToStartRight()
84      {
85          var cmd = new Dictionary<string, Dictionary<string, object>>
86          {
87              {
88                  "right_arm", new Dictionary<string, object>
89                  {
90                      { "move_to",  new Dictionary<string, object>
91                          {
92                              {"right_w0", 0.172},
93                              {"right_w1", -0.039},
94                              {"right_w2", -0.156},
95                              {"right_e0", -0.069},
96                              {"right_e1", 2.172},
97                              {"right_s0", -0.682},
98                              {"right_s1", -0.643}
99                          }
100                     }
101                 }
102             }
103         };
104         return JsonConvert.SerializeObject(cmd);
105     }
```

Ábra 21: A GoToStartRight függvény, ami a jobb kart kezdeti állapotba helyezi

A második szálon fut ezzel párhuzamosan a Kinect kamera képének a feldolgozása. Ehhez először szükségem volt egy referenciára a Kinect szenzorhoz, a szálon belül, illetve egy referenciára a Kinect body kamerájához. Az innen kapott képről kimentettem egy listába az összes érzékelt emberi alakot és azok közül az elsőt vettem figyelembe. Lévén, hogy a Kinect egyszerre hat emberi alakot is tud követni (lásd 5.2. fejezet), ezért fontos, hogy a kamera terében egy felhasználó legyen, mert ellenkező esetben a felhasználó követése nem lesz pontos vagy akár a robot nem várt mozdulatokat tehet meg. Hogyha megérkezett a kamerakép és talált rajta emberi alakot, akkor az algoritmus a talált adatokból megpróbál szögeket számolni. Erre a célra a felhasználó váll, könyök, illetve csukló elhelyezkedéséből nyert szögeit használtam. A szögeket kétféle módszerrel számoltam ki. Ez összesen négy szabadságfokot biztosít (lásd 3. fejezet), a lehetséges hétből. Ahhoz, hogy az emberi kar szegmenseiből szögeket tudjak számolni, le kellett vektorokat képezzek a térben, az ízületek, illetve csontokból. Ezt a szempontot figyelembe véve a szögeket két típusra osztottam.

Az egyik típus az elhajló ízületek. Ilyen a váll, a könyökkel és csípővel bezárt szöge, vagy a könyök, vállal és csuklóval bezárt szöge. Ezeket egy külső könyvtár, a Vitruvius (lásd 5.3. fejezet) segítségével számoltam ki. Ennek a könyvtárnak az Angle metódusa tartalmazza az ízületek vektorokká való leképezését és a szög kiszámítását. Ez a függvény visszatérít egy 0 és 180 fok közötti értéket, ami a mért szög fokban való megfelelője.

```
14      public double CalculateAngleXYAxis(Joint joint1, Joint joint2, Joint joint3)
15      {
16          return joint2.Angle(joint1, joint3);
17      }
```

Ábra 22: Szög számítás a Vitruvius könyvtár Angle függvényével

Ahogy a 22. Ábrán látható, a könyvtár használata annyiból áll, hogy a szög középső pontjának egy kiegészítő függvényét hívjuk le. Ez pedig kiszámolja és visszatéríti a szöget, fokban kifejezve.

A másik típus az elforduló ízületek voltak. Ezek olyan ízületei a robotnak, amelyeknek kiszámításához bonyolultabb műveletre volt szükség. Mivel ennek az elmozdulásnak a dimenziója nem egyezett meg a Kinect kamera által látott két dimenzióval, másképp kellett számolni. Az előbb említett Vitruvius könyvtár első sorban a kamera síkjában tud pontosan számolni, más eszközt kellett használnom. Az előző szögek esetében elég volt a hosszúsági és szélességi (x, y tengely, lásd 5.2. fejezet) tengely alapján számolni, míg ezek az ízületek esetében szükséges a mélységi tengely (z tengely) használata is. Ezt a feladatot már nem tudtam a fentebb használt függvénnyel, ezért egy saját megoldást kellett alkalmazzak. Ahhoz, hogy szöget tudjak számolni szükség van három pontra, amiből két vektort tudok képezni. Jelen esetben ez a három pont a felhasználó gerincét jelölő térbeli pont, a felhasználó jobb vállát jelölő pont és a felhasználó jobb könyökét meghatározó térbeli pont. Ebből képezünk két vektort. A váll mélységi szögeinek kiszámításához használt vektorok a 23. ábrán láthatóak.

