

---

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

**Comanda robotului Baxter folosind  
metode de recunoaștere a  
imaginilor**

**PROIECT DE DIPLOMĂ**

**Coordonator științific:**

**Dr. ing. Szántó Zoltán  
Dr. ing. Márton Lőrinc**

**Absolvent:**

**Szász Attila**

**2023**

UNIVERSITATEA "SAPIENTIA" din CLUJ-NAPOCA  
Facultatea de Științe Tehnice și Umaniste din Târgu Mureș  
Specializarea: Calculatoare

Viza facultății:



## LUCRARE DE DIPLOMĂ

Coordonator științific:  
**Dr. Ing. Márton Lőrinc**  
**Dr. Ing. Szántó Zoltán**

Candidat: **Szász Attila**  
Anul absolvirii: **2023**

**a) Tema lucrării de licență:**

**Comanda robotului Baxter folosind metode de recunoaștere a imaginilor**

**b) Problemele principale tratate:**

- Programarea și comanda robotului Baxter în mediul ROS
- Metode de recunoaștere a imaginilor folosind coduri Aruco

**c) Desene obligatorii:**

- Diagramele UML a aplicației
- Rezultatele algoritmului de prelucrare a imaginilor
- Măsurători experimentale – Mișcarea robotului.

**d) Softuri obligatorii:**

- Program Python pentru comanda robotului Baxter
- Program Python pentru localizarea obiectelor în plan folosind coduri Aruco

**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] F. J. Romero-Ramirez, R. Muñoz-Salinas, R. Medina-Carnicer, Speeded up detection of squared fiducial markers, Image Vis. Comput., Vol. 76, pp. 38–47, 2018.
- [3] D. Jurado-Rodriguez, R. Munoz-Salinas, S. Garrido-Jurado, R. Medina-Carnicer, Design, Detection, and Tracking of Customized Fiducial Markers, IEEE Access, Vol. 9, pp. 140066–140078, 2021.

**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 232.

**Primit tema la data de:** 31.03.2022.

**Termen de predare:** 28. 06. 2023.

**Semnătura Director Departament**



**Semnătura responsabilului  
programului de studiu**



**Semnătura coordonatorului**



**Semnătura candidatului**



## **Declarație**

Subsemnata/ul Szász Attila, absolvent(ă) al specializării Calculatoare promoția 2019-2023 cunoscând prevederile Legii Educației Naționale 1/2011 și a Codului de etică și deontologie profesională a Universității Sapientia cu privire la furt intelectual declar pe propria răspundere că prezenta lucrare de licență de diplomă se bazează pe activitatea personală, cercetarea/proiectarea este efectuată de mine, informațiile și datele preluate din literatura de specialitate sunt citate în mod corespunzător.

Localitatea, Târgu Mureș

Data: 28.06.2023

Absolvent

Semnătura.....



---

# Extras

Aplicarea robotilor industriali este tot mai răspândită în diferite procese de fabricație și logistică, unde precizia, viteza și siguranța sunt factori importanți. Roboții sunt capabili să automatizeze sarcini plictisitoare, periculoase sau prea complexe pentru oameni. Cu toate acestea, colaborarea între roboți și oameni rămâne o provocare, mai ales atunci când roboții trebuie să se adapteze flexibil la mediile și sarcinile aflate în continuă schimbare.

Recunoașterea și manipularea obiectelor de mici dimensiuni sunt sarcini larg utilizate în context industrial și educațional. Cu toate acestea, determinarea tipului și poziționării cuburilor nu este întotdeauna simplă, în special atunci când cuburile au culori sau dimensiuni similare. În această lucrare, prezint o metodă care permite robotului Baxter să diferențieze cu ușurință cuburile folosind marcatori ArUco. Este prezentat modul de comandă a robotului bazat pe informația provenită de la modulul de recunoaștere a imaginilor. Pe parcursul studiului meu, ofer o prezentare detaliată a designului, implementării și testării sistemului.

În lucrarea mea, am dezvoltat un sistem care poate controla robotul Baxter astfel încât să poată recunoaște și manipula obiecte în formă de paralelipiped. Sistemul este compus din trei componente principale: un modul de prelucrare a imaginii, care detectează și identifică obiectele folosind bibliotecile OpenCV și cv2.aruco, un modul de comunicare, care utilizează protocolul ZMQ (ZeroMQ) pentru a trimite și primi date, și un modul de control al robotului, care folosește bibliotecile ROS (Robot Operating System) pentru a controla brațele și cleștii robotului. În funcționarea sistemului, robotul face o fotografie a zonei de lucru, determină poziția și tipul obiectelor, apoi le sortează în funcție de aceste informații.

**Cuvinte cheie:** Baxter, ROS, ArUco, Opencv, Python, procesare de imagini, robotică industrială și educațională

---

**SAPIENTIA ERDÉLYI MAGYAR  
TUDOMÁNYEGYETEM  
MAROSVÁSÁRHELYI KAR  
SZÁMÍTÁSTECHNIKA SZAK**

**Baxter robot képfelismerés alapú  
vezérlése**

**DIPLOMADOLGOZAT**

**Coordonator științific:**

**Dr. ing. Szántó Zoltán  
Dr. ing. Márton Lőrinc**

**Absolvent:**

**Szász Attila**

**2023**

---

# Kivonat

Az ipari robotok egyre inkább elterjednek a különböző gyártási és logisztikai folyamatokban, ahol a pontosság, a sebesség és a biztonság fontos szempontok. A robotok képesek automatizálni olyan feladatokat, amelyek unalmasak, veszélyesek vagy túl bonyolultak az emberek számára. Azonban a robotok és az emberek közötti együttműködés még mindig kihívást jelent, különösen akkor, ha a robotnak rugalmasan kell alkalmazkodniuk a változó környezethez és feladatokhoz.

Téglatest alakú tárgyak felismerése és mozgatása olyan feladat, amelyet számos ipari és oktatási célra használnak. A tárgyak típusának és elhelyezkedésének meghatározása azonban nem mindig egyszerű, különösen akkor, ha azok színükben vagy méretükben hasonlítanak egymásra. Ebben a dolgozatban egy olyan módszert mutatok be, amely a Baxter robot az ArUco kódok segítségével könnyedén megkülönbözteti a kockákat. Leírom, hogyan programoztam a Baxter robotot, hogy képes legyen meghatározni egy munkafelületen lévő tárgyak típusát és helyzetét, valamint rendezni azokat. A tanulmányom során részletesen bemutatom a rendszer tervezését, megvalósítását és tesztelését.

A dolgozatomban egy olyan rendszert fejlesztettem ki, amely képes a Baxter robotot vezérelni úgy, hogy képes legyen téglatest alakú tárgyakat felismerni és mozgatni. A rendszer három fő komponensből áll: egy képfeldolgozó modulból, amely az opencv valamint a cv2.aruco könyvtárak segítségével érzékelik és azonosítják a tárgyakat, egy kommunikációs modulból, amely ZMQ (ZeroMQ) protokollt használva küldi és fogadja az adatokat, és egy robotvezérlő modulból, amely ROS (Robot Operating System) könyvtárakat használva irányítja a robot karjait és megfogóit. A rendszer működése során a robot fényképet csinál a munkaterületről, meghatározza a tárgyak helyzetét és típusát, majd ezek alapján elvégzi a szétválogatást.

**Kulcsszavak:** Baxter, ROS, ArUco, Opencv, Python, képfeldolgozás, ipari és oktatási robotika

---

# Abstract

Industrial robots are more and more wide-spread in various manufacturing and logistics processes, where accuracy, speed, and safety are important factors. Robots are capable of automating tasks that are boring, dangerous, or too complex for humans. However, the collaboration between robots and humans represents challenges, especially when robots need to adapt flexibly to changing environments and tasks.

The recognition and manipulation of small objects are tasks widely used in industrial and educational contexts. However, determining the type and positioning of these objects is not always straightforward, particularly when they have similar colors or sizes. In this paper, I present a method that enables the Baxter robot to easily differentiate cubes using ArUco markers. I describe how I programmed the Baxter robot to determine the type and position of cubes on a workspace and arrange them accordingly. Throughout my study, I provide a detailed presentation of the system's design, implementation, and testing.

In my paper, I have developed a system that can control the Baxter robot to recognize and manipulate cuboid-shaped objects. The system consists of three main components: an image processing module that detects and identifies objects using the OpenCV and cv2.aruco libraries, a communication module that uses the ZMQ protocol to send and receive data, and a robot control module that utilizes the ROS (Robot Operating System) libraries to control the robot's arms and grippers. During the operation of the system, the robot captures an image of the workspace, determines the position and type of objects, and performs sorting based on this information.

**Keywords:** Baxter, ROS, ArUco, OpenCV, Python, image processing, industrial and educational robotics

---

# Tartalomjegyzék

1.	Bevezető .....	10
2.	Célkitűzések .....	12
3.	Irodalomkutatás.....	13
4.	Követelmény specifikáció .....	16
4.1.	Felhasználói követelmények .....	16
4.2.	Redszer követelmények .....	16
4.2.1.	Funkcionális követelmények.....	16
4.2.2.	Nem funkcionális követelmények.....	17
5.	A rendszer leírása.....	18
5.1.	Alkalmazott eszközök .....	19
5.1.1.	Baxter robot .....	19
5.1.2.	Baxter robot alkalmazása kutatásra.....	21
5.1.3.	Világ koordináta, Munkaterület, Inverze kinematika .....	22
5.1.4.	Baxter kamerái .....	25
5.1.5.	Gripperek .....	25
5.1.6.	ZMQ.....	26
5.1.7.	OpenCV .....	28
5.1.8.	ROS .....	29
5.1.9.	ArUco.....	30
5.2.	A Client felőli rész.....	32
5.2.1.	A Client felőli UI leírása .....	32
5.2.2.	A háttérben futó rendszer leírása .....	34
5.3.	A szerver felőli rész .....	36
6.	Megvalósítás .....	40
6.1.	A Baxter robot szimulációs környezetben .....	40
6.2.	A Kamera kép átküldése és feldolgozása .....	42
6.3.	Koordináta rendszerek és kalibrálás .....	46
6.4.	A kísérlet során felhasznált kockák .....	50
7.	Mérések .....	52
8.	Összefoglaló .....	56
8.1.	Következtetések .....	56
8.2.	Fejlesztési lehetőségek .....	57
9.	Irodalomjegyzék.....	58



---

# Ábrák, táblázatok jegyzéke

Ábra 1 A rendszer use-case diagramja .....	16
Ábra 2 A rendszer architektúrája .....	18
Ábra 3 Baxter Seas [8] .....	20
Ábra 4 Baxter csukló [7] .....	20
Ábra 5 Baxter arckifejezései [9].....	21
Ábra 6 Baxter robot koordináta rendszere [9] .....	22
Ábra 7 Jobb és bal kar felül nézetből 0 szögben [9] .....	23
Ábra 8 Matab által generált munkaterület [10] .....	24
Ábra 9 Baxter kamera és infravörös szenzor [11] .....	25
Ábra 10 A baxter robot elektromos és szívó markolója [12][11] .....	26
Ábra 11 ArUco kód példa .....	31
Ábra 12 UI+BackgroundWorker diagram .....	32
Ábra 13 A grafikus felület .....	33
Ábra 14 ArmController osztály diagramja .....	34
Ábra 15 A felhasználói rendszer aktivitás diagram .....	36
Ábra 16 A BaxterServer osztály diagramja .....	37
Ábra 17 A szerver működéséről szóló szekvencia diagram .....	38
Ábra 18 A szerver felé küldött üzenet felépítése .....	39
Ábra 19 A robot engedélyezése szimulációs környezetben .....	40
Ábra 20 A Baxter robot szimulációs környezetben .....	41
Ábra 21 A szerver oldalon történő témákra való feliratkozás .....	42
Ábra 22 A socketek létrehozása .....	43
Ábra 23 Kép kérés küldése .....	43
Ábra 24 A szerver oldalán történő kép lekérése és küldése .....	44
Ábra 25 ArUco kódok detektálása .....	44
Ábra 26 A kockák detektálása .....	45
Ábra 27 Élkeresés utáni kép .....	45
Ábra 28 Robot és a kamera koordináta rendszere .....	46
Ábra 29 Kalibráláshoz használt asztal .....	47
Ábra 30 Kód részlet az adatok fileba való mentéséről .....	48
Ábra 31 Átalakító egyenlet ábrázolva .....	49
Ábra 32 A végberendezés meghatározott koordinátához való küldése. ....	50
Ábra 33 A 3D-s kocka tervrajza .....	51
Ábra 34 A 3D kocka tervezése Autodeskb-en .....	51
Ábra 35 Az adat mentésből kinyert adatok JSON formátumban .....	52
Ábra 36 A bal karról mért adatok ábrázolva .....	53
Ábra 37 A jobb karról mért adatok ábrázolva .....	54
Ábra 38 A végberendezés pozíciójának ábrázolása a munkafolyamat során .....	55

---

## 1. Bevezető

Az elmúlt évtizedben az automatizációban óriási előrelépések történtek. A robotika ennek az automatizációnak nagy részét tölti ki. A robotok egyre fontosabb szerepet töltenek be a mindennapjainkban. A felgyorsult világban, az automatizált környezetünkben egyre nagyobb teret hódítanak a robotok az iparban a gyártási folyamatokban. A lenyűgöző fejlődésüknek köszönhetően bekerültek már az egészségügybe, mezőgazdaságba és a háztartásunkba is. A robotok előnye, hogy széles körben használhatóak, mivel képesek a fárasztó, monoton munkákat nagy precizitással elvégezni. Használhatóak mindenféle munkaterületen, ami az embereknek meghaladja a precizitását, munkabírását, vagy akár csak túl veszélyes számukra. Egyre több cég használja ezeket a gépeket, a termelés és a haszon növelése érdekében. A dolgozat ismerteti a robotokat, majd bemutat egy munkafolyamatot, amit a Baxter robot segítségével valósítunk meg. A Baxter robot egy kooperatív robot, amit a képfelismerés által nyújtott előnyökkel ötvözve fogunk használni.

A robotok története egészen az ókorig nyúlik vissza, de az igazi áttörés az ipari forradalom megjelenésével történt meg. Az emberek különféle módszereket találtak ki a gépek irányítására az elektromosság felhasználásával. Ezek a módszerek lehetővé tették a gépek motorokkal történő működtetését. A kezdetleges időkben ezek a robotok rögzített gépek voltak, amelyek képesek voltak a gyártási folyamatok elvégzésére. A robotok alkalmazása forradalmasította a gyártást. Ezek a fejlődések lehetővé tették a hatékonyabb és kevésbé emberfüggő termeléseket. Az 1980-as év eljövetelével elterjedtek a robotok digitálisan programozhatósága és a mesterséges intelligenciával való használatuk, ezzel megnagyobbítva funkcionalitását az emberiségnek.

A robotok egy mesterségesen létrehozott eszközök, amelyek képesek autonóm módon vagy vezérléssel működni. A robotok előnye, hogy széles körben használhatóak, mivel képesek a fárasztó, monoton munkákat nagy precizitással elvégezni. Használhatóak mindenféle munkaterületen, ami az embereknek meghaladja a precizitását, munkabírását, vagy akár csak túl veszélyes számukra. A kooperatív robotok az automatizálás és az emberi interakció ötvözéséből születtek. A kooperatív robotok használata ellentétben állnak a hagyományos ipari robot alkalmazásokkal, amelyek jól el vannak szigetelve az emberi környezettől. Ezek a robotok képesek önállóan dolgozni, de képesek együttműködni emberekkel is. Ezek a robotok úgy vannak tervezve, hogy közvetlenül az emberrel együtt tudjanak dolgozni. Fizikai kölcsönhatásba tudnak lépni az emberekkel és kiegészítik egymás munkáját ugyanabban a munkaterületen. Ezek a kooperatív robotok rendelkeznek olyan érzékelőkkel és biztonsági funkciókkal, amelyek segítségével észlelik az

---

emberi jelenlétet a munkaterületükön belül. Azon felül a könnyen programozhatóságuk miatt könnyedén alkalmazkodnak a változó munkaterületekhez és munkafolyamatokhoz.

A képfelismerés, vagy más néven a gépi látás olyan technológia, amely lehetővé teszi a gépeknek a vizuális környezet értelmezését, megértését. Ennek a technológiának köszönhetően lehetőség nyílt a számítógépeken a vizuális információk feldolgozására. Ez a technológia jelentős fejlődésen ment keresztül az elmúlt években. A képfelismerést már az 1960-as években elkezdték tanulmányozni, amikor ugyanis a kutatók az emberi látást próbálták lemodellezni a számítógépes környezetekben. Próbálták megérteni és algoritmusokat fejleszteni a képek elemzésére. A számítástechnika rohamos fejlődésében a növekvő számítási kapacitásnak köszönhetően a képfelismerés forradalmi áttöréseken ment keresztül. Ezekben az időkben az algoritmusok jelentősen javultak, amiknek köszönhetően manapság már mindenhol megtalálható a képfelismerés által nyújtott előnyök. Rengeteg nyílt forráskódú könyvtárak jelentek meg a felhasználók számára, amelyek számos eszközt nyújtanak ezen technológia használatára. A robotika területén a képfelismerés technológiának a használata óriási hatást gyakorolt a működésükre és a felhasználásukra. A képfelismerésnek köszönhetően a robotok képesek voltak a helyzetük meghatározására, objektumok felismerésére és nem utolsósorban a környezetükkel való alkalmazkodásra.

---

## 2. Célkitűzések

Céлом egy olyan rendszer létrehozása, amely képes automatikusan elvégezni a kockák rendezéséből álló feladatot egy kooperatív robot segítségével. Ehhez a rendszernek alkalmaznia kell a képfelismerési algoritmusokat, és pontosan vezérelnie kell a robotkarok mozgását a kockák lokalizálása és a mozgatus érdekében. Szándékom egy olyan rendszer kialakítása, amely csökkenti az emberi beavatkozás szükségét a munkafolyamatban.

A rendszernek képesnek kell lennie felismerni a munkaterületen található kockákat, meghatározni azok helyzetüket, síkhoz viszonyított elfordulási szögüket, valamint a típusukat, hogy megfelelőek-e vagy selejtesek. Ez az információ kiemelten fontos a rendszer megfelelő működése szempontjából.

A másik cél a robotkarok irányítása. Fontos, hogy pontosan lokalizáljuk és mozgassuk a karokat a precíz munkafolyamat érdekében. A robotkarok feladata, hogy optimális sorrendben válasszák el a kockákat a selejtesektől, valamint különítsék el a nem optimális darabokat egy selejtgyűjtő helyre. Az optimális sorrendet a kockák sorszáma alapján határozzuk meg.

A rendszer és a robot közötti kommunikáció kiemelten fontos a munkafolyamat lebonyolítása szempontjából. Létre kellett hoznunk egy olyan kommunikációs csatornát a robot és a rendszer között, amely lehetővé teszi a parancsok küldését a robotnak, valamint az adatok fogadását tőle. Az adatok fogadása különösen fontos a képfogadás folyamatában.

A rendszer megvalósításánál fontos volt a skálázhatósága. Ez azt jelenti, hogy a munkaterület méretének bővítése, valamint a munkaterületen lévő kockák számának változtatása könnyedén kivitelezhető.

---

### 3. Irodalomkutatás

A tárgyak felismerése és szétválasztása nem újdonság a robotikában. Sok hasonló projekt létezik, amelyek hasonló célokat tűztek ki, mint én. Az alábbi cikkek segítségével betekintést nyerhetünk több témába, például hogyan válogatnak ki labdákat, vagy hogyan működnek az ArUco kódok vagy a képfelismerési algoritmusok a robotikában.

Az [1] vázol egy alkalmazást, amely képes felismerni és rendezni különböző színű labdákat. A rendszerhez egy Raspberry Pi kamerát használtak, valamint egy Arduino által vezérelt robotkart. A labdák felismeréséhez OpenCV-t alkalmaztak.

A cikk részletesen ismerteti a robotrendszer felépítését és a robotkar mozgásának vezérlését. Az eredmények szerint a robot hatékonyan és pontosan elvégezte a szín alapú rendezési feladatot. A cikk részletesen bemutatja, hogyan használták ki az OpenCV által adott lehetőségeket a színfelismeréshez és a tárgydetektáláshoz. Leírja, hogyan lehet elkülöníteni a színeket egymástól és hogyan lehet alkalmazni a Hough-transzformációt a tárgyak meghatározásához. Részletesen kitér arra is, hogyan lehet összekötni a Raspberry Pi-t az Arduinóval egy robotikai rendszerben. A Raspberry Pi volt az az eszköz, amely elvégezte a különböző képfelismerő algoritmusokat, és kommunikált az Arduinóval, amely vezérli a robotkart. Bemutatja, hogyan oldották meg ezt a kommunikációt, és hogyan lehet az Arduinót használni a robotkart vezérelni és a szenzorok értékeit kiolvasni. Továbbá bemutatja, hogyan lehet a robotkart a megfelelő pozícióba és orientációba helyezni ahhoz, hogy megfelelően kezelje a tárgyak helyzetét és irányítsa a markolókat.

A [2] számú cikkben egy alacsony költségű tárgyak rendezésére képes robotkar van bemutatva, amelyben egy Raspberry Pi-t és egy Arduinót használtak. A cikk részletesen ismerteti, hogyan lehet a Raspberry Pi segítségével irányítani a robotkarokat és rendezni a tárgyakat. A robotot egy Arduinó alapú motorvezérlővel és szervomotorral valósították meg.

A cikk bemutatja a rendszer felépítését és működését. Leírja, hogyan lehet a Raspberry Pi-val vezérelni a robotkart, valamint hogyan lehet a vezérlő parancsokat megfelelően feldolgozni. Emellett betekintést kapunk a szoftveres megvalósításba is, amely tárgyalja a szükséges algoritmusokat és eszközöket.

A [3] számú cikkben egy robotikai alkalmazást láthatunk, amely képes önállóan navigálni beltéri környezetben ArUco kódok segítségével. Ez a robot hibajavító képességgel rendelkezik, amely a

---

Reed-Solomon kódok használatában nyilvánul meg. A Reed-Solomon kódok alapvetően a hibajavításra szolgálnak, és lehetővé teszik a hibásan érkező adatok helyesítését. Az eljárás során az eredeti adatokat hozzárendelik egy hibajavító kódsorozathoz, amely tartalmazza az adatokból származó redundáns információkat.

A cikk bemutatja a robot rendszernek a felépítését, a vizuális marker generálását és detektálását. Kitér arra, hogy hogyan lehet használni a Reed-Solomon kódokat az ArUco kódok hibajavításánál. Ezeket a kódokat lehet használni a vizuális marker redundanciájának növelésére és a hibák detektálására, valamint azok korrigálására, amelyek a markerek kopása vagy részleges takarása miatt keletkezhetnek.

A cikk részletesen tárgyalja, hogy ezeket a markereket hogyan lehet használni az aktuális pozíció és orientáció meghatározására.

A [4] számú cikk egy robotikai alkalmazást mutat be, amely képes felismerni és detektálni a tárgyakat a robot számára. A robot egy rendezésből álló feladatsort kell végrehajtson. A robot egy webkamerát és Matlab vezérlőt használ.

A cikk bemutatja a robot felépítését, a képfeldolgozó algoritmusokat, valamint a tesztelési eredményeket. Az eredmények szerint a robot képes volt felismerni és detektálni a különböző alakú és színű tárgyakat. A cikk jól szemlélteti, hogyan lehet képfeldolgozó algoritmusokat használni a tárgyfelismeréshez. Betekintést nyerhetünk abba, hogy hogyan lehet elkülöníteni a tárgyakat egymástól, hogyan lehet kinyerni és osztályozni a jellemzőiket.

A cikk kitér arra is, hogy hogyan lehet a Matlab vezérlőt használni a robotkar vezérlésére. Ennek segítségével lehet a robotkart a megfelelő pozícióba és orientációba küldeni, valamint lehet nyitni és zárni a markolókat.

A [5] -os számú cikkben egy földalatti tetőtámaszt mutat be. A cikk bemutat egy robotikai alkalmazást, amely képes ezt a tetőtámaszt két robotkarként vezérelni. Matlab/Simulink környezetet használnak a robotkarok modellezéséhez és a vezérlő algoritmusok teszteléséhez.

A cikk bemutatja a robot rendszerének felépítését, valamint a robotkarok kinematikáját, a direkt és az inverz modelljét. Emellett mutat egy szimulációs környezetet, ahol tesztelve volt. A cikk kitér arra, hogy hogyan lehet a robotkarok kinematikai modelljeit használni a robotkarok mozgásának meghatározására. Bemutatja, hogy hogyan lehet kiszámítani a robotkarok végpontjainak pozícióját és orientációját a adott csuklásszögek alapján. Emellett bemutatja az inverz kinematikai módszert is, amely a szükséges csuklásszögeket határozza meg a végpontok pozíciója és orientációja alapján.

---

A [6]-os számú cikkben a Baxter robot teljesítményét mutatják be. A robot egy webkamerát és egy Matlab vezérlőt használ. A cikk részletesen bemutatja a Baxter robot rendszerének felépítését, valamint a kinematikai pontosságát. Emellett összehasonlításra kerül egy hagyományos ipari karral is. A kutatás eredményei szerint a robot pontossága meglehetősen korlátozott, de képes kezelni a háztartási méretű tárgyakat. A robot pontossága szemléltetéséhez egy négyzet rajzolásával mutatta be, ahol látszott, hogy a Baxter robot pontossága jelentősen eltér az ipari robot pontosságához viszonyítva ami egy Denso robot volt.

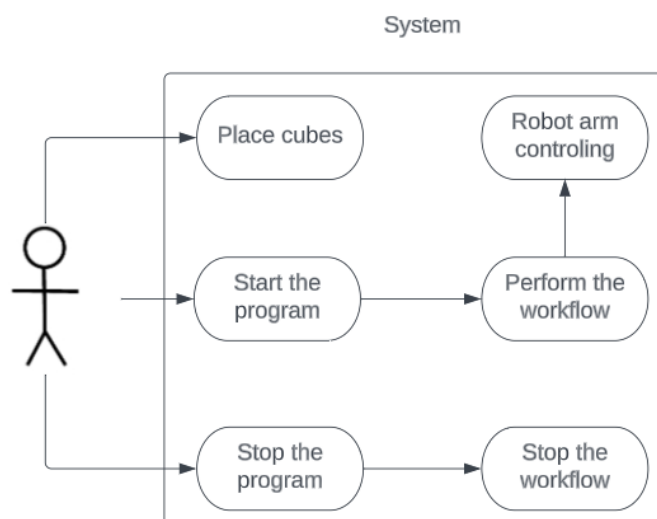
A cikkben olvashatunk arról, hogy hogyan lehet a Baxter kutató robotot használni biztonságos ember-robot együttműködéshez, valamint hogyan lehet a robotot programozni különböző feladatokhoz. Továbbá ismerteti, hogyan lehet használni a direkt és inverz kinematikai modelleket a végpontok pozíciójára és a csuklók pozíciójára meghatározására. Leírja továbbá, hogy hogyan lehet a Matlab vezérlőt használni a robotkarok irányítására.

---

## 4. Követelmény specifikáció

### 4.1. Felhasználói követelmények

A rendszer használata nagyon egyszerű és kényelmes a felhasználó számára. Ezt az egyszerűséget jól szemlélteti a 1. ábrán bemutatott use-case diagram is. A rendszer automatikusan végzi el a szükséges feladatokat, a robot önállóan dolgozik, így a felhasználónak csak el kell indítania és ellenőriznie kell a munkafolyamatot. A munka befejezése után a felhasználónak le kell állítania a rendszert. A leállítás után a robot visszatér a kiindulási pozíciójába és készen áll az újraindításra. Ha a felhasználó nem állítja le a rendszert, akkor az továbbra is keresi a kockákat, és ha talál egyet, akkor azonnal elhelyezi azokat a megfelelő helyre. A rendszer használatával a felhasználó hatékonyan és biztonságosan tudja rendezni a kockákat különböző terepeken.



Ábra 1 A rendszer use-case diagramja

### 4.2. Rendszer követelmények

#### 4.2.1. Funkcionális követelmények

A funkcionális követelmények azt határozzák meg, hogy milyen funkciókat vagy szolgáltatásokat kell nyújtania a rendszernek vagy a szoftvernek. Ezek a követelmények részletesen leírják, hogy milyen konkrét tevékenységeket kell végrehajtania a rendszernek, hogyan kell reagálnia a különböző bemenetekre, és hogyan kell alkalmazkodnia a változó körülményekhez. A funkcionális követelmények segítenek meghatározni a rendszer célját, hatókörét és teljesítményét. A dolgozatomban fellelhető funkcionális követelmények:



- 
- A rendszernek képesnek kell lennie felismerni és megkülönböztetni a különböző típusú kockákat a munkafelületen.
  - A rendszernek képesnek kell lennie precízen mozgatni a karjait a megfelelő helyre, ügyelve a többi ember biztonságára, megtartva a kooperatív robot jellegét.
  - A rendszernek képesnek kell lennie arra, hogy észlelje, ha egy ember a munkaterületen tartózkodik, és ebben az esetben le kell állítania a munkafolyamatot.
  - A rendszer automatikusan kell tudnia a munkáját végezni, anélkül, hogy emberi beavatkozás kelljen.
  - A rendszer folyamatosan naplózza az adatokat, amikor a munkafolyamat végrehajtódik

#### **4.2.2. Nem funkcionális követelmények**

A nem funkcionális követelmények olyan követelmények, amelyek nem közvetlenül a rendszer funkcióival vagy szolgáltatásaival kapcsolatosak, de befolyásolják annak működését. A nem funkcionális követelmények elengedhetetlenek a rendszer hatékony és biztonságos működéséhez.

A rendszer működtetéséhez szükséges, hogy a felhasználó rendelkezzen Python alkalmazások futtatására alkalmas környezettel. Minimum Python 3-as verzióra van szükség a rendszer futtatásához. A kommunikációhoz telepíteni kell a ZMQ (ZeroMQ) csomagot, valamint a képfeldolgozáshoz szükséges az OpenCV és a CV2.aruco csomagok használata.

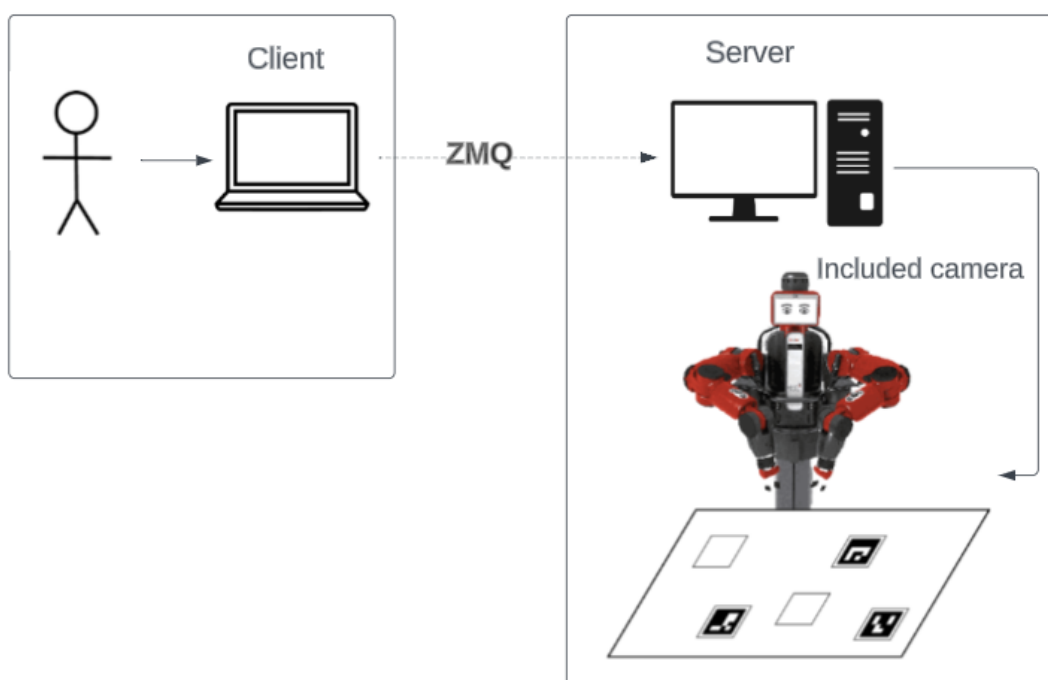
A rendszer fejlesztése Ubuntu 16, valamint Windows operációs rendszeren történt. A fejlesztői környezetként a Microsoft Visual Studio Code volt használva, valamint Python nyelven lett lefejlesztve. A rendszer működéséhez elengedhetetlen a közös Wi-Fi hálózathoz való csatlakozás. A képfeldolgozás zavartalan működése érdekében fontos, hogy a robot környezetében ne legyen túl erős fényforrás, mivel az visszaverődhet a tárgyakra. Emellett érdemes elkerülni az árnyékok képződését a munkaterületen. A robot tapadókorongjainak köszönhetően szükséges egy kompresszor a megfelelő tapadás biztosítása érdekében.

A felhasznált robotnak biztonságosnak kell lennie, ami azt jelenti, hogy nem okoz sérülést sem önmagában, sem a környezetében. A robot, ha ütközik valamivel, azonnal leállítja a munkafolyamatot. Ezen biztonsági funkciók mellett a robotnak rendelkeznie kell egy vészkapcsolóval is, amely bármikor megszakíthatja a robot mozgását.

## 5. A rendszer leírása

A megvalósított rendszer konkrétan egy robotvezérlő rendszer, amely két fő egységből tevődik össze. Az első egység a szerver felőli rész, amely a robottal való kommunikációért felelős, és a második egység a felhasználó felőli rész, amely a számítási erőforrásokat és a robot utasításait kezeli. A szerver felőli rész gondoskodik a kommunikációról a robot és a rendszer között, a felhasználó felőli rész pedig a felhasználó interakcióját biztosítja a rendszerrel, valamint kezeli a számításokat és az utasításokat, amelyeket a robot végrehajt.

A rendszerben ez a két egység szorosan együttműködik, hogy biztosítsa a megfelelő működést. A 2. ábrán látható az architektúra, ahol jól megfigyelhető a felhasználói és a szerver oldali egység kommunikációja, valamint az egységek közötti kapcsolatok.



Ábra 2 A rendszer architektúrája

---

## 5.1. Alkalmazott eszközök

Az alábbi eszközök és technológiák segítségével valósítottam meg a projektet. Baxter robot, amely egy kooperatív robot, amivel képesek vagyunk különféle feladatokat elvégezni. A roboton található kamera, amely segítségével meghatározzuk a tárgyak helyzetét. A robot karjain lévő megfogó eszközök segítségével a tárgyakat tudjuk mozgatni. A projekt során használt 3D nyomtatott kockák, amelyeket mozgatunk. A kommunikációra használt ZMQ könyvtár, a képek feldolgozására használt OpenCV könyvtár, és az ArUco kódok, amelyek segítenek a tárgyak azonosításában és helyzetében.

Az alábbi részekben részletes leírásokat kaphatunk a felhasznált eszközökről.

### 5.1.1. Baxter robot

A Baxter robotot a Rodney Brooks által alapított Rethink Robotics<sup>1</sup> építette. Két változatot is bemutatott abban az időben, egy iparba szánt robotot és egy oktatásra szánt robotot. Kinézetben és felépítésben nem volt különbség, de szoftver szempontjából igen. A robotot egyszerűbb gyári feladatokra tervezték, mint például tárgyak mozgatása, emelése vagy rendezése.

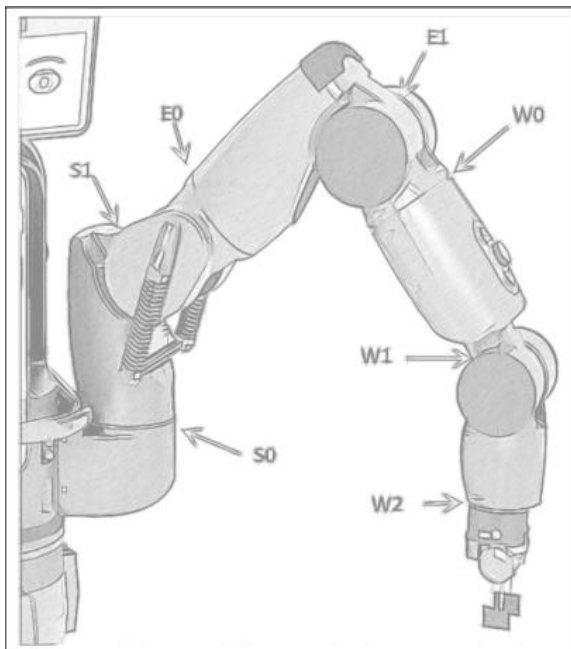
A Baxter robot egy rendkívül fejlett két karú robot, amit az együttműködő munkavégzésre találtak ki. Egyik legfőbb jellemzője, hogy mindkét karja 7 szabadságfokkal (DOF) rendelkezik, ami lehetővé teszi neki az óriási mozgásteret, és mindenféle pozícióba való eljutást. A 7 szabadságfokot jól szemelteti a 4. ábrán található kép. A 7 szabadságfokon kívül fontos még megemlíteni a csuklós működtetőelemeket (ízületi hajtóművek), amiknek köszönhetően egyedivé teszi a gyártórobotok között [7]. A Baxter robot ízületei sorozat-rugós-hajtóművekből (series-elastic actuators, SEAs), amelyekben rugó található a motor és fogaskerek között. A Baxter robot ízületei és azok felepetési jól láthatók az 3. ábrán.