Ábra 23: A váll mélységi szögének számításához használt vektorok

Mivel a felhasználó vállának elfordulása térbeli mozgás, az egyik tengely, ami szerint számolunk, az a Kinect z tengelye így meg kell választani a másikat. Mivel az általunk érdekelt mozgás vízszintes, ezért a másik tengelynek a Kinect koordinátarendszerének x tengelyét választottam. Így tehát megvolt a dimenzió, ami szerint ki tudtam az elfordulás szögét számolni.

Ezután kiszámoltam a pontokból képezett vektorok skaláris szorzatát és a vektorok hosszait. A skaláris szorzatot és a vektorok hosszait felhasználva ki tudtam számolni a szög koszinuszát (a skaláris szorzat elosztva a hosszak szorzatával kapjuk meg a koszinuszt). Ebből pedig arkuszkoszinuszzal megkaptam a szöget radiánban, ezt pedig átalakítottam fokká.

```
19      public double CalculateAngleXZAxis(Joint joint1, Joint joint2, Joint joint3)
20      {
21          CameraSpacePoint position1 = joint1.Position;
22          CameraSpacePoint position2 = joint2.Position;
23          CameraSpacePoint position3 = joint3.Position;
24
25          double[] vector1 = new double[] { position2.X - position1.X, 0, position2.Z - position1.Z };
26          double[] vector2 = new double[] { position3.X - position2.X, 0, position3.Z - position2.Z };
27
28          double dotProduct = vector1[0] * vector2[0] + vector1[1] * vector2[1] + vector1[2] * vector2[2];
29          double magnitude1 = Math.Sqrt(vector1[0] * vector1[0] + vector1[1] * vector1[1] + vector1[2] * vector1[2]);
30          double magnitude2 = Math.Sqrt(vector2[0] * vector2[0] + vector2[1] * vector2[1] + vector2[2] * vector2[2]);
31
32          double angleInRadians = Math.Acos(dotProduct / (magnitude1 * magnitude2));
33          double angleInDegrees = angleInRadians * 180.0 / Math.PI;
34
35          return angleInDegrees;
36      }
```

Ábra 24: A mélységi szög kiszámítása

Ahogy az a 24. ábrán látható, a metódusnak megadtam három, a Kinect által érzékelt ízületet (a fenti esetben a gerinc, a jobb váll és a jobb könyök voltak ebben a sorrendben). Ezeknek a térbeli pontoknak kimentettem a koordinátáikat egy-egy CameraSpacePoint típusú lokális változóba, majd ezek alapján képeztem a két vektort, amikkel a szöget számoltam. A két vektor skaláris szorzatát a dotProduct változóba tároltam, a vektorok hosszait pedig a magnitude1, illetve

magnitude2 változókba. Ezután kiszámoltam a szöget radiánban a fent említett módszerrel és átalakítottam fokba, a függvény pedig azt téríti vissza.

A következő lépés volt, a megkapott szögeket átalakítani olyan formába, hogy azt a robotnak el lehessen küldeni. Mind a robot ízületeinek a szögei, mind az emberi kar ízületeinek a szögei lineárisan változnak, így egy konverziós képletet kellett meghatározzak mind a négy megvezérelhető szabadságfokra. Mind az emberi mozgás, mind a robotkar mozgása valamilyen szinten limitált, ezért olyan pontokat kellett válasszak, amik a mozgás szélső értékeiket jelölik, amiket aztán lemértem mind a Kinectel mért értéket, mind a robot által visszaküldött értéket.