Ennek a sorozat rugós hajtóműveknek számos előnyeik vannak a merev hajtóművekkel szemben, mint például az ütési terhelésekkel szembeni tolerancia, a növekvő csuklóteljesítmény és a mechanikai kimeneti impedancia<sup>2</sup>. Ezek az előnyök lehetővé teszik a robotok pontosabb és stabilabb működését, valamint csökkentik a környezetükben okozott károkat. Ezeknek az érzékelőknek és a SEA-knak köszönhetően a robot pontosan tudja a karjait lokalizálni és segít az érintkezés érzékelésében.

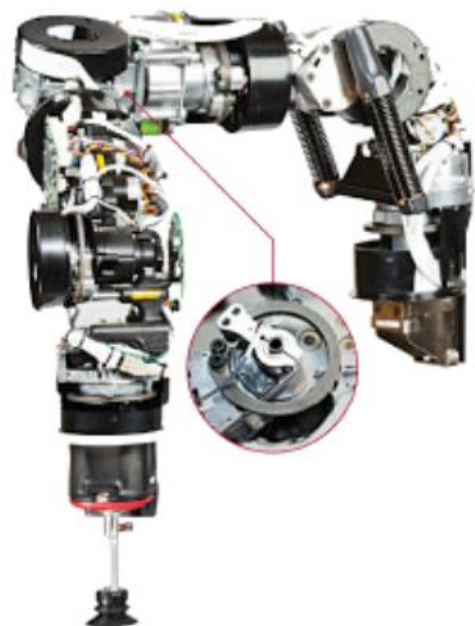
---

<sup>1</sup> <https://www.bbc.com/news/business-20800118>

<sup>2</sup> <https://www.intechopen.com/chapters/51224>



Ábra 4 Baxter csuklói [7]



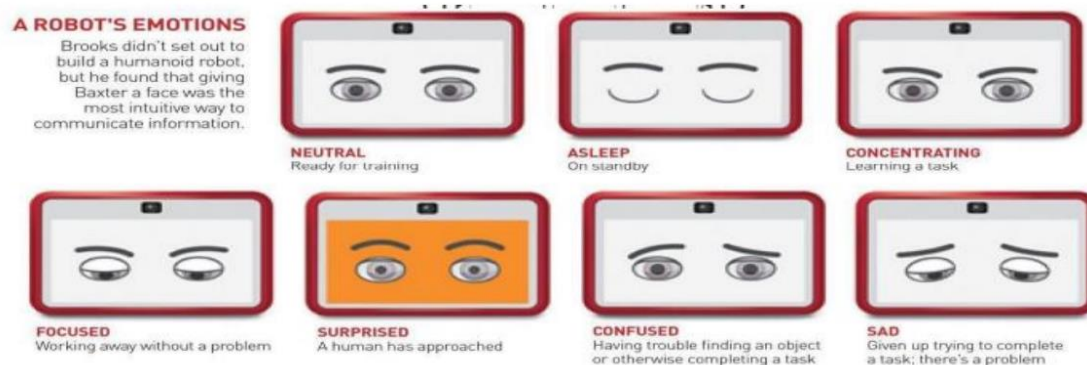
Ábra 3 Baxter Seas [8]

A kooperatív (Cobot) robotok olyan robotok, amiket úgy terveztek, hogy közvetlenül az emberrel együtt tudjanak dolgozni. A kooperatív robotok fizikai kölcsönhatásba tudnak lépni az emberekkel és kisegítik egymás munkáját ugyanabban a munkatérben. A Baxter robot számos érzékelővel van felszerelve, amelyek lehetővé teszik számára, hogy különböző feladatokat végezzen. Ezek a szenzorok segítik a robotot az emberekkel való együttműködésre. A következő szenzorok segítik a robotot a környezet érzékelésére:

A fejen található egy 360 fokban érzékelő szenzor, ami segít a környezet felismerésében és a térben történő helyezkedésben. A robot fején van még egy 1024x600-as<sup>3</sup> felbontású kijelző, ami segít a külvilággal való kommunikálásban, emellett még van 3 kamera rajta, infravörös érzékelők és a karok végén gyorsulásmérők. A 5. képen található a robot fejen levő ki jelző, és hogy az milyen kifejezésekkel segíti a külvilággal való kommunikációt. Ezek a szenzorok lehetővé teszik a Baxter robot biztonságos használatát a munkaterületeken.

A Baxter robot fején található még egy ledcsík, ami képes jelezni, hogy éppen milyen folyamatban van a robot.

<sup>3</sup> <https://sdk.rethinkrobotics.com/wiki/Head>



Ábra 5 Baxter arc kifejezései [9]

### 5.1.2. Baxter robot alkalmazása kutatásra

A Baxter robot második verzióját a Rethink<sup>4</sup> cég egy évvel az előző modell megvalósítása után mutatta be. Ez a verzió elsősorban az akadémiai és kutató szervezetek számára készült. A kutatási célokra szánt robot hardvere teljesen megegyezett az eredeti gyártási célokra szánt robot hardverevel, de azonban a szoftver verziók teljesen megkülönböztettek. Az SDK segítségével a kutatók és diákok képesek voltak egyedi szoftvert fejleszteni a Baxter robot számára, amellyel közvetlenül lehetett futtatni a robtoon található ROS parancsokat és scripteket. Ezeknek az SDK-knak köszönhetően a Baxter a ROS masterként tudott működni, ami lehetővé tette a robot irányítását.

Az SDK (Software Development Kit) egy olyan szoftver fejlesztői csomag, amely segíti a fejlesztőket, hogy egy szoftvert hozzanak létre egy adott platformhoz vagy rendszerhez. Ezek az SDK-k általában eszközöket és dokumentációkat tartalmaznak, jelen esetben olyan eszközöket, amik lehetővé tették a szoftverek fejlesztését a Baxter számára.

A Baxter robotot számos kutatóterületen használták<sup>5</sup>. Rengeteg egyetemi és cégés projekt valósult meg vele. Nagyszerű demonstráló és oktató robot volt, amivel könnyen lehetett tanulni. A cégnek a célja az volt, hogy olyan platformot hozzon létre, amin keresztül bárki könnyedén tudjon alkalmazásokat fejleszteni.

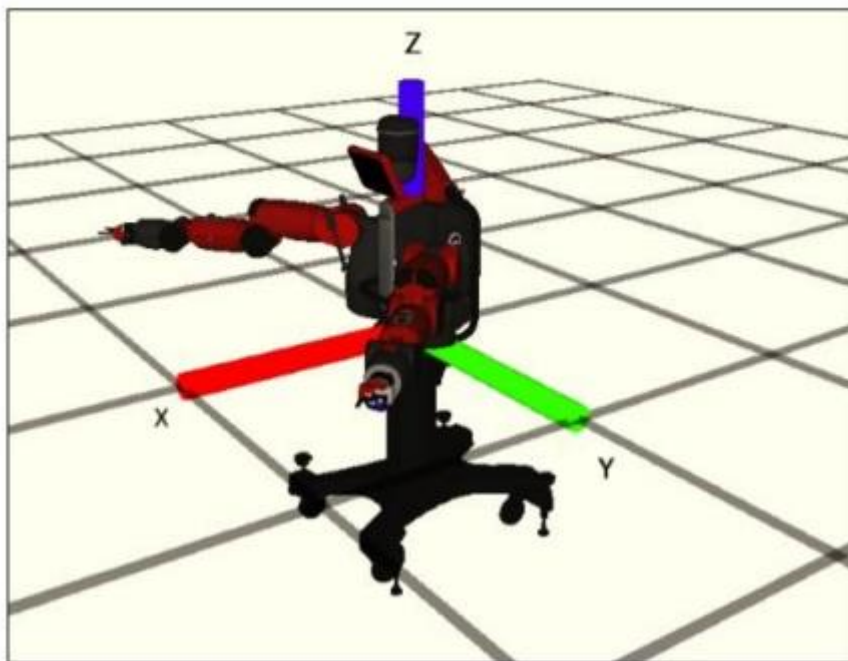
<sup>4</sup> <https://spectrum.ieee.org/rethink-robotics-baxter-robot-factory-worker>

<sup>5</sup> <https://spectrum.ieee.org/rethink-robotics-baxter-research-robot>

### 5.1.3. Világ koordináta, Munkaterület, Inverze kinematika

A világkoordináta<sup>6</sup> rendszer egy olyan rendszer, amely lehetővé teszi a robot pontos mozgásának a meghatározását. A robotok mozgását a világkoordináta rendszer segítségével lehet programozni. Ahhoz, hogy a végberendezés pozícióját és orientációját tudjuk meghatározni, szükségünk van egy alap koordináta rendszerre, amelyhez tudjuk a többi pozíciót mérni.

A kamera alapállása adja az x tengely, ami előre fut a Baxter középvonalán, az y tengely balra fut a középvonaltól, és a z tengely függőlegesen felfelé fut. A Baxter alap koordináta-rendszerének z tengelye a Baxter törzsének alapjánál található. Ez a  $z = 0$  pozíció. Az  $x = 0$ ,  $y = 0$  pozíció a Baxter előlapja mögött van, a középvonal mentén a függőleges tengelyen. A 6. ábrán szemléltetve van, hogy ez a koordináta-rendszer hogyan helyezkedik el a robotra vetítve.

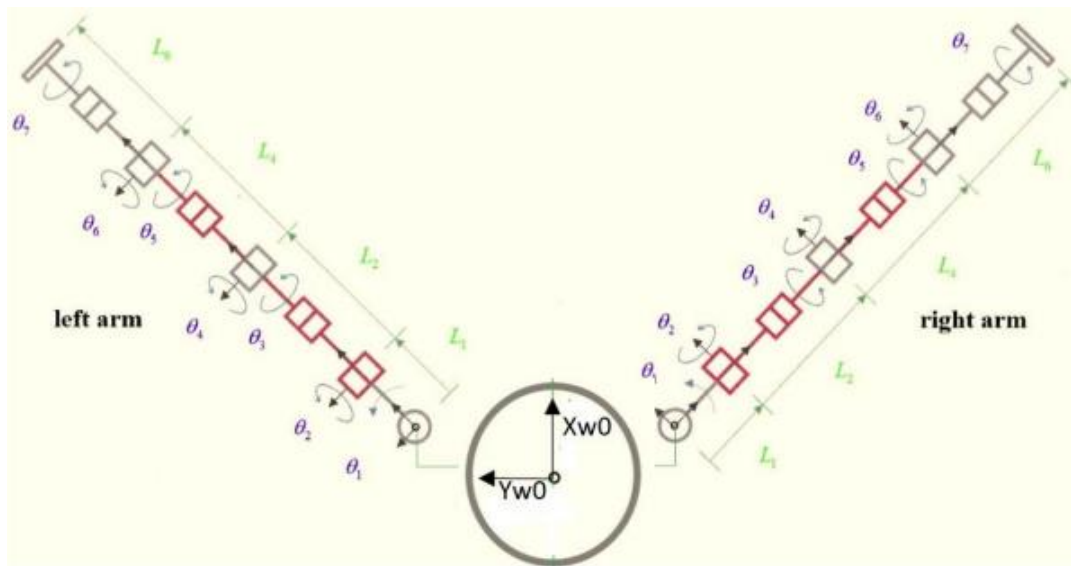


Ábra 6 Baxter robot koordináta rendszere [9]

A koordináta rendszer használata fontos, hogy meghatározzuk a Baxter robot karjain lévő fogók (végberendezések) pozícióját. Az Origó (0,0,0) mért x,y,z távolság adja meg a végberendezés pontos helyét. A koordináta rendszer nagyon hasznos, mivel az az origója a műveletek során nem mozog, ezért akármikor megtudjuk kapni a pontos helyet a robot végberendezéseinek az origóhoz viszonyítva. Akármilyen mozdulatokat végzünk, a csuklók helyzete és a végberendezés helyzete pontosan meghatározható az alapkoordinátáknak köszönhetően. Az 7. ábrán jól látható, hogy a

<sup>6</sup> <https://developercenter.robotstudio.com/api/robotstudio/articles/Concepts/Math/Coordinate-Systems.html>

robot karjai hogyan viszonyulnak az alap koordinátarendszerhez, valamint az is, hogy milyen területet fednek le, ha fentről nézzük őket.



Ábra 7 Jobb es bal kar felül nézetből 0 szögben [9]

A robotika területén a robot manipulátor munkaterületét<sup>7</sup> gyakran úgy határozzák meg, mint azon pontok halmazát, amelyek elérhetők a végberendezése által, vagyis a tér, amelyben a robot dolgozik. Ez lehet egy 3D tér vagy egy 2D felület.

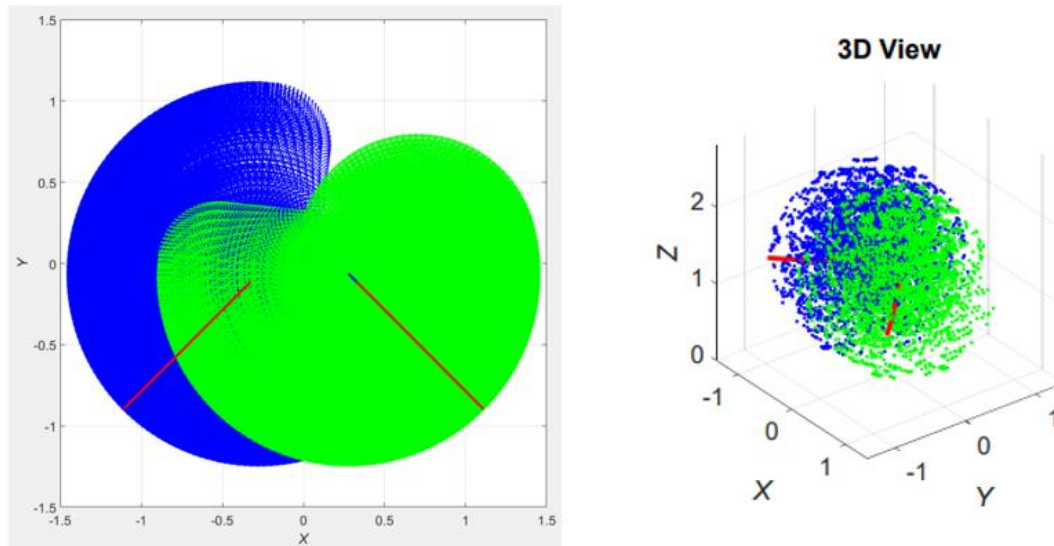
A robot munkaterülete a végberendezés azon pontjainak halmaza, amelyeket elérhet a munkafolyamata alatt, más szóval az a tér, ahol a robot mozog, ez lehet 3D illetve 2D-s terület. A robot munkaterülete függ a robot karjainak hosszúságától és azok mozgékonyaságától. Ezek a területek meghatározása nagyon fontos a programozásuk során, mert csak abba a területbe tudnak eljutni, ami beletartozik ebbe a pontok halmazába, valamint veszélyes lehet mások számára ebben a területben tevékenykedni miközben a robot dolgozik.

Az alábbi ábra megmutatja a Baxter robot jobb/bal karjának 3D-s munkaterületét, amit Forward Kinematics Problem (FKP)<sup>8</sup> iterációval lett létrehozva MATLAB környezetben. A matlabos változatban is jól látható, hogy az elérhető munkaterületben jelentős átfedés van a két kar között. Ez az átfedés jól látható a 8. ábrán. Ami észrevehető, hogy az hivatalos Rethink munkaterülete bár jól szemlélteti a munkaterületet, az nem méretarányos, szóval az alábbi képen a méretarányos terület látható.

<sup>7</sup> [https://www.ms.sapientia.ro/~martonl/Docs/Lectures/Robotok\\_Geometria\\_Kinematikaja](https://www.ms.sapientia.ro/~martonl/Docs/Lectures/Robotok_Geometria_Kinematikaja)

<sup>8</sup> <https://ljvmiranda921.github.io/notebook/2017/01/25/forward-kinematics-standford-manipulator/>

A FKP egy olyan matematikai módszer, ami a robotkarok mozgásának számítására szolgál, célja az, hogy megtalálja a robotkarok végpontjainak az összes pozícióját és orientációját a csuklók alapján.



Ábra 8 Matab által generált munkaterület [10]

Az inverz kinematika a robotok geometriájának és kinematikájának egy fontos területe, amely lehetővé teszi az adott végberendezések pozíciójának és orientációjának alapján a csuklók kiszámítását. Az elvárt végberendezés pozícióját a bázis koordináta-rendszerhez képest adjuk meg, és az orientációt a roll-pitch-yaw (R-P-Y) vagy Euler szögekkel határozzuk meg. A 6 szabadságfokkal rendelkező robotok nagyon elterjedtek az iparban, és ezek esetében hat egyenletből álló egyenletrendszert kell megoldanunk. A kapott egyenletrendszer nem lineáris, ezért numerikus módszereket kell alkalmaznunk a megoldáshoz. Gyakran előfordul, hogy az elvárt pozícióhoz több csuklókonfiguráció is tartozik, és ilyen esetben azt a megoldást választjuk, amely a legtávolabb van a munkaterület határaitól<sup>9</sup>.

Az inverz kinematika implementációja megtalálható a Baxter robot alap programjai között, és ezeket a MOVEIT<sup>10</sup> csomag segítségével oldják meg.

<sup>9</sup> [https://www.ms.sapiientia.ro/~martonl/Docs/Lectures/Robotok\\_Geometriaja\\_Kinematikaja](https://www.ms.sapiientia.ro/~martonl/Docs/Lectures/Robotok_Geometriaja_Kinematikaja)

<sup>10</sup> [https://github.com/RethinkRobotics/baxter\\_examples/blob/master/scripts/ik\\_service\\_client.py](https://github.com/RethinkRobotics/baxter_examples/blob/master/scripts/ik_service_client.py)



---

#### 5.1.4. Baxter kamerái

A Baxter robotnak 3 kamerája van. Ezek a kamerák a következő képpen helyezkednek el<sup>11</sup>. Egy kamera található a Baxter robot tetején (fején) és kettő a két karjai végpontján. A kamerák csuklók jobb felső sarkán kaptak helyet, amiknek a maximális felbontásuk 1280x800 pixel, de ebből az effektív felbontás az 640x400 pixel. Ezek kamerák rögzítési sebessége 30 fps (képkocka/másodperc), ami elegendő a projektünk elvégzéséhez. A kamerák fókusz távolsága pedig 1,2 mm. Ezen kamerák kívül a karok még számos más szenzorral vannak felszerelve. Található rajtuk még egy infravörös szenzor, ami a kamerák irányába figyel, amivel képes érzékelni távolságokat 4-40 cm között. A 9. ábra szemlélteti a kamera és szenzor elhelyezkedését a vég berendezésben. Ezen kívül a végberendezés tartalmaz még egy-egy háromtengelyű gyorsulásmérő szenzort is [10].



*Ábra 9 Baxter kamera és infravörös szenzor [11]*

#### 5.1.5. Gripperek

A Baxter robotnál két lehetőség van tárgyak fogására: egy elektromos fogóval és egy szívókoronggal, amelyeket a Rethink cég kínál. A projektben mind a két fogónak nagy szerepe van. Az elektromos fogó két ujjal rendelkezik, amik segítségével könnyedén lehet tárgyakat mozgatni. Ez a fogó eltávolítható/cserélhető betétekkel rendelkezik. Ezek a betétek lehetővé teszik a tárgyak pontosabb tartását és azok biztosabb megfogását. A betéteknek köszönhetően többféle

---

<sup>11</sup> [https://sceweb.sce.uhcl.edu/harman/CENG5931Baxter2015/Guides/BAXTER\\_Introduction\\_2\\_08\\_2016.pdf](https://sceweb.sce.uhcl.edu/harman/CENG5931Baxter2015/Guides/BAXTER_Introduction_2_08_2016.pdf)

---

konfigurációval lehet használni. Kívülre és belülre is egyaránt elhelyezhetőek, aminek segítségével többféle tárgyat tudunk mozgatni. A fogó 144 mm-re nyitható ki, és a fogók konfigurációjával bármilyen tárgyat megtudunk ragadni ezen tartomány belül. A fogók még erőszabályzókkal is el vannak látva, amik elősegítik a merev és kevésbé merev tárgyak mozgatását is.

A másik lehetőség az, amikor a szívókorongot használjuk. A projektben ez a megfogási mód jelentősebb részt tesz ki. A vakum korongos fogó működése a vákuum elvén alapszik. A korongot a felemelendő tárgy fölé helyezzük, majd rátesszük a tárgy felületére. Ezt követően kiszívjuk a levegőt a korong alól, így vákuumot hozunk létre. Ez a vákuum egy erős tapadóerőt hoz létre a korong és a tárgy között, amely a tárgy könnyedén való mozgását eredményezi. A fogót egy külső légtartály (kompresszor) táplálja, amely biztosítja a szükséges légáramlást a vákuum létrehozásához. Ez a vákuumos szívókorong jól működik sima, vagy lapos tárgyakon, amelyek felületén létrehozható a szükséges vákuum.



*Ábra 10 A baxter robot elektromos és szívó markolója [12][11]*

### **5.1.6. ZMQ**

A ZeroMQ (más néven ØMQ, ZMQ, vagy 0MQ) egy nyílt forráskódú üzenetküldő könyvtár, ami lehetővé teszi az alkalmazások, rendszerek közötti kommunikációt. Ez a kommunikáció lehet szinkron és aszinkron is. A ZeroMQ-t gyakran használják összetett rendszerekben, ahol nagy gyorsasággal kell egymással az alkalmazások kommunikálniuk. Ennek a segítségével a programok egyszerűen és megbízhatóan tudnak küldeni és fogadni üzeneteket anélkül, hogy fizikailag

---

kapcsolatban lennének egymással. A ZeroMQ többféle kommunikációs mintát támogat, ami sokoldalú felhasználást nyújt ennek a rendszernek. Támogatja a kérdés-válasz (request/reply), kiadó-előfizető (publish/subscribe) valamint a push-pull mintát is. Ezek a minták lehetővé teszik a rugalmas kommunikációt a modulok / alkalmazások között, amivel könnyedén tudják alkalmazkodni a fejlesztő igényeihez.

A ZeroMQ egy socket alapú kommunikációs rendszert nyújt. A kommunikáció felépítéséhez egyik oldalon az alkalmazásnak egy küldő socket kell létrehoznunk, a másik oldalon meg egy fogadót. Ezeken a socketeken fog a kommunikáció végbemenni. Az üzenetek küldésére alapvetően több protokollon történhet, például TCP/IP (Transmission Control Protocol/Internet Protocol, az interneten használt leggyakoribb kommunikációs protokoll) vagy az IPC (Inter-Process Communication), amely a folyamatok közötti kommunikációra szolgál. Az IPC lehetővé teszi a folyamatok közötti adatok és erőforrások megosztását, valamint az üzenetek, jelzések és szinkronizáció kialakítását.

A socketek egy olyan végpont a hálózatban, amin keresztül kommunikáció történik. Ezek a végpontok adatok küldésére és fogadására szolgálnak. A socketeket általában IP-címekkel és portszámokkal használják, amelyek megadják a kommunikáció menetét. A ZMQ-ban több socket típust különböztethetünk meg, mint például a ZMQ::REQ, aminek segítségével kéréseket tudunk küldeni és válaszokat tudunk fogadni. Ezen felül még megtalálható a (ROUTER and DEALER Socket, PUB and SUB Sockets, PAIR Sockets, PUSH/PULL Sockets)<sup>12</sup>

A robotok esetében a ZMQ-t gyakran használják a robot irányításához és a különböző szoftverkomponensek közötti adatok és parancsok átadásához. Például a robot irányítórendszerének lehet ZMQ alapú kommunikációs csatornája, amelyen keresztül a felhasználó parancsokat küldhet a mozgástervezőnek, és a mozgástervező visszajelzéseket adhat a felhasználónak.

Még egy fontos előnye a ZMQ-nak hogy többcsatornás kommunikációt is támogat, ami elengedhetetlen a hatékony működés végett. Ez a többszálás kommunikáció lehetővé teszi az aszinkron kommunikációt a szoftverkomponensek között és a felhasználó között. Ez a kommunikáció segítségével a robot és a felhasználó egyszerre több feladatot tud elvégezni és különböző egymástól független komponensekkel tud kommunikálni.

---

<sup>12</sup> <https://intelligentproduct.solutions/technical-software/introduction-to-zeromq/>

---

### 5.1.7. OpenCV

Az OpenCV<sup>13</sup> (Open Source Computer Vision Library) egy nyílt forráskódú könyvtár, amelyet a számítógépes látás és gépi tanulás területén használnak. Az OpenCV projekt hivatalosan 1999-ben indult, mint az Intel kutatási kezdeményezése a CPU-intenzív alkalmazások fejlesztésére. Ez a projekt része volt több más kezdeményezésnek is, beleértve a valós idejű sugárnyomtatást és a 3D megjelenítő falakat. Az OpenCV-t azért hozták létre, hogy közös infrastruktúrát biztosítson a számítógépes látás alkalmazásai számára, ezzel felgyorsítva és egységesítve a számítógépes látás szerkezetét. Ez a könyvtár teljesen nyílt forráskódú és az Apache 2 licenc alatt álló termékként lehetővé teszi a szabad felhasználást és módosítást.

Az OpenCV több mint 2500 beépített algoritmussal rendelkezik, amelyek között megtalálhatók a klasszikus és a legmodernebb számítógépes látás és gépi tanulás algoritmusok. Ennek a könyvtárnak több mint 47 ezer felhasználói közössége van, és a letöltések száma meghaladja a 18 milliót, ami a legnépszerűbb gépi látás könyvtárnak számít 1. A könyvtárat széles körben használják egyéni, kisebb és nagyobb vállalkozások is, például a Google, Yahoo, Honda és Toyota.

Ennek a könyvtárnak a célja a képek és videók elemzése, manipulációja és értelmezése, valamint különböző látással kapcsolatos feladatok végrehajtása (például objektumfelismerés, mozgásérzékelés, arcfelismerés és képfeldolgozás). A leggyakrabban használt algoritmusok segítségével képesek vagyunk éleket detektálni a képeken, szűrőket alkalmazni, objektumokat/arcokat értelmezni és felismerni. Ezek a funkciók egyaránt alkalmazhatók képi és videó formátumokra.

A könyvtárat több programozási nyelvben is használhatjuk, mint például a C++, Python és Java, hogy csak a legnépszerűbbeket említsem 2. Ez nagy rugalmasságot biztosít a fejlesztőknek, akik könnyedén a megszokott környezetükben élvezhetik ennek az előnyeit.

Az OpenCV könyvtár használata nagyon népszerű a robotika területén. A robotok elhelyezkedésének és környezetük értelmezésének szempontjából kulcsfontosságú szerepet játszik. A robot kamerájából érkező kép feldolgozását az OpenCV végzi el. Ennek segítségével lehetőségünk van az asztal tetején lévő kockák azonosítására és pozíciójuk meghatározására. Ez jelentős előrelépést jelent a robotikában, mivel lehetővé teszi a robot számára, hogy interakcióba lépjen környezetével és feladatokat hajtson végre. Az OpenCV megkönnyíti a fejlesztők számára a látási rendszer implementálását.

---

<sup>13</sup> <https://opencv.org/>

---

### 5.1.8. ROS

A Robot Operating System<sup>14</sup> (ROS) egy nyílt forráskódú operációs rendszer, amelyet a robotok számára hoztak létre. Ez az operációs rendszer a robotika kutatásokra és fejlesztésekre támogatására hozták létre. A fő célja az, hogy egy egységes keretrendszert biztosítson azok számára, akik robotikával foglalkoznak, ez egy egyszerű platform, ahol könnyedén lehet fejleszteni és megosztani alkalmazásokat. A ROS segítségével a fejlesztők könnyedén tudnak algoritmusokat és kódokat megosztani egymással. Előnye, hogy sok előre beépített funkciót tartalmaz, ilyenek például az arcfelismerés, mozgáskövetés, sztereó látás (több kamera segítségével), robotvezérlés, útvonaltervezés.

A nevében is szereplő operációs rendszer (OS) kifejezés téves, mert valójában nem egy operációs rendszert helyettesít, hanem csak egy keretrendszert biztosít a fejlesztőknek. A ROS egy Linux alapú rendszer, amely lehetővé teszi az alacsony szintű eszközevezérlést. Az operációs rendszer alapvető feladata a hardver és a szoftver közötti kommunikáció kezelése, valamint a rendelkezésre álló erőforrások megfelelő elosztása. A ROS nem vállalja ezeket a feladatokat, hanem olyan keretrendszert nyújt, amely megkönnyíti a fejlesztést. Általában a ROS-t egy Linux operációs rendszeren futtatják.

A ROS egy amerikai fejlesztésű rendszer, amelyet 2007<sup>15</sup>-ben a Stanford Egyetemen terveztek, és azóta már hatalmas népszerűségnek örvend. A keretrendszerük elősegíti a kód újrafelhasználást és a csapatos együttműködést a robotok területén<sup>16</sup>.

A kommunikációhoz a publisher/subscriber modellt használják a szoftveres komponensek között. Ezeket a szoftveres komponenseket node-oknak nevezik. Ezek a node-ok lehetnek szenzorokat vezérlő, adatfeldolgozást végző egységek. A node-ok topic-okon keresztül kommunikálnak egymással. Egyes node-ok publikálhatnak a topic-okra, míg mások feliratkoznak rá és adatokat kapnak róluk. Ezek a topic-ok segítenek az adatok megosztásra a rendszerben. Ezen felül megtalálható egy Master szolgáltatás, ami nyilvántartja a rendszerben futó node-okat és a rendelkezésükre álló topic-okat. A rendszer indításakor a master látja a node-okat, így lehetőség nyílik azok közötti kommunikációra. Összegezve az indításkor a node-ok csatlakoznak a masterhez, és azon keresztül tudnak adatokat küldeni és fogadni a topic-okon keresztül.

---

<sup>14</sup> <https://www.ros.org/>

<sup>15</sup> [https://users.iit.uni-miskolc.hu/~tomba/KorszeruInfTech\\_lev/ROS\\_segedlet.pdf](https://users.iit.uni-miskolc.hu/~tomba/KorszeruInfTech_lev/ROS_segedlet.pdf)

<sup>16</sup> [https://hu.frwiki.wiki/wiki/Robot\\_Operating\\_System](https://hu.frwiki.wiki/wiki/Robot_Operating_System)

---

A ROS rendszer előnye még az, hogy támogatja a párhuzamos végrehajtást is, ami azt jelenti, hogy egy időben több node-ot is tud kezelni/futtatni, így párhuzamosan tudnak ezek kommunikálni egymással. Ez lehetővé teszi a nagyobb és összetettebb alkalmazások fejlesztését is.

Összegezve a ROS számos előnyt kínál a robotika fejlesztők számára. Elsősorban az újra felhasználható szoftverkomponensek nagy választékával rendelkeznek, amelyek segítségével egyszerűen építhetünk összetett robotika alkalmazásokat. Másodszor, a ROS szorosan integrálódik a robotplatformokkal és érzékelőkkel, ezáltal könnyűvé teszi a hardver- és szoftverintegrációt.

### **5.1.9. ArUco**

Az Augmented Reality University of Cordoba (ArUco) kódok olyan mesterségesen generált jelölők, amelyeket a számítógépes látás és gépi tanulás területén használnak. Ezek a kódok széles körben alkalmazhatók a robotika területén, és segítségükkel a robotok képesek felismerni és lokalizálni a környezetükben található tárgyakat. Az ArUco kódokat először Rafael Muñoz és Sergio Garrido-Jurado vezette kutatócsoport dolgozta ki a spanyol Málaga Egyetemen. Céljuk az volt, hogy kifejlesszenek egy egyszerű, könnyen felismerhető és követhető kódrendszert a számítógépes látási rendszerek számára. Az ArUco kódokat 2014-ben vezették be, és azóta széles körben alkalmazzák a robotika, az augmented reality (kibővített valóság) és a gépi látás területén.

Az ArUco kódok [13] a fiducial [14] jelölők egyik jól ismert típusa. A Fiducial jelölők olyan jelek vagy tárgyak, amelyeket a képfeldolgozó rendszerek könnyen fel tudnak ismerni és azokat olvasni. Ezek a jelölők általában egyszerű geometriai alakzatokat tartalmaznak jól előtérbe szíkülönbséggel és úgy vannak kitalálva, hogy a képfeldolgozó rendszerek számára könnyen észlelhetők legyenek (11. Ábra). Az egyik legismertebb ilyen Fiducial jelölő az a QR code. Az aruco is hasonló a Qr codehoz és ugyan azt a jelölő családot képviseli. Fehér négyzetek egy feketén háttéren.

Ezeknek az aruco kódoknak kapacitásuk nagyon kicsi, leginkább csak egy számot (id) t jelölnek. Felhasználásuk igencsak eltérő lehet, nagy szerepük van a kiterjesztett valóságban, valamint a robotikában ahol ezek segítségével könnyen lehet a kódokat a térben pozicionálni.

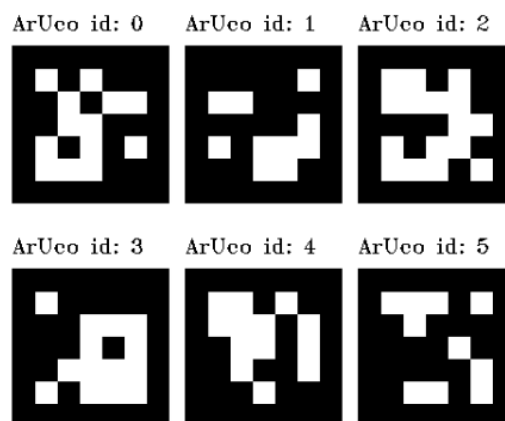
Az aruco kód felismerő algoritmusok pontosan fel tudják ezeket a kódokat ismerni és meg tudják határozni a beléjük kódolt számokat. Az algoritmusok képesek a térbeli pozíció meghatározására és az forgatási mátrix megadására.

Fontos még megjegyezni, hogy ezek a kódok Dictionary-ba helyeződnek. Ezek a dictionaryk segítenek a kódok pontos meghatározásában, a dictionaryk egy számot jelentenek, ami meghatározza a kód mátrixának méretét, hogy hányasor hányas legyen. A felismerő algoritmusban mindig meg kell adni, hogy a felismerendő code milyen könyvtárba tartozik. Ennek nagy előnye az hogyha egy projektben több ilyenfajta codot szeretnénk használni akkor csak azokat veszi figyelembe, amelyeket mi előre meghatároztunk, a többit ignorálja.

A munkám során ezek az ArUco kódok felhasználásával határozom meg a kockák pontos térbeli helyeit, és a felismerő algoritmusok segítségével megkapom a koordinátákat és a forgatási mátrixot.

Ezek az alkalmazott eszközök együttesen lehetővé teszik a Baxter robot számára, hogy felismerje és mozgassa a kockákat a kamera segítségével. A kamera érzékeli a kockák helyzetét, majd az OpenCV segítségével feldolgozza a képeket, hogy azonosítsa a kockákat és meghatározza a helyzetüket a térben. Az ArUco kódok további információkat szolgáltatnak a kockák azonosításához és pontos pozíció meghatározásához. A vezérlőegység irányítja a Baxter robot karjait és megfogó eszközét, hogy felvegye és helyezze el a kockákat a megfelelő helyre.

Ezen eszközök használatának előnyei közé tartozik a hatékonyság és a precizitás növelése. A robot képes nagy pontossággal felismerni és mozgatni a kockákat, ami csökkenti az emberi hibalehetőségeket és növeli a termelékenységet. Emellett a robotok és az alkalmazott eszközök rugalmasan alkalmazkodnak a változó környezeti feltételekhez és feladatokhoz, így sokoldalúan felhasználhatók.