Vegyük példának az emberi karon a könyököt. Ennek a két szélső állapota vagy teljesen kinyújtva, vagy teljesen behúzva lehet. Ezt a Kinect segítségével lemértem és az így kapott értékeket lejegyeztem. Hasonló állapotba helyeztem a Baxter robotkarját is. Ebben az esetben a robotkar tovább is tud fordulni a teljesen egyenes állapotnál, ezért igyekeztem úgy beállítani a robotkart, hogy az minél jobban hasonlítson az emberi kar állapotához. Ebben a két állapotban lemértem a robot szögeit. Az így kapott értékekből pedig Matlab-ban egy egyenes egyenletet számoltam. Ez a 25. ábrán látható. A kiszámított képlet ábrázolva pedig a 26. ábrán.

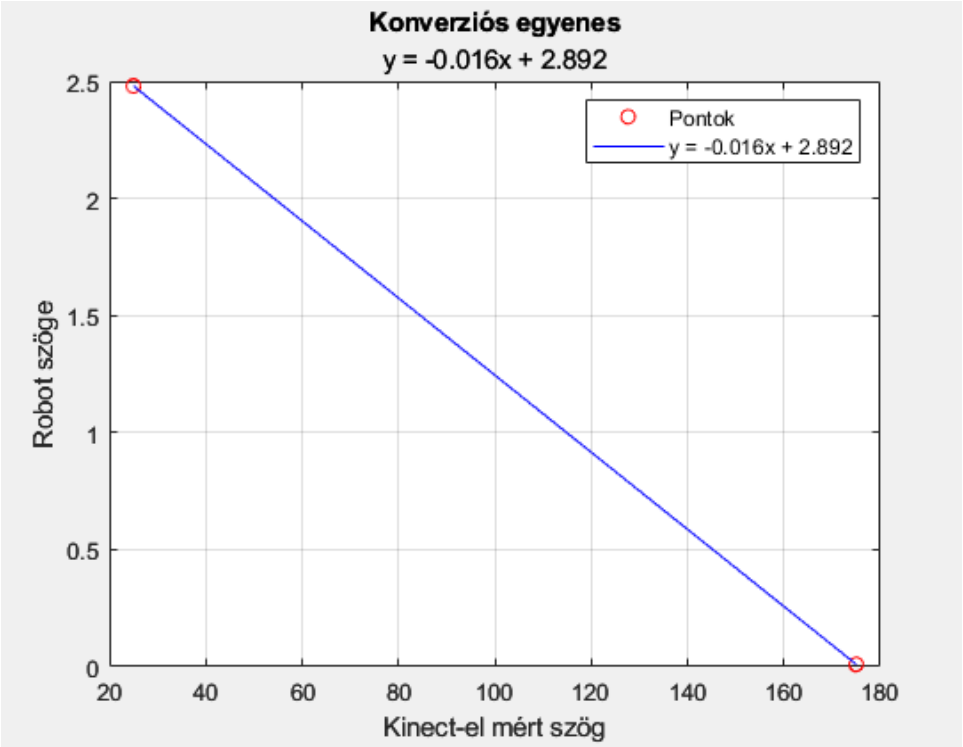
```
x = [25 175];
y = [2.48 0.01];

coefficients = polyfit(x, y, 1);
a = coefficients(1);
b = coefficients(2);

equation = sprintf('y = %.3fx + %.3f', a, b);
disp(equation);

plot(x, y, 'ro');
hold on;
plot(xLine, yLine, 'b-');
title('Két ponton átmenő egyenes');
xlabel('X tengely');
ylabel('Y tengely');
hold off;
```

Ábra 25: Az egyenes egyenletének kiszámítása



Ábra 26: Az átalakításra használt egyenlet ábrázolva

Az így kapott egyenletet tudtam aztán a szögek átalakítására használni. Miután a mért értékeket három tizedes pontosságúra alakítottam, mert így elegendő pontossággal tudtam a robot karjait vezérelni. Az így kapott szögeket a buildJsonAnglesLeft illetve buildJsonAnglesRight metódusokkal alakítottam olyan formába, amit tovább tudtam a gateway szervernek küldeni. Ebben a függvényben a mért szögeket kiszámoltam a fentebb említett képletekkel és felépítettem egy üzenetet ezekkel az adatokkal.

```
107     public string buildJsonAnglesLeft(double angleElbow, double angleElbowRotation, double angleShoulder, double angleShoulderRotation)
108     {
109         double angle1 = angleElbow * (-0.016) + 2.892;
110         double angle2 = angleShoulder * (-0.016) + 1.254;
111         double angle3 = angleShoulderRotation * (-0.018) + 0.85;
112         double angle4 = angleElbowRotation * (-0.014) - 1.42;
113         var cmd = new Dictionary<string, Dictionary<string, object>>
114         {
115             {
116                 "left_arm", new Dictionary<string, object>
117                 {
118                     { "set_joint_positions",  new Dictionary<string, object>
119                         {
120                             {"left_w0", 0.579},
121                             {"left_w1", 0.934},
122                             {"left_w2", 0.001},
123                             {"left_e0", angle4},
124                             {"left_e1", angle1},
125                             {"left_s0", angle3},
126                             {"left_s1", angle2}
127                         }
128                     }
129                 }
130             }
131         };
132         string toSend = JsonConvert.SerializeObject(cmd);
133         return toSend;
134     }
```

Ábra 27: A buildJsonAnglesLeft függvény

Ahogy az a 27. ábrán látható, először kiszámoltam a képletek alapján a robot csuklóinak szögeit, majd ezekből egy JSON formátumú üzenetet építettem fel, ami tartalmazza a vezérlési parancsot és a kiszámolt szögeket.

Emellett a rendszernek egyik funkcionalitása, hogy a Baxter ipari robot megfogó eszközeit is tudjuk kézmozdulatokkal kezelni. Ehhez első sorban fontos, hogy a felhasználó milyen követési módot választott, mivel, ha tükrözve kell a robotnak követni, akkor a jobb és bal kéz gesztusai felcserélődnek. Ilyenkor a felhasználó jobb keze, a robot bal kezét vezérli és fordítva. A megfogó eszközök vezérlésére a Kinect SDK egyik funkcióját használtam, ugyanis beépítetten támogatja a kézmozdulatok felismerését (lásd 5.2. fejezet). Így amikor újabb képkocka érkezik, miután a szögeket elküldtem a szerver felé, megnézem, hogy a felhasználó keze nyitott vagy zárt állapotban van, és aszerint küldök egy üzenetet a szervernek, hogy a robot végberendezéseit nyitott vagy zárt állapotba állítsa.

```
204         public string BiuldJsonGripperMessageLeft(bool state)
205         {
206             string toSend = "";
207             if (state == true)
208             {
209                 var cmd = new Dictionary<string, Dictionary<string, object>>
210                 {
211                     {
212                         "left_arm", new Dictionary<string, object>
213                         {
214                             { "gripper", 0 }
215                         }
216                     }
217                 };
218                 toSend = JsonConvert.SerializeObject(cmd);
219             }
220             else
221             {
222                 var cmd = new Dictionary<string, Dictionary<string, object>>
223                 {
224                     {
225                         "left_arm", new Dictionary<string, object>
226                         {
227                             { "gripper", 1 }
228                         }
229                     }
230                 };
231                 toSend = JsonConvert.SerializeObject(cmd);
232             }
233             return toSend;
234         }
```

Ábra 28: A megfogó eszközt kezelő függvény

```
74      if (body.HandLeftState == HandState.Closed)
75      {
76          if (_trackingMode == TrackingMode.Symmetric)
77          {
78              sendSocket.SendFrame(calculator.BiuldJsonGripperMessageLeft(true));
79              sendSocket.ReceiveFrameString();
80          }
81          else
82          {
83              sendSocket.SendFrame(calculator.BiuldJsonGripperMessageRight(true));
84              sendSocket.ReceiveFrameString();
85          }
86      }
87      else
88      {
89          if (_trackingMode == TrackingMode.Symmetric)
90          {
91              sendSocket.SendFrame(calculator.BiuldJsonGripperMessageLeft(false));
92              sendSocket.ReceiveFrameString();
93          }
94          else
95          {
96              sendSocket.SendFrame(calculator.BiuldJsonGripperMessageRight(false));
97              sendSocket.ReceiveFrameString();
98          }
99      }
```

Ábra 29: A felhasználó bal kezének állapot ellenőrzése

Ahogy az a 28. és 29. ábrán látható, a függvény meghívásakor true vagy false értéket kap, az alapján, hogy a felhasználó keze milyen állapotban van. Amikor a felhasználó keze zárt állapotban van, akkor a függvény true értéket kap, amikor nyitott állapotban van, ez az érték false.