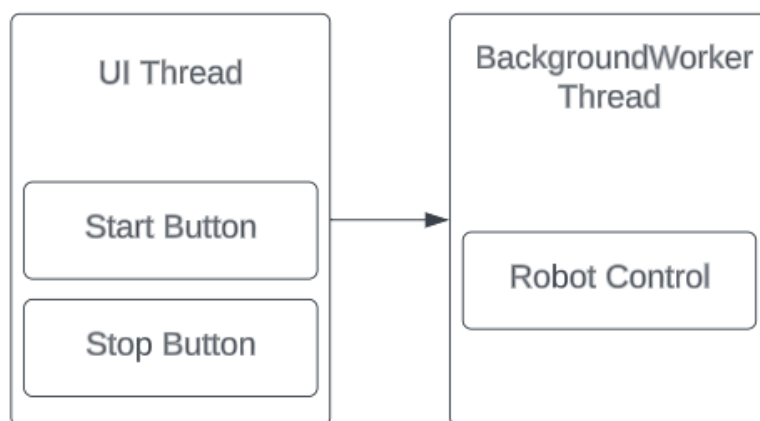
*Ábra 11 ArUco kód példa*

---

## 5.2. A Client felöli rész

A felhasználói rendszer a fő elem ebben a projektben. Ez végzi el minden olyan utasítást, amit a felhasználótól kap, vagy amit a feladat megkövetel. A felhasználó gépén futó utasítások sorozata alkotja a működő robot munkafolyamatát. Itt történik az adatok fogadása és azok értelmezése, valamint a robot irányítása és ellenőrzése.

Ez a felület két fő részből áll. Az egyik a felhasználó által is jól látható kezelőfelület, ami igazán egyszerű és érthető módon lett felépítve, nem túl bonyolult, mindössze két részből áll, egy elindításhoz szükséges gombból és a munkafolyamat lezárásáért felelős gombból. Ez azért helyeztem el, mert ha valami olyasmi történik a munkafolyamat alatt, ami nem várt eredményeket hozhat, akkor azzal a gombbal automatikusan leállítható a munkafolyamat, és a robot a kezdő pozíciójába fog visszatérni. A másik rész a háttérben futó programozási környezet, ami biztosítja a kommunikációt a robot és a számítógép között, valamint tartalmazza azokat az algoritmusokat és függvényeket, amelyek szükségesek a robot mozgásának és viselkedésének meghatározásához.



Ábra 12 UI+BackgroundWorker diagram

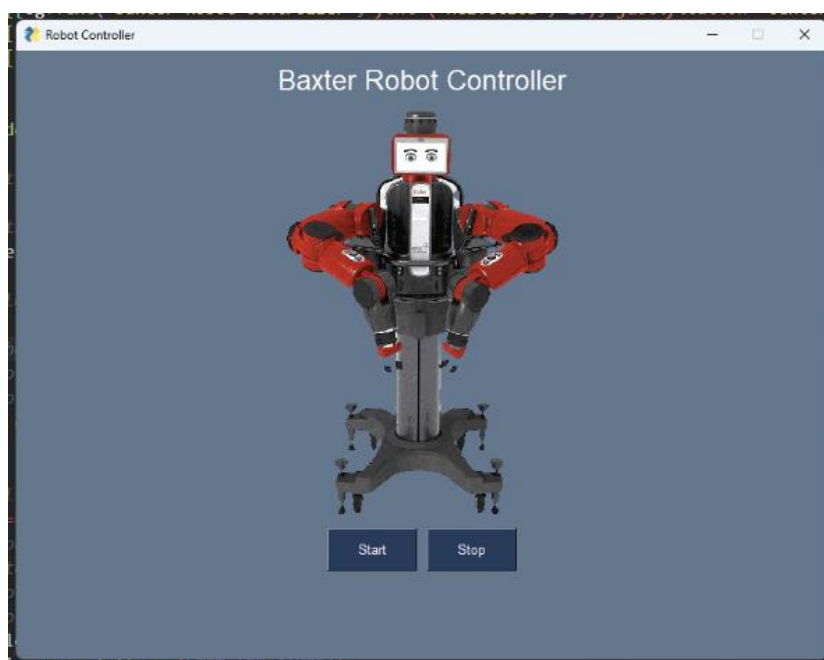
### 5.2.1. A Client felöli UI leírása

A grafikus felhasználói felületet egy python nyelvű könyvtár segítségével hoztam létre. A PySimpleGUI nevű könyvtár egy erős és egyszerű eszköz, amellyel gyorsan és könnyedén lehet vizuális interfészeket készíteni. A könyvtár lehetőséget ad a programozóknak arra, hogy interaktív alkalmazásokat fejlesszenek ki hatékonyan és rugalmasan.



A PySimpleGUI<sup>17</sup> rendkívül sokoldalú és adaptív könyvtár. Könnyedén integrálható a Python alapú projektekbe, és kifejezetten a könnyű és alapvető feladatokra lett kifejlesztve. Kompatibilis a legnépszerűbb operációs rendszerekkel, mint például a Windows, a macOS és a Linux. Így a programozók olyan felhasználói felületet tudnak létrehozni, amely megfelel a felhasználók igényeinek és elvárásainak. A könyvtár egyszerűsége miatt könnyen használható, akár kezdő vagy haladó szintű programozók is. Emellett rengeteg példa és dokumentáció áll rendelkezésre a könyvtár használatához.

A grafikus felület, amit készítettem, letisztult és átlátható. Az elemek logikus elrendezésben vannak, és világos és érthető utasításokat tartalmaznak (13. ábra). Ez a felület segíti a felhasználót abban, hogy könnyen kommunikáljon a rendszerrel és javítsa a felhasználói élményt.



Ábra 13 A grafikus felület

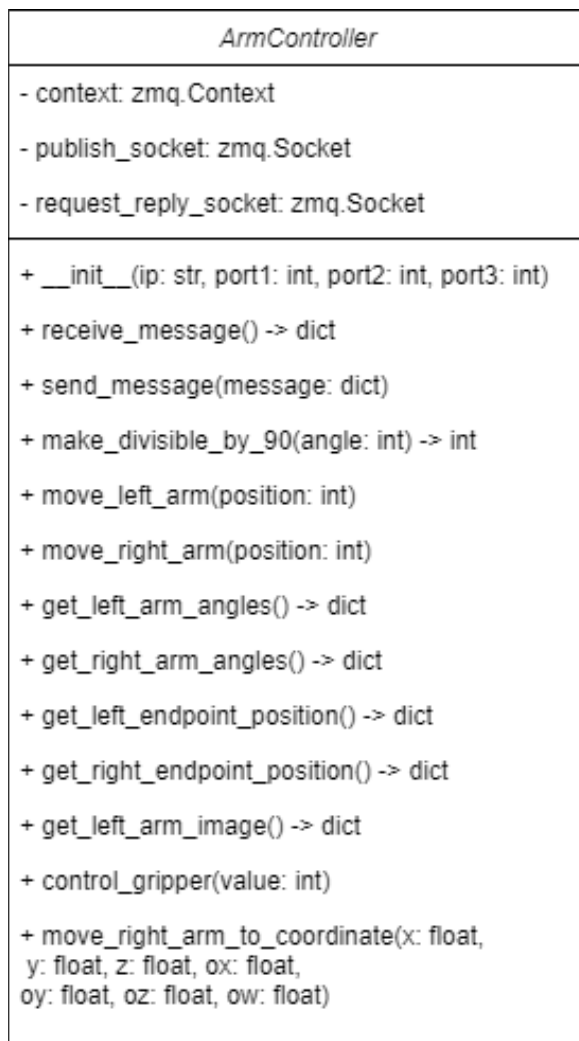
<sup>17</sup> <https://www.pysimplegui.org/en/latest/>

---

### 5.2.2. A háttérben futó rendszer leírása

A felhasználó gépen futó rendszer Python nyelven lett megvalósítva. A Python környezetében számos könyvtárral találkozunk, amelyek megkönnyítették a munkafolyamat lebonyolítását. Ezek a könyvtárak jelentősen hozzájárultak az elvégzendő feladatok hatékonyságához. A rendszer kezeli az összes háttérben zajló munkafolyamatot, és egyszerre több műveletet végez. Ezek közé tartozik a lépéssorozatok végrehajtása és a logfájl létrehozása is.

A rendszer egyszerűsége és átláthatósága érdekében létrehoztam egy osztályt, amely tartalmazza az összes szerverrel való kommunikációhoz szükséges metódusokat. Ezek a metódusok nagy mértékben segítettek és jelentősen hozzájárultak a rendszer letisztultságához és egyszerűsítéséhez. A `controller` változó példányosításával hozzáfértem annak összes metódusához, és ez volt felelős az összes kommunikációért.



Ábra 14 ArmController osztály diagramja

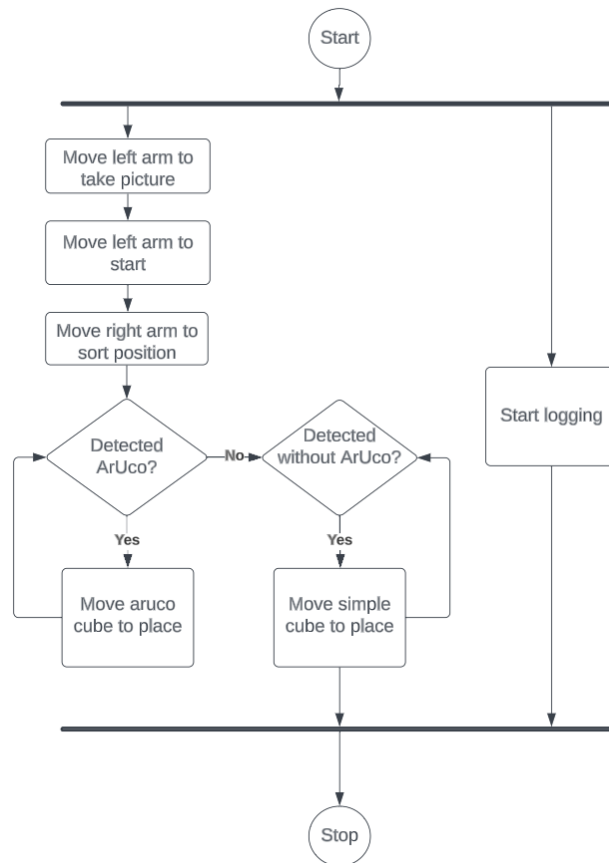
---

A 14. ábrán látható ArmController nevezetű osztály az alábbi funkciókat tölti ki:

- Attribútumok:
  - context: A zmq.Context objektum.
  - publish\_socket: Ezen tudom fogadni a szerverről jövő üzeneteket.
  - request\_reply\_socket: Ezen tudok kommunikálni a szerverrel.
- Metódusok:
  - receive\_message: fogadja az üzeneteket a socketen keresztül.
  - send\_message: küld egy üzenetet a socketen keresztül.
  - make\_divisible\_by\_90: a forgatási szöget beállítja olyan értékre ami osztható 90-el.
  - move\_left\_arm/Move\_right\_arm : a karok mozgatásáért felelősek.
  - get\_left\_arm\_angles/Get\_right\_arm\_angels: Visszaadja a jobb es bal kar szögeit.
  - get\_left\_endpoint\_position/Get\_right\_endpoint\_position: Visszaadja a jobb es ball végpontok pozícióit.
  - get\_left\_arm\_image: Visszaadja a bal kar kamerájából származó képét numPy tömbként.
  - control\_gripper: Kezeli a karok markolóit.
  - move\_right\_arm\_to\_coordinate: A jobb kart egy megadott koordináta-hoz mozgatja .

Miután a példányosítás megtörtént, szükségem volt néhány további függvényre, amelyek a képfeldolgozásért voltak felelősek. Két különálló függvény került létrehozásra. Az egyik felelős az ArUco kódok detektálásáért, míg a másik egy élkeresésen alapuló OpenCV kódot tartalmazott. Ezeknek a függvényeknek a feladata az volt, hogy visszatérítsék a kockák központi koordinátáit.

A 15. ábrán látható a felhasználói rendszer működési elve. A folyamat elindulásakor egyszerre 2 szál indul el, egyiken a naplózás míg a másikon a munkafolyamat zajlik. A robot ball kezével csinál egy képét, és ha van rajta ArUco kóddal ellátott kocka akkor azokat helyre rakja, ha már nincs több ilyen akkor újból csinál egy képét és helyre rakja az ArUco kóddal nem rendelkező kockákat. Ez a folyamat lejárta után vég állapotba megy és ilyenkor a naplózás is befejeződik.

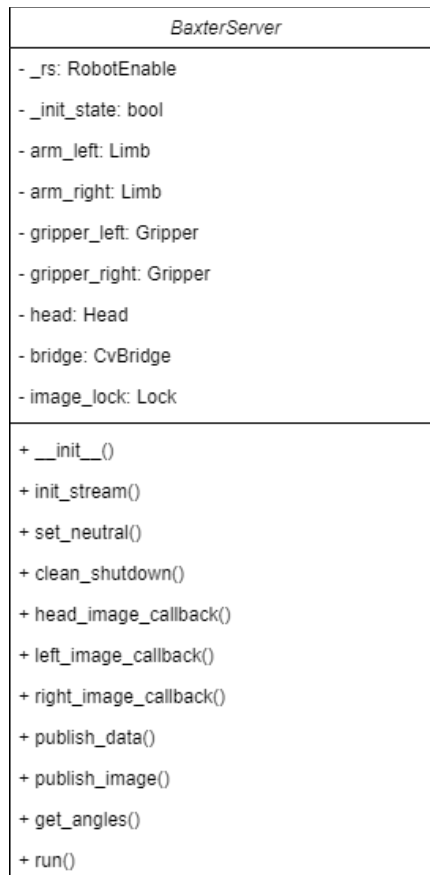


Ábra 15 A felhasználói rendszer aktivitás diagram

Ezek a függvények és az `ArmController` osztály együttműködve biztosították a kommunikációt a felhasználói rendszer és a szerver között, és lehetővé tették a robot mozgását és viselkedését.

### 5.3. A szerver felőli rész

A BaxterServer egy python program, amely egy szerverként működik a Baxter robot és a felhasználói rendszer között. A szerver folyamatosan kommunikál a robotkarokkal, a gripperekkel, valamint képi és adatfolyamatokat közvetít a felhasználó felé. A 16. ábrán látható a szerver osztálydiagramja, amely inicializáláskor csatlakozik a Baxter robotra, majd engedélyezi azt és beállítja a kezdőállapotokat. Ezek után feliratkozik a kameraképre és inicializálja a kommunikációt a szerver és a kliens között. A szerver képes adatokat közvetíteni a robot állapotáról, például a karok és a gripperek pozíciójáról, orientációjáról és nyomásérzékeléséről. Emellett lehetőségünk van vele kapcsolatba lépni a robotkarok mozgatására, valamint a markolók aktiválására vagy deaktiválására. A szerver továbbá képes kezelni a különböző üzeneteket és parancsokat, amelyeket a kliens küldhet neki.

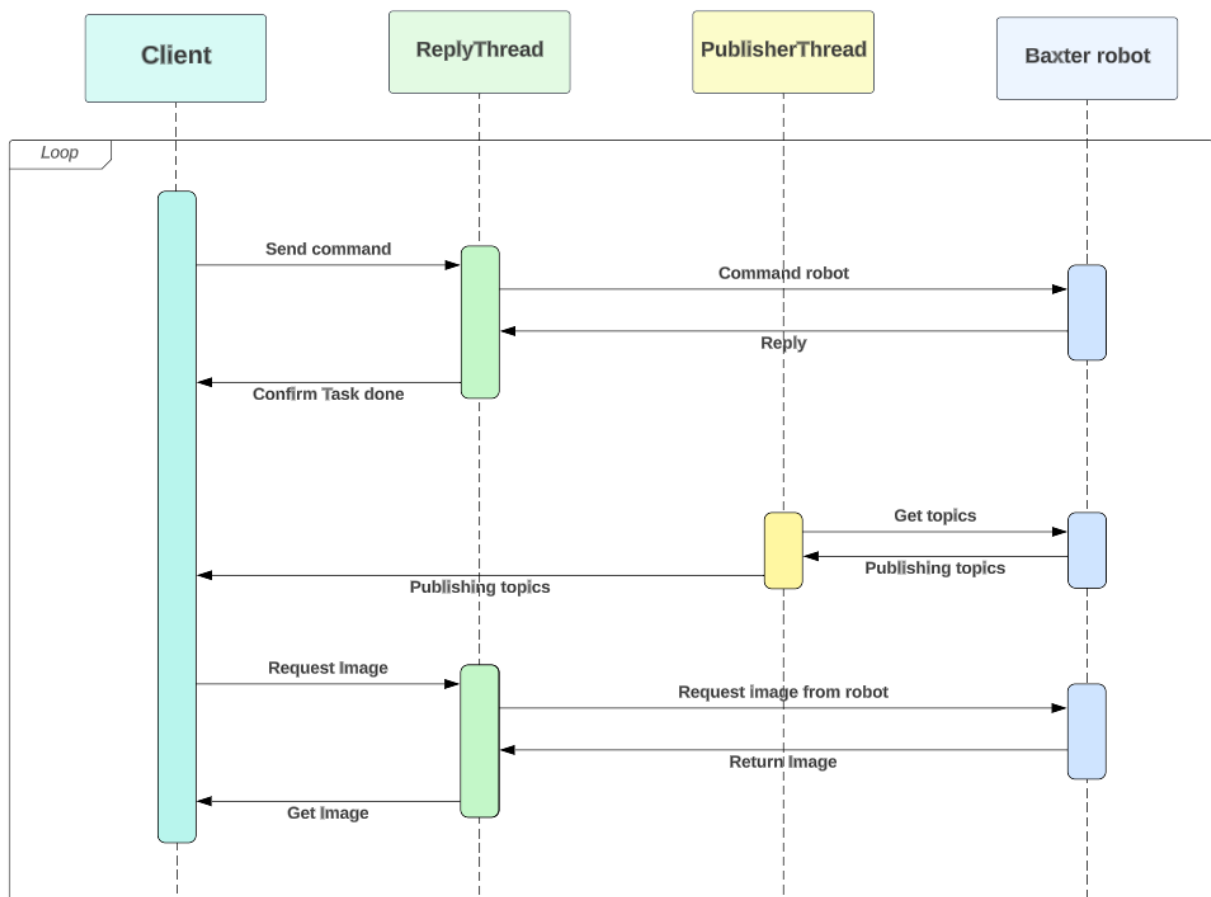


*Ábra 16 A BaxterServer osztály diagramja*

A 16. ábrán látható BaxterServer nevezetű osztály az alábbi funkciókat tölti ki:

- Attribútumok:
  - - \_rs: A Baxter robotot engedélyező objektum.
  - arm\_left / arm\_right: A bal és a jobb kar objektuma.
  - gripper\_left / gripper\_right: A bal és a jobb markoló objektuma.
  - bridge: Egy átalakító objektum a képek konvertálásához.
  - image\_lock: Egy zárolási objektum a képek szinkronizálásához.
- Metódusok:
  - init\_stream: Itt inicializálódnak a socketek.
  - set\_neutral: Az alaphelyzetbe állítja a robotkarokat.
  - clean\_shutdown: Leállítja a programot és letiltja a robotot.
  - head/left/right\_image\_callback: Visszahívó függvény a fej/bal/jobb kéz kameraképének fogadásához és feldolgozásához.
  - publish\_data: Publikálja a robot állapotára vonatkozó adatokat.
  - publish\_image: Publikálja a bal kéz kameraképet az egyik socketen keresztül.
  - get\_angles: Lekéri és publikálja a karok szögét és sebességét.
  - run: Elindítja a folyamatot és várakozik az üzenetekre, majd végrehajtja azokat.