A rendszer rendelkezik egy vészleállítási feltétellel. Abban az esetben, hogyha a felhasználó a karjait egy „X” alakba helyezi, a követés leáll és a robot visszaáll kezdeti állapotába.

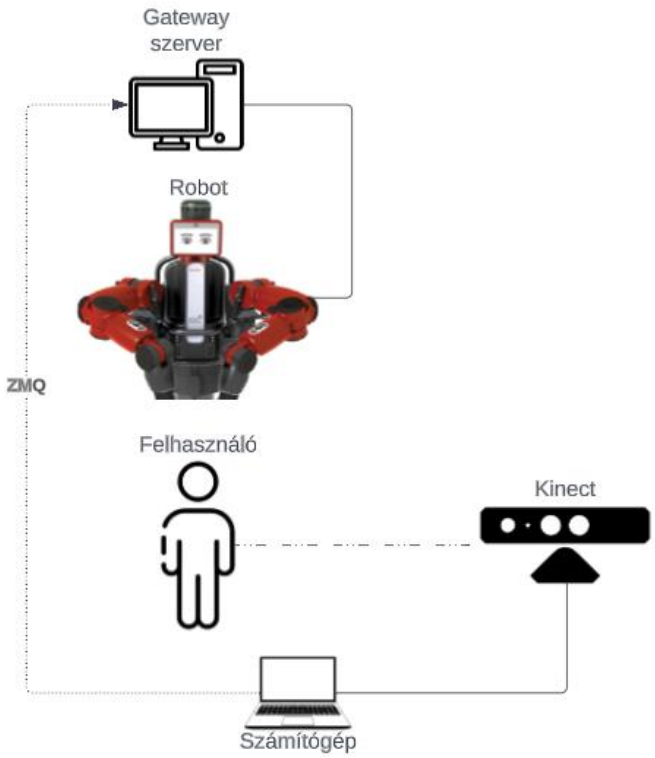
```
170     if (body.Joints[JointType.HandLeft].Position.X > body.Joints[JointType.HandRight].Position.X)
171     {
172         sendSocket.SendFrame(calculator.GoToStartLeft());
173         sendSocket.ReceiveFrameString();
174         sendSocket.SendFrame(calculator.GoToStartRight());
175         sendSocket.ReceiveFrameString();
176         return;
177     }
```

Ábra 30: A vészleállítási feltétel

Ez a feltétel a 30. ábrán látható. Abban az esetben, ha a felhasználó kezei egy „X” alakzatban állnak, a Kinect vízszintes tengelyén, az X koordinátája a bal kéznek nagyobb lesz, mint a jobbnak. Mivel ez egy nem szokványos kézmozdulat és jól elkülöníthető, ezt a mozdulatot választottam vészleállítási feltételnek.

6. Mérések

A rendszer teszteléséhez el kellett végeznem pár mérést, amivel a robot mozgásának helyességét akartam ellenőrizni. A mérés elvégzéséhez előkészítettem a rendszert, hogy a robot és a felhasználó (ebben az esetben én), egy síkban legyenek. A felhasználóval szemben áll a Kinect és a követő applikáció mellett, a felhasználó számítógépén futott az információk rögzítéséért felelő applikáció. Ez folyamatosan rögzítette a robot paramétereit, felhasználva a rendszer publikáló szálát. A rendszer elrendezése a 31. ábrán látható.



Ábra 31: A rendszer elrendezése a mérés alatt

A teszt során egy meghatározott mozgás sorozatot tettem meg, úgy, hogy közben a robot követte a mozgásomat. Ezalatt rögzítettem a robot egyik karjának paramétereit. A jobb karnak mértem a mozgását.