A rendszer szekvencia diagramja bemutatja 17. ábra, hogy a szerver oldalon két szál folyamatosan működik. A PublisherThread-en keresztül a szerver csak a robottól érkező adatokat osztja meg a klienssel, ezeket folyamatosan küldi JSON formátumban. A másik szál a ReplyThread, amin keresztül a kliens és a szerver közötti kommunikáció történik. Itt küldünk egy JSON formátumban lévő parancsot a szervernek, amit ő feldolgoz és átküld a robotnak. Miután a folyamat elvégződött, küld egy választ a robot, amit továbbít a kliensnek. Ha ez a válasz nem érkezik meg, akkor újabb üzenetet nem tudunk küldeni neki. Ezen a szálon történik meg a képadatok lekérése is, csak itt nem egy válasz jön vissza, hanem maga a képadat, amivel további munkafolyamatokat végzek.



Ábra 17 A szerver működéséről szóló szekvencia diagram

---

Az üzenetek az alábbi struktúrába rendeződnek, ami a 18. ábrán látható. Megadjuk elsősorban, hogy melyik karral szeretnénk a munkafolyamatot elvégezni, majd, hogy pontosabban mit szeretnénk csinálni vele, majd az ehhez szükséges adatokat adjuk át vele.

```
cmd = {'left_arm':{  
    'move_to' : {  
        'right_s0': -0.7029466960484908,  
        'right_s1': -1.3341797902633385,  
        'right_w0': -0.034514567727421806,  
        'right_w1': 1.4488448541577732,  
        'right_w2': -0.13077186216723152,  
        'right_e0': 0.09012137128826805,  
        'right_e1': 1.4473108733698878  
    }  
}
```

*Ábra 18 A szerver felé küldött üzenet felépítése*

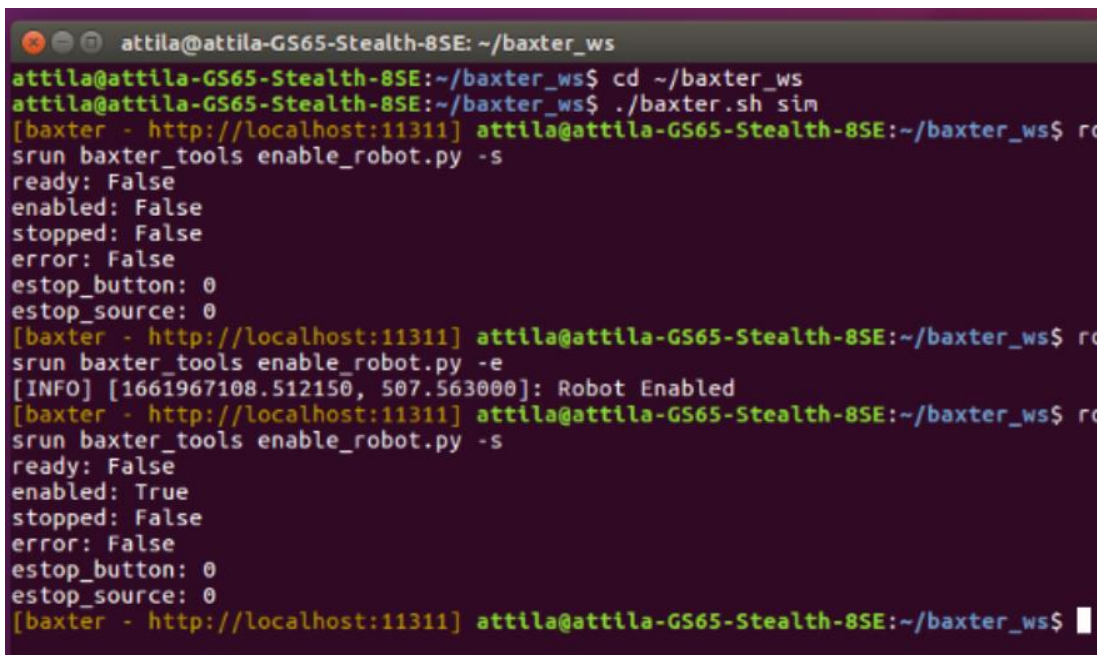
---

## 6. Megvalósítás

### 6.1.A Baxter robot szimulációs környezetben

A projekt megvalósítása során először a robot működésének dokumentációját tanulmányoztam. A projekt kivitelezésének első fázisában a robot működésének megértése volt a fő cél. Ehhez létrehoztam a saját gépemén egy megfelelő környezetet, amiben könnyedén tudtam vizsgálni a robot működését. Ehhez a munkakörnyezethez egy ROS-t kellett telepítenem. A telepítéshez szükségem volt egy Linux operációs rendszerre. Miután a rendszer települt, a ROS Kinetic telepítése volt a következő lépés. A ROS Kineticet azért választottam, mert ez az egyik legelterjedtebb ROS verzió, amely támogatja az Ubuntu 16 operációs rendszert. Emellett a könyv, amelyből a Baxter robot működését tanulmányoztam, szintén ROS Kinetic környezetet használt a példákban. Ebben nagy segítség volt a telepítési folyamatot leíró hivatalos oldal. Miután a környezet készen állt, a Baxter SDK-t kellett telepítenem a dokumentáció alapján.

A `ros-kinetic-full` csomagban megtalálható volt a Gazebo szimulátor program. Ebben a programban tökéletesen és könnyedén lehetett a robot mozgását szimulálni. Ebben a szimulációs környezetben tökéletesen látható volt a robot és annak minden mozgása. A környezetben elindításkor a robot kikapcsolt állapotban van, ami azt jelenti, hogy nem tud parancsokat fogadni tőlünk. Ezt az elindításkor engedélyezni kellett. A 19. ábrán a terminál található, amin a robot engedélyezése történik.



```
attila@attila-GS65-Stealth-8SE: ~/baxter_ws
attila@attila-GS65-Stealth-8SE:~/baxter_ws$ cd ~/baxter_ws
attila@attila-GS65-Stealth-8SE:~/baxter_ws$ ./baxter.sh sim
[baxter - http://localhost:11311] attila@attila-GS65-Stealth-8SE:~/baxter_ws$ roslaunch baxter_tools enable_robot.py -s
ready: False
enabled: False
stopped: False
error: False
estop_button: 0
estop_source: 0
[baxter - http://localhost:11311] attila@attila-GS65-Stealth-8SE:~/baxter_ws$ roslaunch baxter_tools enable_robot.py -e
[INFO] [1661967108.512150, 507.563000]: Robot Enabled
[baxter - http://localhost:11311] attila@attila-GS65-Stealth-8SE:~/baxter_ws$ roslaunch baxter_tools enable_robot.py -s
ready: False
enabled: True
stopped: False
error: False
estop_button: 0
estop_source: 0
[baxter - http://localhost:11311] attila@attila-GS65-Stealth-8SE:~/baxter_ws$
```

Ábra 19 A robot engedélyezése szimulációs környezetben





---

## 6.2.A Kamera kép átküldése és feldolgozása

A rendszerem egyik kulcsfontosságú eleme a kamera képének átvitele és feldolgozása. Ezt a feladatot a programom a ZMQ technológiával oldja meg, amely hatékony és megbízható kommunikációt biztosít a robot kamera képének átvitelére. Az alkalmazásban egyetlen kamerát használok, amely a robot bal karjára van szerelve, és a kamera képének átvitele a fő feladata. A programom feliratkozik a kamerától érkező ROS image üzenetekre, amelyeket különböző topikokban küldenek. Az üzeneteket OpenCV formátumba alakítom át, hogy könnyen feldolgozhatóvá váljanak a további műveletek számára.

```
def __init__(self):

    print("Initializing node... ")
    rospy.init_node("streamer_node")
    self._rs = baxter_interface.RobotEnable(CHECK_VERSION)
    self._init_state = self._rs.state().enabled
    print("Enabling robot... ")
    self._rs.enable()
    self.arm_left = baxter_interface.Limb('left')
    self.arm_left.set_joint_position_speed(1.0)
    self.arm_right = baxter_interface.Limb('right')
    self.arm_right.set_joint_position_speed(1.0)
    self.gripper_left = baxter_interface.Gripper('left')
    self.gripper_right = baxter_interface.Gripper('right')
    self.head = baxter_interface.Head()
    self.bridge = CvBridge()
    head_image_topic = 'cameras/head_camera/image/'
    rospy.Subscriber(head_image_topic, Image, self.head_image_callback)
    left_image_topic = 'cameras/left_hand_camera/image/'
    rospy.Subscriber(left_image_topic, Image, self.left_image_callback)
    right_image_topic = 'cameras/right_hand_camera/image/'
    rospy.Subscriber(right_image_topic, Image, self.right_image_callback)
    self.init_stream()

    self.image_lock = threading.Lock()

def left_image_callback(self, msg):
    self.left_img = self.bridge.imgmsg_to_cv2(msg, 'bgr8')
```

*Ábra 21 A szerver oldalon történő témákra való feliratkozás*

A szerver oldalon két szálát használok, ami jól látható a 17. Ábrán. Az egyik szál egy PUB (Publisher) socketet kezel, amely JSON formátumban folyamatosan közli a robot aktuális állapotát. A másik szál egy REP (Reply) socketet kezel, amelyen keresztül lehet kérni és fogadni a kamera képét. A két socket különböző porton keresztül csatlakozik a programhoz, hogy elkerülje az összekeveredést és biztosítsa a megbízható kommunikációt.

---

```
def init_stream(self):
    self.port = "15557"
    self.context = zmq.Context()
    self.socket = self.context.socket(zmq.PUB)
    self.socket.bind("tcp://*:%s" % self.port)

    self.port2 = "15558"
    self.reply_socket = self.context.socket(zmq.REP)
    self.socket2.bind("tcp://*:%s" % self.port2)
```

*Ábra 22 A socketek létrehozása*

Miután a csatlakozások megtörténtek, a program a REP socketen várja az érkező üzeneteket. Amennyiben az üzenet tartalmazza a "left\_arm" kulcsot, a program meghívja a "publish\_image" nevű metódust, amely felelős a kamera képének publikálásáért.

```
while not rospy.is_shutdown():

    msg = self.socket2.recv_json()
    print(msg)

    if 'left_arm' in msg:
        par = msg['left_arm']

        if 'image' in par:
            if par['image'] == 1:
                self.publish_image()
                image_sent = True
```

*Ábra 23 Képérés küldése*

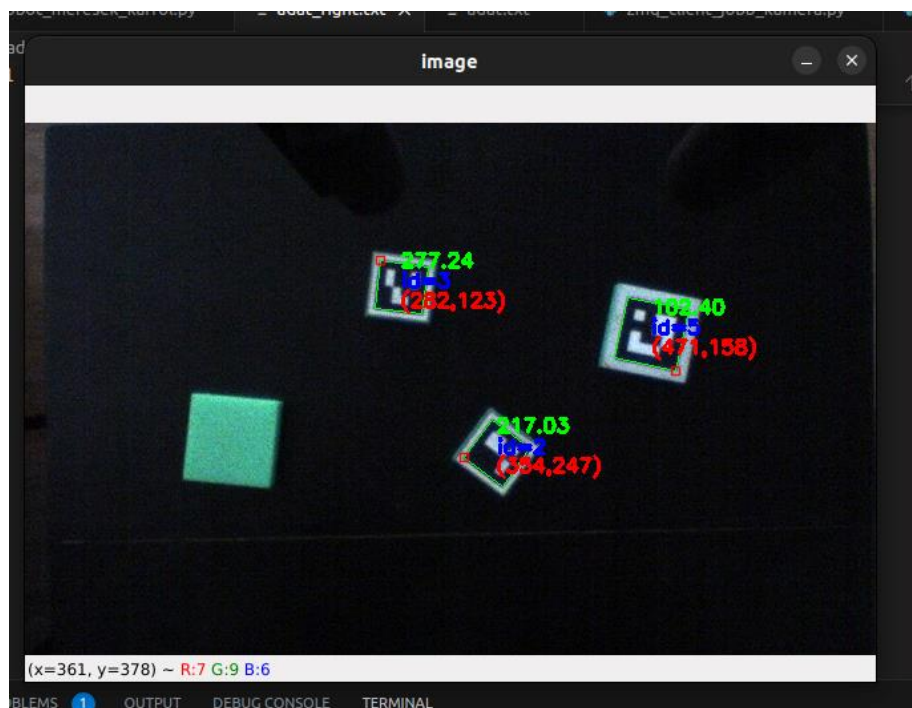
A "publish\_image" metódus lezárja az "image\_lock" nevű zárat, hogy megakadályozza a párhuzamos hozzáférést a képhez. Ezután elmenti a bal kéz kamerájának képét egy fájlba, majd Base64 kódolással átalakítja a képet karakterlánccá. Ez a karakterlánc lesz a válasz, amit a program visszaküld a socketen. A "image\_lock" zár feloldása után más szálak is hozzáférhetnek a képhez. A "image\_lock" szükséges annak elkerülésére, hogy egyszerre több kép érkezzen a topik révén. Ez elengedhetetlen, hogy ne alakuljon ki versenyhelyzet a képérésében. Ha egyszerre több szál próbálja olvasni a bal kéz kameráját, akkor az adatok sérülhetnek, ezért a "image\_lock" biztosítja, hogy csak egy szál férjen hozzá a topikhoz, és elkerülje a problémákat.

```
def publish_image(self):
    self.image_lock.acquire()
    cv2.imwrite("camera.jpeg", self.left_img)
    filename = "camera.jpeg"
    with open(filename, "r") as image_file:
        encoded_string = base64.b64encode(image_file.read())
    image_file.close()
    self.socket2.send_string(encoded_string)
    time.sleep(0.5)
    self.image_lock.release()
```

*Ábra 24 A szerver oldalon történő kép lekérése és küldése*

A felhasználó oldalon a program egy JSON üzenetet küld a REP socketen keresztül, amelyben a bal kéz kamerájának képét kéri. A robot válaszul egy Base64 karakterláncot küld vissza. A program dekódolja ezt a karakterláncot és OpenCV formátumra alakítja, hogy további feldolgozásra alkalmassá váljon.

Az így kapott képen a program meghívja a "detect\_aruco\_markers" nevű függvényt, amely felelős az összes ArUco marker felismeréséért. Ebben a függvényben a cv2.aruco könyvtárat használom, amely rendelkezik beépített detektáló algoritmussal. Először a képet szürkeárnyalatossá alakítom, hogy könnyebb legyen a detektálás. Az ArUco kódok megtalálása után létrehozok egy "marker" nevű listát, amelyben tárolom a kódok azonosítóját, koordinátáit és szögeit. A marker középpontját az ArUco kód sarokpontjainak átlagolásával határozom meg.



*Ábra 25 ArUco kódok detektálása*

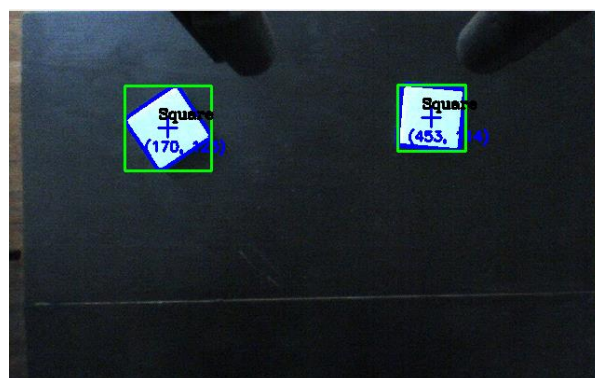
---

A kapott képen jól látható a koordináták az elfordulási szögek és az id-k is. Ehhez az ArUco detektáló algoritmus megírásában az alábbi linkek segítettek<sup>18,19</sup>.

Ezután a program egy másik függvényt hív meg, amely felelős a nem ArUco kóddal ellátott kockák felismeréséért. Ehhez a függvényhez szintén szürkeárnyalatossá alakítom a képet, majd Gauss-szűrőt alkalmazok a zajok csökkentése érdekében. Ezután használom a Canny él-detektáló algoritmust a kontúrok megtalálásához. A függvényem kiszámítja a kontúrok területét, majd egy közelítő poligonnal approximálja azokat. Ennek eredményeként simábbá válnak a görbék, és könnyebben meghatározhatók a geometriai alakzatok. Az approximálás során meghatározom a poligon sarkainak számát is. Ha egy négyzetet találunk, a cv2.boundingRect módszer segítségével meghatározom a négyzet bal felső sarkának x és y koordinátáit, valamint a szélességet és a magasságot. Ezekkel az értékekkel egyszerűen kiszámítható a négyzet középpontja.



Ábra 27 Élkeresés utáni kép



Ábra 26 A kockák detektálása

Ehhez az élkereső algoritmusok alapjául az alábbi git linken található kód segített<sup>20</sup>.

Ezek a folyamatok lehetővé teszik a program számára, hogy megbízhatóan átvigye és feldolgozza a robot kamera képét, valamint felismerje az ArUco markereket és más geometriai alakzatokat a képen. Ezáltal lehetővé válik a további feladatok végrehajtása, például a robot navigációja vagy a környezetének térképezése.