Amint elindítottam az általam tervezett a alkalmazást, elkezdtem rögzíteni a robot paramétereit is. Az általam elvégzett mozdulatok a következők voltak:

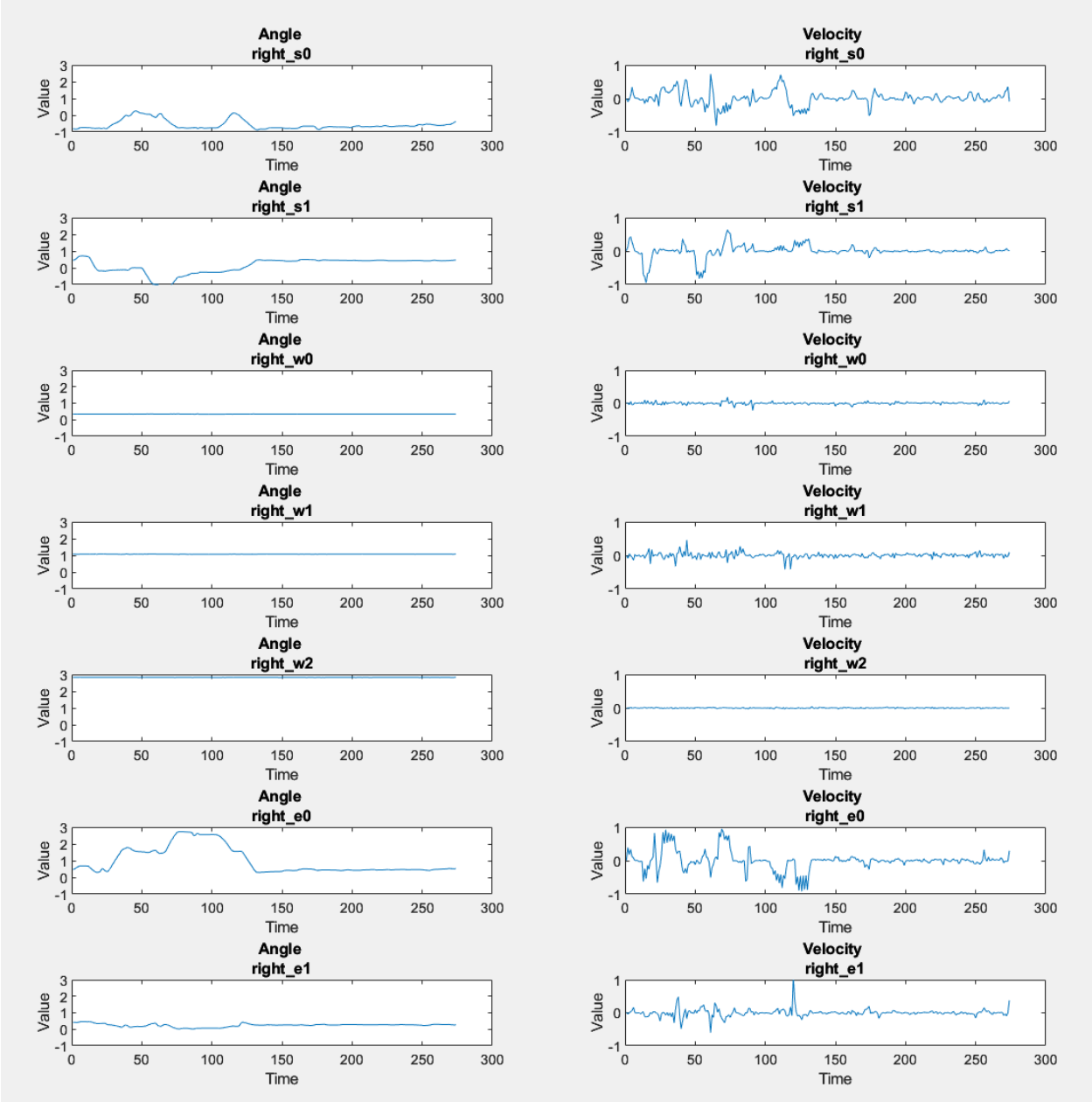
- Az első állapot a jobb kar oldalra néz, vízszintesen kinyújtva
- Ezután függőleges állapotba kerül, ugyanúgy kinyújtva
- A harmadik állapotban előre néz, szintén kinyújtva
- Majd vissza az első állapotba

A rögzített adatokat Matlab segítségével összegeztem diagramokba. Minden csuklónak lejegyeztem a szögeinek és a gyorsulásainak a változását, ezek a 32. ábrán láthatóak.

A mérésekből látszik, az általam használt megközelítés egyik problémája. Mivel a követés megoldását vektor módszerrel oldottam meg (lásd 5.3.4. fejezet, 23. ábra), ezért a robot mozgása

hasonlít a felhasználóéra (jelen esetben az enyémre), viszont a lehetséges hét megvezérelhető csuklóból csak négyre tudok elegendő információt kinyerni és így csak ezt a négy csuklót tudtam vezérelni (s0, s1, e0, e1).

Ez a 32. ábrán látottakból is kivehető. Amíg a négy megvezérelt csukló szögei a mozdulatok szerint váltakoznak, addig a right_w0, right_w1 illetve right_w2 szögeinek változásait egy egyenes írja le, a sebességek változásai pedig véletlenszerű értékek, mivel ezek a teljes kar mozgásából származnak, nem pedig a csuklók saját mozgásaiból.



Ábra 32: A mért adatok diagramokban

Ezekből a mérésekből látszik a rendszer legnagyobb hiányossága. A probléma direkt kinematikai feladatként való megközelítése magával vonta, hogy bizonyos csuklók megvezérlés nélkül maradtak.

Emellett a mérés során kiderült, hogy a rendszer egyéb funkciói az elvárások szerint működnek. A kommunikáció mindkét szálon megfelelően működik, párhuzamosan lehet a robot adatait olvasni és a robotot vezérelni.

7. Összefoglaló

7.1. Következtetések

A dolgozat egy rendszer tervezését és megvalósítását mutatja be, amely képes egy felhasználó mozdulataival egy ipari robot karjait vezérelni. A rendszer két fő komponensből áll, az egyik egy felhasználói interfész, a másik pedig egy robot interfész. Az első komponens felel a felhasználótól kinyert adatokért, a második pedig a robot vezérlését biztosítja.

A dolgozatban sikerült egy olyan rendszert megvalósítanom, amely egy Kinect v2 kamera segítségével képes, egy szerveren keresztül, a Baxter kétkarú ipari robotot megvezérelni a felhasználó mozdulatai alapján.

A rendszer képes felismerni az emberi alakot és annak mozgását, illetve az alapján parancsokat küldeni. A rendszer képes a Baxter robot megfogó eszközeit kézmozdulatok alapján vezérelni, illetve rendelkezik egy vészleállítási feltétellel, amivel a felhasználó távolról le tudja állítani a követést.

A rendszerhez tartozik egy egyszerű felhasználói felület, amely lehetővé teszi, hogy a felhasználó ki tudja választani, a rendszer megfelelő használatához szükséges paramétereket.

A kivitelezés alatt többféle probléma is felmerült, az egyik ilyen a Kinect szenzor kamera terében lévő több felhasználóból ered, mivel az alkalmazás csak abban az esetben tud pontosan a felhasználót követni, hogyha csak egy emberi alakot lát. Egy másik ilyen probléma volt, a Kinect kamerából eredő észlelési problémák. Vannak testtartások, ahol a kamera egy-egy ízületet nem, vagy rossz helyen érzékel, ilyenkor a robotkar nem várt módon viselkedhet.

7.2. Fejlesztési lehetőségek

A rendszer továbbfejlesztésére több lehetőség is van. Az egyik, egy plusz szenzor bevezetése, amellyel a tenyér pozícióját is jobban le lehet követni. Egy másik megoldás lenne az inverz kinematikai feladatot ötvözni a felhasználó karjainak helyzetével és így egy pontos és emberi mozgásokat biztosító követést kifejleszteni.

A másik továbbfejlesztési lehetőség, az arcfelismerés beiktatása, illetve a felhasználók regisztrálása és azok arc adatainak elmentése. Ezzel megoldódna a rendszernek az a problémája, hogy egyszerre csak egy felhasználó lehet a kamera terében, mert így a rendszer tudná, hogy az épp bejelentkezett felhasználót vegye csak figyelembe.