---

<sup>18</sup> <https://pyimagesearch.com/2020/12/21/detecting-aruco-markers-with-opencv-and-python/>

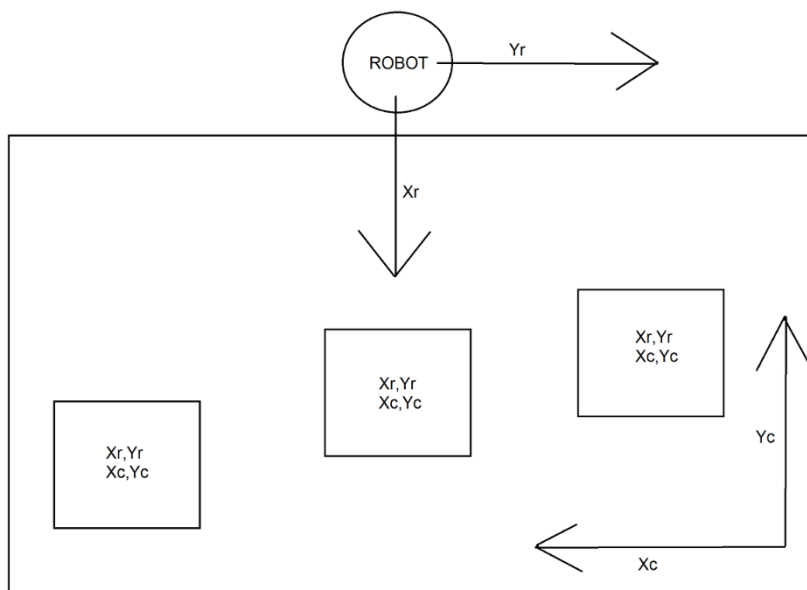
<sup>19</sup> [https://docs.opencv.org/4.x/d5/dae/tutorial\\_aruco\\_detection.html](https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html)

<sup>20</sup> <https://github.com/murtazahassan/Learn-OpenCV-in-3-hours/blob/master/chapter8.py>

### 6.3. Koordináta rendszerek és kalibrálás

A koordinátarendszerek egységesítése kulcsfontosságú volt a projekt szempontjából. Miután megvoltak a kockák helyzetei (koordinátái), ezeket össze kellett egyeztetni a robot koordinátarendszerével, vagyis át kellett helyezni a világkoordináta rendszerbe.

Amint látható a 28. ábrán a robot koordináta rendszere eltért a képek által alkotott koordináta rendszertől. Ezeket különböző matematikai képletekkel és a MATLAB segítségével egyesítettem.



Ábra 28 Robot és a kamera koordináta rendszere

A pontokat a térben vektorral lehet leírni. A koordinátarendszerben található pontot az egységvektorokkal írjuk le:

$$i = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, j = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, k = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

Mivel a kockák mérete egységes, és a kamerával készült képek mindig ugyanabból a magasságból készültek, a z koordinátákat egy konstans értékkel helyettesítettem. Megkellet határoznom azt az átalakító függvényt, ami a kamera koordinátából valós koordinátát alkot.

$$\begin{pmatrix} Xc \\ Yc \end{pmatrix} \rightarrow \begin{pmatrix} Xr \\ Yr \end{pmatrix}$$

A kamera koordinátáit valós (robot) koordinátarendszerbe kellett átalakítanom. Az „Xr” változó a robot x tengelyen található koordinátáját helyettesíti, míg az „Xc” a kamera x tengelyen található koordinátát jelenti. Ez a helyzet igaz az y tengelyre is. Az alábbi egyenletrendszerrel kívántam meghatározni azt az egyenest, amely átalakítja a koordinátákat:



$$\begin{cases} X_r = ax * X_c + bx \\ Y_r = ay * Y_c + by \end{cases}$$

Ezekkel az egyenletekkel kaptam meg az átalakításokat végző egyeneseket. Ehhez 4 változót kellett meghatároznom, amelyek segítségével könnyen átalakíthatom a kamera által adott x és y koordinátákat a robot x és y koordinátává.

Ebben az esetben a feladat megoldása lehetetlen volt, ezért több mérést kellett végeznem az értékek meghatározásához.

A kamera által alkotott x,y koordináták könnyen megtudtuk határozni a függvények segítségével, elemben azt, hogy hol helyezkednek el a kockák a robot koordináta rendszeréhez képest azt egyelőre nem tudtam. Ehhez jött nagy segítségemre a `get_left_endpoint_position` metódus, ami visszatérítette a vég berendezés koordinátáit. Ezután az elhelyezett kockáról csináltam egy képét, amin megkaptam a kamera szemszöget, majd a kocákhoz vittem a robot vég berendezését és lekertem annak a koordinátáit is. Ezzel a módszerrel megvolt a valós és a kamera által alkotott x,y változók.

Annak érdekében, hogy a rendszerem minél pontosabban működjön, több mérést is elvégeztem. A kalibrációs asztalom a 29. ábrán látható.



Ábra 29 Kalibráláshoz használt asztal

A kalibrációs folyamat egyszerűbbé és gyorsabbá tétele érdekében létrehoztam egy kódot, amely feliratkozik a végberendezés koordinátáit visszaadó témára, és amikor a kocka fölé vittem, és úgy gondoltam, hogy megfelelő helyen van, egy "p" betű lenyomásával elmentettem az értékeket egy fájlba. Hasonlóan jártam el a kamera kép koordinátaival is. A 30. ábrán látható kódrészletben az ArUco kódok koordinátáit elmentettem egy fájlba. Az első részben a szerverről fogadom a képet, amelyet stringként kapok, majd átalakítom OpenCV képformátumba. A kép után meghívom a 25. ábrán látható ArUco kód kereső függvényt, és miután megkaptam a koordinátákat, azokat az "adatKep.txt" fájlba szúrtam be a "write\_to\_file" függvény segítségével.

```
def write_to_file(data):
    global count
    global count2
    with open("adatKep.txt", "a") as file:

        x = round(data['x'], 5)
        y = round(data['y'], 5)
        file.write(f"{count}. x:{x} y:{y}\n")
        if count2%3 == 0:
            count += 1
        count2+=1

def get_image_data():
    cmd = {'left_arm':
           {'image' : 1}}
    socket2.send_json(cmd)
    msg = socket2.recv_string()
    encoded_string = msg
    decoded_bytes = base64.b64decode(encoded_string)

    image = cv2.imdecode(np.frombuffer(decoded_bytes, np.uint8), -1)
    markers = detect_aruco_markers(image)
    markers.sort(key=lambda marker: marker['id'], reverse=True)
    print(markers)
    for item in markers:
        x, y = item['coordinates']
        return{
            'x': x,
            'y': y
        }
```

*Ábra 30 Kód részlet az adatok fileba való mentéséről*

Miután kinyertem az adatokat, könnyedén meg tudtam írni a megoldó egyenletrendszer:

$$\begin{cases} Xr_1 \approx ax * Xc_1 + bx \\ Xr_2 \approx ax * Xc_2 + bx \\ Xr_3 \approx ax * Xc_3 + bx \\ Xr_4 \approx ax * Xc_4 + bx \\ Xr_5 \approx ax * Xc_5 + bx \\ \dots \end{cases}$$

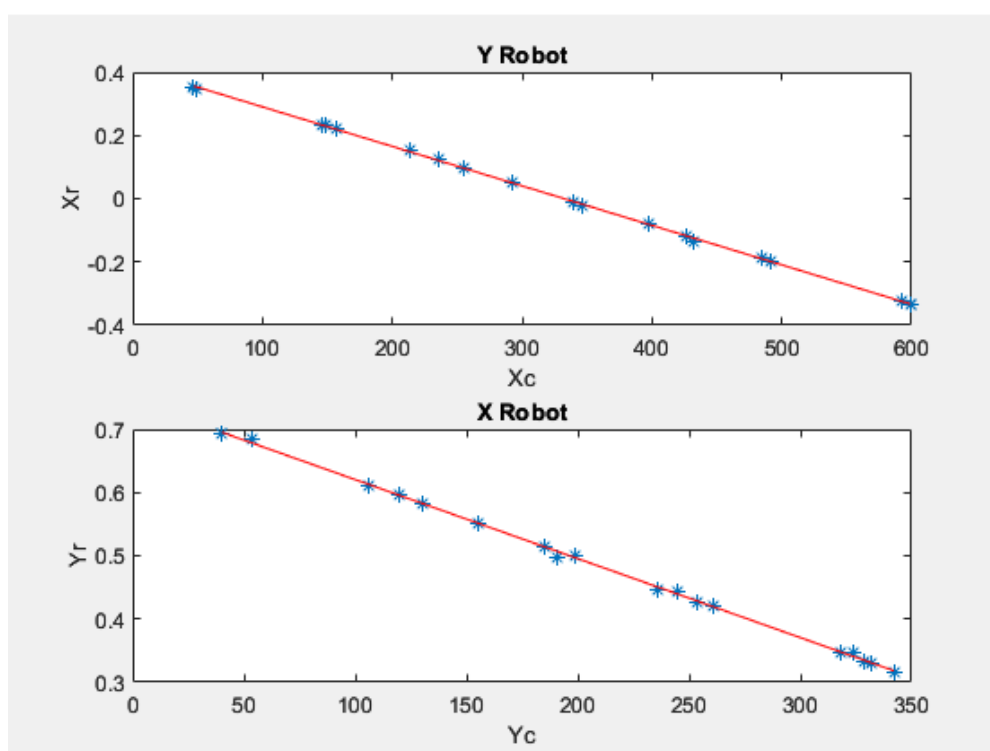


Az értékek behelyettesítése után a következőképpen néztek ki:

$$\begin{cases} -0.321703 \approx ax * 592 + bx \\ -0.333733 \approx ax * 599 + bx \\ -0.19946 \approx ax * 491 + bx \\ -0.187063 \approx ax * 485 + bx \\ -0.117557 \approx ax * 427 + bx \\ \dots \end{cases}$$

Ugyanígy jártam el az y tengely esetén is. Azért csak közelítő értékeket tudtam számolni, mert mindig voltak minimális eltérések a mért értékekkel kapcsolatban, ezért segítségül hívtam a MATLAB-ot, amellyel könnyen kiszámoltam az egyenes egyenletét, amely megoldja a koordináták közti eltérést.

A MATLAB-ban tároljuk a kinyert koordinátákat. Először az x koordináták között illesztünk egy elsőfokú polinomot (egyenest) a `polyfit` függvénnyel. Az egyenes együtthatóit egy külön változóban tároltam. Hasonlóan jártam el az y koordinátákkal is. Majd ezeket az illesztéseket vizualizáltam.



Ábra 31 Átalakító egyenlet ábrázolva

Ezek segítségével meghatároztam az említett „ax”, „bx”, „ay” és „by” konstansokat.

---

A robot szemszögéből látható x és y koordinátákat ismerve az inverz kinematika segítségével megtudtam mondani, hogy a robot milyen helyzetbe menjen. A fent említett kinematika megoldotta azt, hogy a robot csukloí milyen szögbe kell helyezze ahhoz, hogy elérjen a kijelölt pontra. A `move\_right\_arm\_to\_coordinate` metódussal feltudtam venni a kockákat, és az előzetes szögből, amit az ArUco kód adott meg, megtudtam határozni, hogy mennyit kell elfordulnia a kockának, hogy vízszintes legyen.

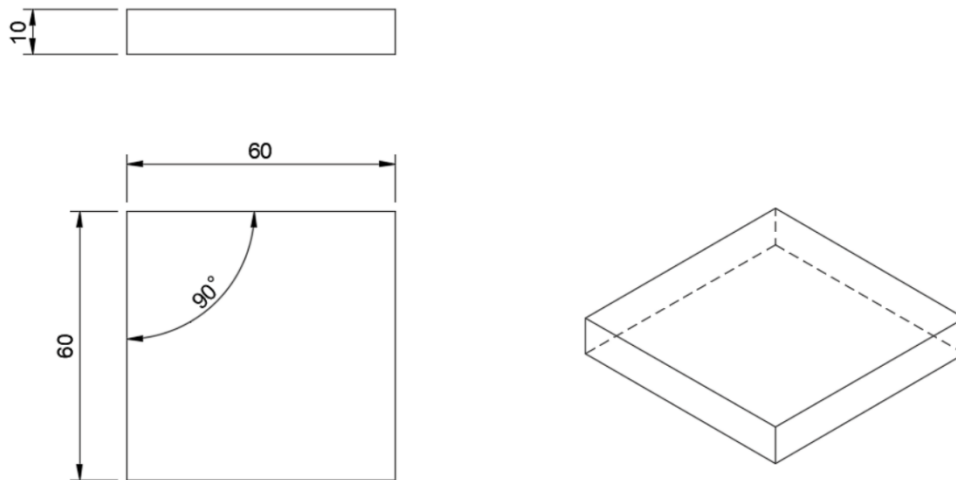
```
def move_right_arm_to_coordinate(self, x, y, z, ox, oy, oz, ow):
    cmd = {'right_arm':
           {'ball_coordinate':{'x':x,'y':y,'z':z,'ox':ox,'oy':oy,'oz':oz,'ow':ow}}
    }
    self.send_message(cmd)
    msg = self.receive_message()
    print(msg)
```

*Ábra 32 A végberendezés meghatározott koordinátához való küldése.*

Ezek után már csak a végpontba kellett őket helyezni. A végpontokat előre meghatározott rögzített koordináták alapján változtattam meg, csak az elhelyezési magasságon variáltam minden iterációban, hogy a kockákat egymásra rakja. Ezeket a folyamatokat számtalan finomhangolás követett a konstans értékein belül, hogy a robot megfelelő pontossággal mozogjon. Ezeket a változtatásokat a kódom módosításával és azok utasítások követésével a robot reakciójából vontam le.

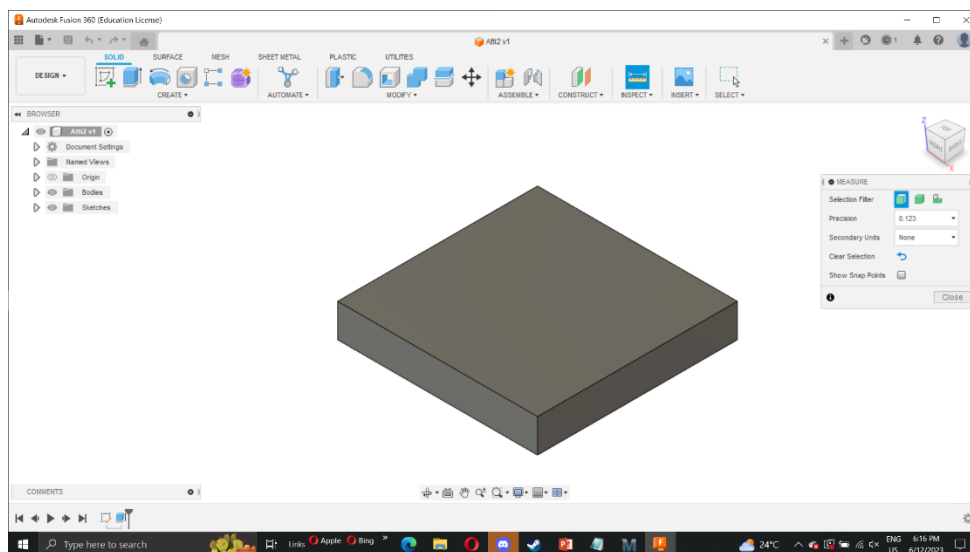
## 6.4. A kísérlet során felhasznált kockák

A háromdimenziós nyomtatás segítségével gyorsan és egyszerűen valós tárgyakat lehet létrehozni digitális modellekből. Használhatunk gyantát vagy műanyagot a rétegezés során, hogy bármilyen alakú háromdimenziós tárgyat elkészítsünk. A projekt során a 3D nyomtatást választottam a kockákhoz, mert ez a módszer gyorsabb és könnyebben megvalósítható volt számomra.



*Ábra 33 A 3D-s kocka tervdraja*

A 3D nyomtatás működése 3 fő részből áll: elsősorban meg kell tervezni a modellt (33. ábra) majd egy számítógépes program, segítségével megalkotjuk a 3 dimenziós rajzot. A kész 3D modell megtekinthető a 34. ábrán. Ezután egy szeletelő programmal fel kell osztani a modellt vékony rétegre, és meg kell adni a nyomtatási paramétereket. Végül a nyomtató a szeletelő program utasításai alapján elkészíti a tárgyat rétegről rétegre. A kocka megtervezésében és szeletelésében nagy segítséget adott az Autodesk Fusion 360 program. Ez egy teljeskörű tervező szoftver, amelyben lehetőségünk van 3D modellek létrehozására, tervezésére és vizualizálására.