Egy másik lehetőség a rendszer fejlesztésére, egy webszerver beiktatása, vagy a meglévő szerver kibővítése, így nem lenne kötelező egy hálózaton legyen a felhasználó számítógépe és a szerver, illetve a robot. Ezzel a megoldással akárhonnan lehetne a rendszert használni.

8. Irodalomjegyzék

[1] Khajone, Saurabh A., S. W.s Mohod, and V. M. Harne. "Implementation of a wireless gesture controlled robotic arm." International Journal of innovative research in computer and communication engineering 3.1 (2015): 375-379.

[2] Aggarwal, Love & Gaur, Varnika & Verma, Puneet. (2013). Design and Implementation of a Wireless Gesture Controlled Robotic Arm with Vision. International Journal of Computer Applications. 79. 39-43. 10.5120/13805-1906.

[3] P. Atre, S. Bhagat, N. Pooniwala and P. Shah, "Efficient and Feasible Gesture Controlled Robotic Arm," 2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS), Madurai, India, 2018, pp. 1-6, doi: 10.1109/ICCONS.2018.8662943.

[4] K. K. Biswas and S. K. Basu, "Gesture recognition using Microsoft Kinect®," The 5th International Conference on Automation, Robotics and Applications, Wellington, New Zealand, 2011, pp. 100-103, doi: 10.1109/ICARA.2011.6144864.

[5] Yi Li, "Hand gesture recognition using Kinect," 2012 IEEE International Conference on Computer Science and Automation Engineering, Beijing, China, 2012, pp. 196-199, doi: 10.1109/ICSESS.2012.6269439.

[6] Ben Abdallah, Ismail, Yassine Bouteraa, and Chokri Rekik. "Kinect-based sliding mode control for Lynxmotion robotic arm." Advances in Human-Computer Interaction 2016 (2016).

[7] Reddivari, Hitesh, et al. "Teleoperation control of Baxter robot using body motion tracking." 2014 International conference on multisensor fusion and information integration for intelligent systems (MFI). IEEE, 2014.

[8] https://sdk.rethinkrobotics.com/wiki/Hardware_Specifications

[9] https://sdk.rethinkrobotics.com/wiki/Workspace_Guidelines

[10] Zhang, Zhengyou. "Microsoft kinect sensor and its effect." IEEE multimedia 19.2 (2012): 4-10.

[11] Quigley, Morgan, et al. "ROS: an open-source Robot Operating System." ICRA workshop on open source software. Vol. 3. No. 3.2. 2009.

[12] https://github.com/LightBuzz/Vitruvius

[13] https://zeromq.org/

[14] https://github.com/Vangos/kinect-2-coordinate-mapping

[15] Macenski, Steven, et al. "Robot Operating System 2: Design, architecture, and uses in the wild." Science Robotics 7.66 (2022): eabm6074.

[16] ROS Concepts: http://wiki.ros.org/ROS/Concepts

[17] Han, Jungong, et al. "Enhanced computer vision with microsoft kinect sensor: A review." IEEE transactions on cybernetics 43.5 (2013): 1318-1334.

[18] Fankhauser, Péter, et al. "Kinect v2 for mobile robot navigation: Evaluation and modeling." 2015 international conference on advanced robotics (ICAR). IEEE, 2015.

[19] Corti, Andrea, et al. "A metrological characterization of the Kinect V2 time-of-flight camera." Robotics and Autonomous Systems 75 (2016): 584-594.

[20] Basheer, Mawj Mohammed, and Asaf Varol. "An overview of robot operating system forensics." 2019 1st International Informatics and Software Engineering Conference (UBMYK). IEEE, 2019.

[21] ZeroMQ. (n.d.). ZeroMQ - The Guide: Transports. Retrieved from http://zguide.zeromq.org/docs/chapter2/

UNIVERSITATEA SAPIENTIA DIN CLUJ-NAPOCA
FACULTATEA DE ȘTIINȚE TEHNICE ȘI UMANISTE, TÎRGU-MUREȘ
SPECIALIZAREA CALCULATOARE

Vizat decan

Ș.l. dr. ing Kelemen András

Vizat director departament

Conf. dr. ing. Domokos József