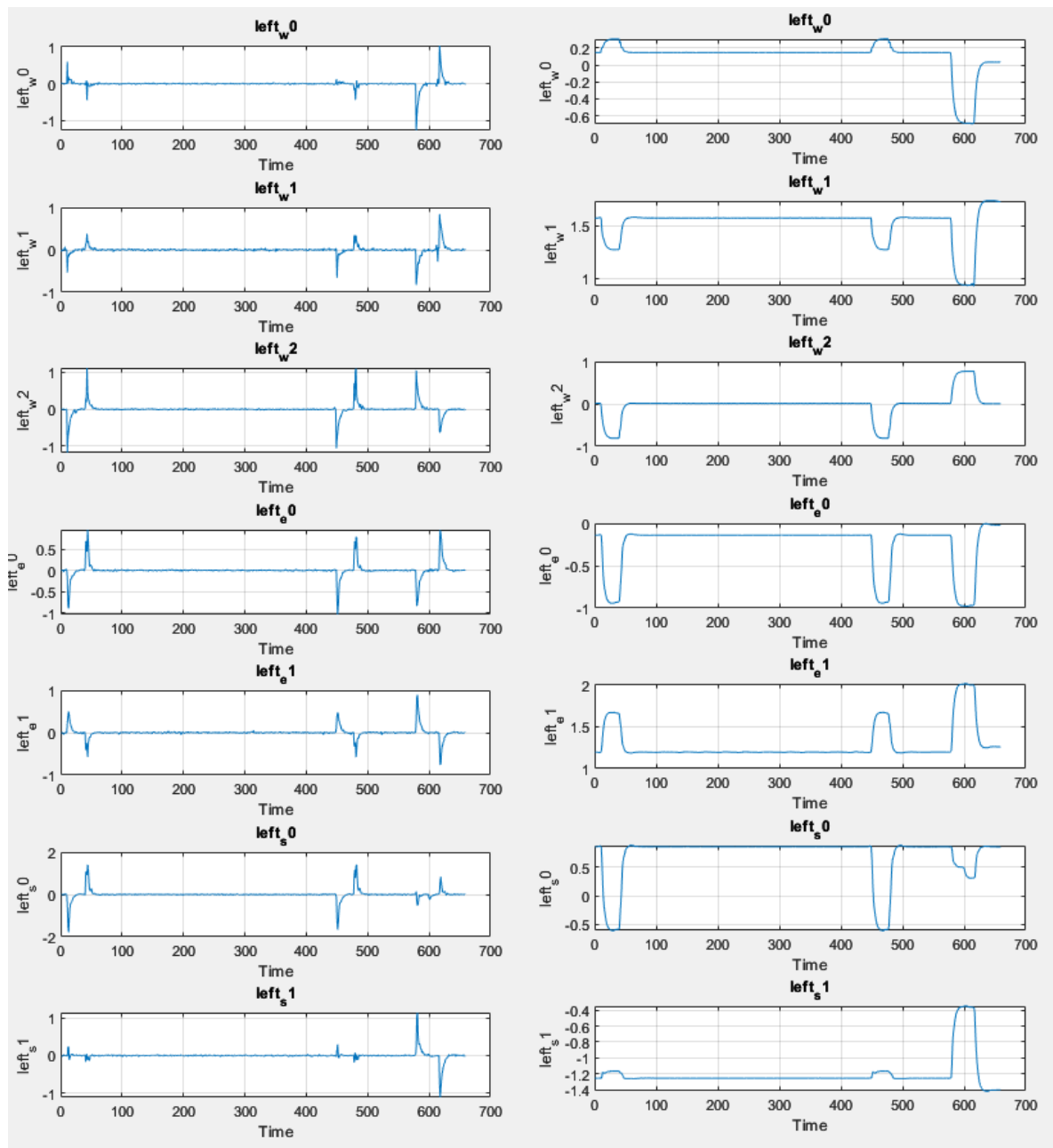


*Ábra 34 A 3D kocka tervezése Autodeskben*

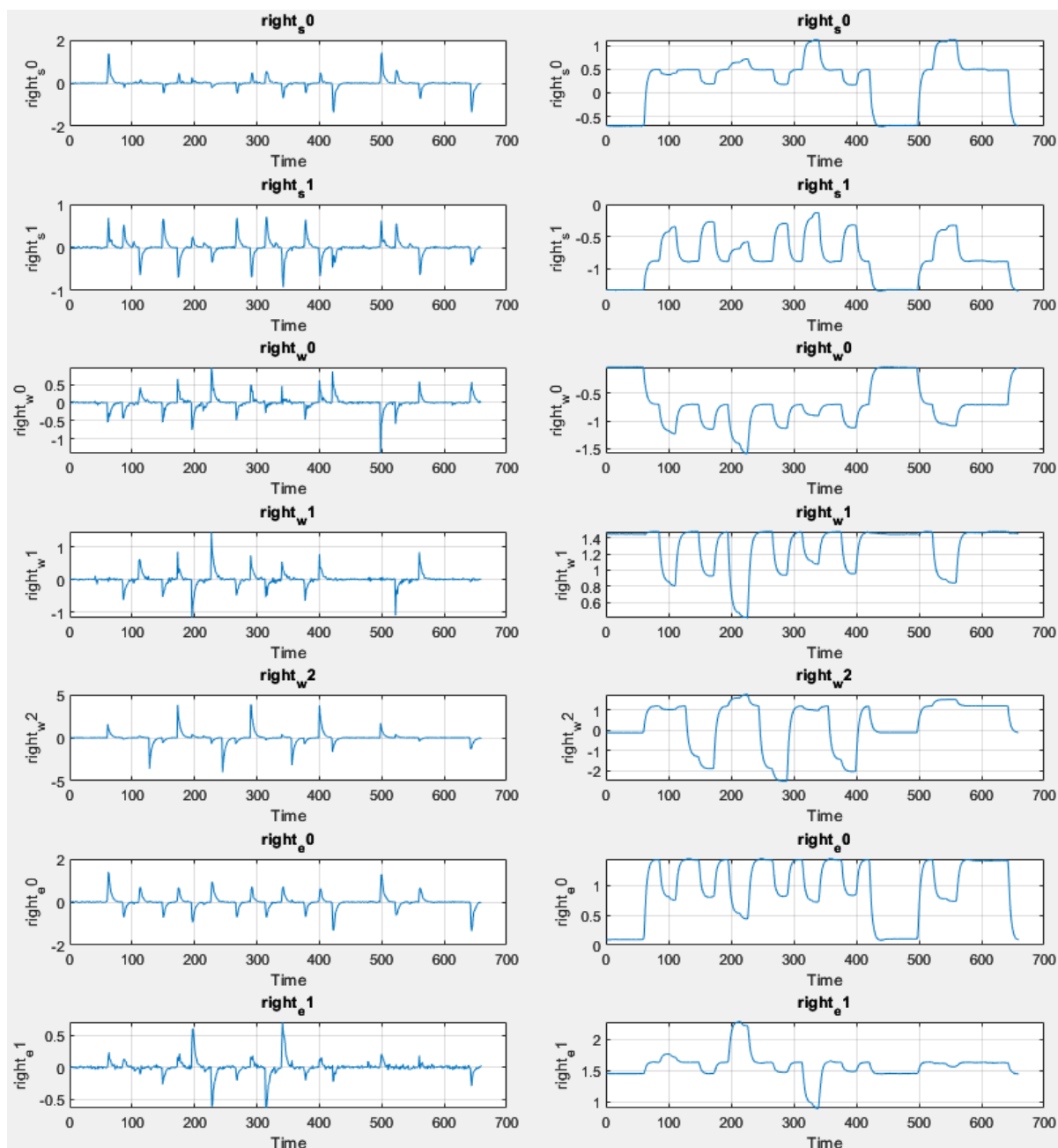


Bal oldali képen a robotkar sebessége és szöge figyelhető meg. A 'velocities' kulcs alatt találhatók az egyes motorok sebességei, ezek numerikus értékek, amik az aktuális sebességet jelölik. Az 'angles' kulcs alatt az egyes motorok szögei találhatók, ezek radiánban vannak kifejezve. A jobb oldali képen pedig a végberendezés pozícióját és orientációját találhatók. A 'position' kulcs alatt megkapjuk az x, y, z koordinátákat, amik meghatározzák a végberendezés pontos pozícióját a 3 dimenziós térben. Az 'orientation' kulcshoz tartozó x, y, z, w írják le az aktuális orientációt.

A 36. és a 37. ábrán látható a motrok külön-külön a munkafolyamatuk során kinyert szögek és sebességek.



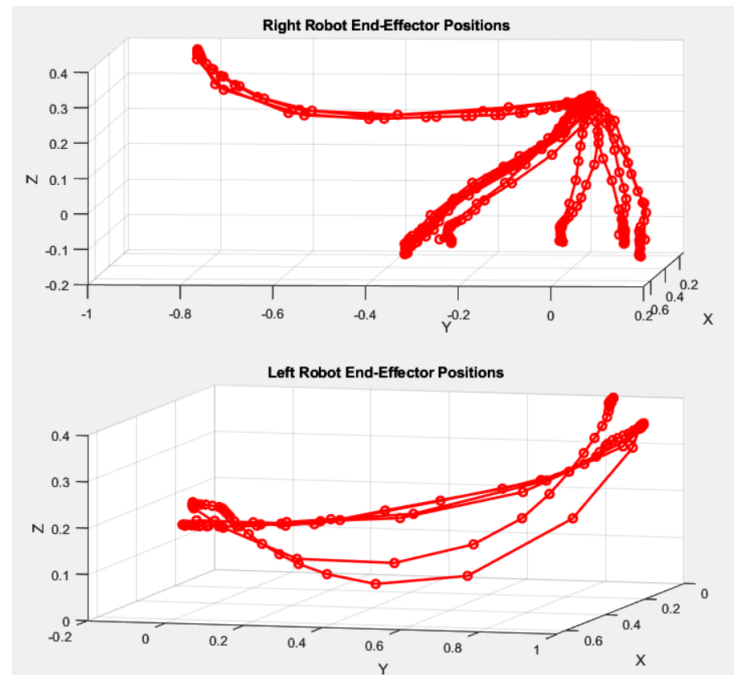
Ábra 36 A bal karról mért adatok ábrázolva



Ábra 37 A jobb karról mért adatok ábrázolva

A fenti képen jól láthatók a bal és a jobb kar sebességei, valamint a szögek értékei, amelyek külön-külön motorokra vannak felosztva. Ezek az értékek segítségével jól látható, hogy a munkafolyamat milyen fázisban van. A jobb kar esetében több kilengést lehet megfigyelni, mivel ez a kar végzi a kockák rendezését. Ebből látható, hogy hogyan változnak a motorok sebességei a különböző lépések során. A 37. ábrán látható a "rightw2" szög, amelyen jól látszik, hogy a kockák fogása hogyan történt. Ez a motor felelős a végberendezés forgatásáért. Megfigyelhető, hogy mindig ugyanabba az irányba történik a 3 forgatás, mivel a kockákat mindig egységes irányba forgatjuk, hogy párhuzamosak legyenek a munkaasztallal. A 36. ábrán pedig a bal kar figyelhető meg, ahol

3 kilengést láthatunk. Ez érthető is, mert ez a kar háromszor mozdul el a munkafolyamat során. Az első kilengés, amikor fényképet készít, középre megy a munkaterületen. A második kilengés szintén a fotókészítést reprezentálja, de már nem az ArUco kódokkal ellátott kockákat keresi. Az utolsó kilengés arra szolgál, hogy amikor a bal kar odamegy a jobb kar mellé, elvegye tőle a selejtes darabot, amit később a selejtegyűjtő helyre visz.



Ábra 38 A végberendezés pozíciójának ábrázolása a munkafolyamat során

A 38. ábrán a végberendezés mozgásterét mutatom be a munkafolyamat alatt. A képen látható, hogy a jobb kar több területet fedett le, mint a bal kar, mivel a jobb kar volt a felelős a kockák felvételéért és átadásáért. A felvételi folyamat során a jobb kar a kockák helyzetének megfelelően mozgott a munkaterületen. A kép felső részén ezek a mozgások jól követhetők. A bal kar feladata a kockák fényképezése volt, amelyhez a bal kar egy előre meghatározott pozícióba kellett mozognia. A kép alsó részén látható, hogy a bal kar hogyan indult ki az eredeti helyzetéből, hogyan érte el a fényképezési pozíciót, és hogyan vette át a kockát a jobb kartól.

---

## 8. Összefoglaló

### 8.1. Következtetések

A dolgozatom központi témája egy rendszer megvalósítása, amely képes különböző kockák szétválogatására egy munkaterületen. Az általam tervezett autonóm rendszer két fő részből épül fel, melyek közös működése teszi lehetővé a kockaválogatást. Az első rész a felhasználói rész, amely a szükséges számításokat és feldolgozásokat végzi el a munkafolyamat során. A második rész pedig a szerver, amely felelős a robot irányításáért.

Az általam kifejlesztett autonóm rendszer eredményeként sikeresen megvalósítottam a kockák szétválogatását a munkaterületen. A rendszerem meghatározta a kockák helyzetét, és az ArUco kódok használatával precízen meghatározta a kockák elfordulási szögét is. Emellett a rendszerem képes volt azoknak a kockáknak a felismerésére is, amelyek nem rendelkeztek ArUco kóddal, és ezeket az előre meghatározott selejtezőhelyre irányította. A kockák felismeréséhez az OpenCV képfeldolgozási könyvtárat használtam. A rendszerem pontosan mozgatta a kockákat a célpontjuk felé, és a megfelelő időben kapcsolta be és ki a megfogásukért felelős vég berendezésen található markolókat. A felhasználói élmény fokozása érdekében kialakítottam egy kezelőfelületet, amelyen keresztül a munkafolyamat indítható és leállítható. A leállítás után a robot visszatért az eredeti helyzetébe, és készen állt az újraindításra.

A kockák helyzetének meghatározásához a roboton található beépített kamerát használtam. A bal oldali karjába integrált kamera segítségével minden munkafolyamat előtt képet készítettem, majd a felhasználói rész segítségével képfeldolgozó algoritmusokat alkalmaztam a képek elemzésére. A feldolgozás után a felhasználói rész parancsokat küldött a robotnak a karok mozgatásához. A robot és a felhasználói komponens közötti kommunikációt egy köztes szerver biztosította, amely átalakította és értelmezhető formátumba rendezte a beérkező parancsokat a robot számára. Az adatok gyors és megbízható átvitele érdekében a ZMQ kommunikációs keretrendszert használtam, amely hatékonyan biztosította a két rész közötti aszinkron adatcserét.

Összefoglalva, általam tervezett munkafolyamat sikeresen megvalósítható a Baxter robot segítségével. Az általam fejlesztett rendszer működőképes. Fontos megemlíteni, hogy a Baxter robot kiválóan alkalmas autonóm feladatok ellátására, azonban a pontosságának hiánya miatt némi nehézséget jelenthet a kisebb tárgyak mozgatása. Emellett a karba épített kamera felbontása sem tökéletes a képfeldolgozó algoritmusok számára, hiszen érzékeny a külső fényviszonyokra, ami



---

homályos képek készítéséhez vezethet. A robotkarok vezérlésére alkalmazott inverz kinematika tökéletesen működött, mindig a megfelelő koordinátákra pozicionálva a végberendezést.

## 8.2. Fejlesztési lehetőségek

A továbbfejlesztési lehetőségek során számos újítást tervezek bevezetni, amelyek még hatékonyabbá és sokoldalúbbá teszik a rendszert. A továbbfejlesztési lehetőségek közül az egyik legfontosabb a mozgásban lévő tárgyak kiválasztásának megvalósítása lenne, amelynek segítségével a robot képes lenne szétválogatni a futószalagon érkező elemeket. Ezzel a funkcióval a baxter robotot egy ipari környezetben is hasznosíthatnánk, ahol a gyártósoron érkező termékeket kellene minősíteni és selejtezni. A robot ebben az esetben is megtartaná a cobot jellegét, azaz együttműködne az emberi munkatársakkal. A robot így képes lenne olyan feladatokat is elvégezni mint például kiválogatni a szalagon érkező termékeket, amik rendben vannak azokat tovább engedi, amelyikek selejtesnek nyilvánítódnak, azokat külön helyre válogatná.

Egy másik fejlesztési lehetőség a kockák pontosabb felismerése érdekében a külső kamera használata lenne a munkaterület felderítésére. Ez lehetővé tenné, hogy részletesebb és pontosabb képeket készítsünk, amelyek segítségével az átalakító algoritmusok precízebben tudnák elvégezni a feladatukat. Ennek eredményeként a robotkar pontosabban pozicionálhatná a tárgyakat. Ezenkívül a külső kamera segítségével elkerülhető lenne a homályos képek készítése, amelyek miatt nem tudja azonosítani a kóddal ellátott tárgyakat.

A harmadik célkitűzött fejlesztési lehetőség az felhasználói felület bővítése több funkcióval. Például lehetőséget biztosítanánk a felhasználónak a kockák rendezésének sorrendjének beállítására, vagy akár folyamatosan kivetíthetnénk a kamera képét, amelynek segítségével a felhasználó valós időben követhetné a rendszer működését.

Ezen továbbfejlesztések révén a rendszer még hatékonyabbá és sokoldalúbbá válna, lehetővé téve számos új feladat elvégzését.

---

## 9. Irodalomjegyzék

- [1] Zhang, W., Zhang, C., Li, C., & Zhang, H. (2020). Object color recognition and sorting robot based on OpenCV and machine vision. 2020 IEEE 11th International Conference on Mechanical and Intelligent Manufacturing Technologies (ICMIMT). doi:10.1109/icmimt49010.2020.9041220
- [2] Pereira, V., Fernandes, V. A., & Sequeira, J. (2014). Low cost object sorting robotic arm using Raspberry Pi. 2014 IEEE Global Humanitarian Technology Conference - South Asia Satellite (GHTC-SAS). doi:10.1109/ghhc-sas.2014.6967550
- [3] Gubbi, J., Sandeep, N. K., Reddy, K. P. K., & Balamuralidhar, P. (2017). Robust markers for visual navigation using Reed-Solomon codes. 2017 Fifteenth IAPR International Conference on Machine Vision Applications (MVA). doi:10.23919/mva.2017.7986900
- [4] Kumar, R., Lal, S., Kumar, S., & Chand, P. (2014). Object detection and recognition for a pick and place Robot. Asia-Pacific World Congress on Computer Science and Engineering. doi:10.1109/apwccse.2014.7053853
- [5] Pop, E., Leba, M., Sirb, V. C., & Badea, A. G. (2008). Modeling, simulation and control for an underground roof support as two robotic arms. 2008 IEEE International Conference on Automation, Quality and Testing, Robotics. doi:10.1109/aqtr.2008.4588807
- [6] Cremer, S., Mastromoro, L., & Popa, D. O. (2016). On the performance of the Baxter research robot. 2016 IEEE International Symposium on Assembly and Manufacturing (ISAM). doi:10.1109/isam.2016.7750722
- [7] 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.
- [8] <https://sdk.rethinkrobotics.com/wiki/Arm>
- [9] <https://www.diva-portal.org/smash/get/diva2:1333640/FULLTEXT01.pdf>
- [10] Williams, R. L. "Baxter Humanoid Robot Kinematics© 2017 Dr. Bob Productions Robert L. Williams II Ph. D. williar4@ ohio. edu Mechanical Engineering Ohio University April 2017." URL <https://www.ohio.edu/mechanical-faculty/williams/html/PDF/BaxterKinematics.pdf> (2017).
- [11] [https://sceweb.sce.uhcl.edu/harman/CENG5931Baxter2015/Guides/BAXTER\\_Introduction\\_2\\_08\\_2016.pdf](https://sceweb.sce.uhcl.edu/harman/CENG5931Baxter2015/Guides/BAXTER_Introduction_2_08_2016.pdf)
- [12] <http://collabrobots.com/baxter-robot-accessories/>

- 
- [13] F. J. Romero-Ramirez, R. Muñoz-Salinas, és R. Medina-Carnicer, „Speeded up detection of squared fiducial markers”, *Image Vis. Comput.*, köt. 76, o. 38–47, aug. 2018, doi: 10.1016/j.imavis.2018.05.004.
- [14] D. Jurado-Rodriguez, R. Munoz-Salinas, S. Garrido-Jurado, és R. Medina-Carnicer, „Design, Detection, and Tracking of Customized Fiducial Markers”, *IEEE Access*, köt. 9, o. 140066–140078, 2021, doi: 10.1109/ACCESS.2021.3118049.

---

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

Vizat decan

Conf. dr. ing. Domokos József



Vizat director departament

Ș.l. dr. ing. Szabó László Zsolt

