

Pannon Egyetem

Műszaki Informatikai Kar

Villamosmérnöki és Információs Rendszerek Tanszék



Braghini Benjamin Matthew

2022

Témavezető: Rádli Richárd, PhD hallgató

**Innovatív Ipar 4.0 megoldások
robot környezetekben**

Kézirat lezárásának dátuma: 2022.11.14.

Innovatív Ipar 4.0 megoldások robot környezetekben

Készítette: Braghini Benjamin Matthew

Programtervező informatikus, BSc

Műszaki Informatikai Kar,

Villamosmérnöki és Információs Rendszerek Tanszék

Témavezető: Rádli Richárd Bence

Napjainkban, mikor a negyedik ipari forradalom egyre nagyobb teret hódít magának, előtérbe kerülnek az olyan ágazatok, mint a gépi látás és a robotika. E két területet kombinálva, a mérnökök képesek olyan autonóm robot rendszerek fejlesztésére, amire korábban nem volt lehetőségük. Az ilyen rendszerek konstruálásához viszont fontos, hogy a robotok tisztában legyenek az őket körülvevő környezettel. A dolgozat célja a jelenleg Ipar 4.0-ban használt módszerek feltérképezése, majd a megszerzett tudás birtokában innovatív megoldások fejlesztése robot környezetben.

Egy robotrendszer érdemi használatba vétele előtt nagyon fontos, hogy mind a robotkar, valamint a robotkaron lehelyezkedő megfogó és kamera kalibrálva legyenek egymáshoz képest. Erre a klasszikus megoldás, ha elhelyezünk egy sakktáblát a munkaterületen, és előre felvett pontok alapján végig haladva képeket készítünk. A pályamunkám keretein belül ezt szeretném kiváltani egy teljesen dinamikus megoldással. Ennek a megközelítésnek a célja az lenne, hogy a sakktábla akárhol lehet a munkaterületen, a rendszer megkeresné a sakktáblát, és bekalibrálná magát.

További célkitűzés, már létező objektum detektáló és pózbecslő algoritmusok alkalmazása és fúziója. Ezekkel az algoritmusokkal két célt igyekszek megvalósítani: először egy memória modul, majd egy alaplap detektálását és pózbecslését szeretném elvégezni. Végezetül a memória modult megfogva, azt egy robotkarra szerelt megfogóval kiemelni a tálcájából, és behelyezni az alaplap egy tetszőleges memória foglalatába.

Tartalomjegyzék

Jelölésjegyzék	7
1. Bevezetés	8
1.1. A dolgozat célja	9
1.2. A dolgozat felépítése	10
2. Felhasznált technológiák	11
2.1. Hardver oldal	11
2.1.1. Universal Robots 5	11
2.1.2. OnRobot RG2 megfogó	11
2.1.3. Basler ace aca2440-35uc	12
2.2. Szoftver oldal	14
2.2.1. Numpy	14
2.2.2. URX	14
2.2.3. Urllib	14
2.2.4. Socket	15
2.2.5. OpenCV	15
3. Létező megvalósítások	16
3.1. Robot manipulátor és több 3D kamerarendszer automatikus ka- librációja	16
3.2. Teljesen automatikus hand-eye kalibrációs rendszer	17
3.3. Önadaptív rendszer robot összeszerelési műveletekhez	19
4. Ipar 4.0	20
4.1. A robotika és az automatizálás szerepe az Ipar 4.0-ban	21
4.2. Robotika alkalmazása az Ipar 4.0-ban	23

5. Kamera és robot kalibrálása, pózbecslés	25
5.1. Perspektív kamera modell	25
5.2. Perspektív kamera modellezése	27
5.3. Kamera mátrix	28
5.4. Lencse torzítások	29
5.5. Kamera kalibráció	30
5.6. Hand-eye kalibráció	31
5.7. Robot kinematika	33
5.8. Pózbecslés	33
 6. A feladat megvalósítása	 34
6.1. Mozgatások	34
6.2. Palettázás	35
6.2.1. Felmerült problémák	37
6.3. Dinamikus megközelítés	38
6.4. Megfogó	41
6.5. Kamera és hand-eye kalibráció	41
6.6. Alaplap detekció	43
6.7. Memória modul detekció	45
6.8. Összeszerelés	45
 7. Tesztek és mérési eredmények	 48
7.1. Robot pontossági teszt	48
7.2. Kalibráció pontosságának a tesztelése	48
7.3. Detektálás pontosságának tesztelése	49
7.4. Memória megfogás és behelyezés	51
 8. Összefoglalás	 52
 9. Függelék	 54
9.1. Robot pontossági teszt	54
9.2. Kalibráció pontosságának a tesztelése	62
9.3. Detektálás pontosságának tesztelése	63
9.4. Memória megfogás és behelyezés	65

Köszönetnyilvánítás

Köszönjük az OTKA K 135729 Geometriai tulajdonságok kinyerése lényegi vizuális jellemzőkből és az NVIDIA GPU Grant program támogatását.

Jelölésjegyzék

AVG Automated Guided Vehicle

FoV Field of View

GPU Graphical Processing Unit

HE Hand-Eye

MP Megapixel

MSE Mean Squared Error

NYAK Nyomtatott áramkör

PnP Perspective-n-Point

RAM Random Access Memory

ROS Robot Operating System

TCP Tool Central Point

UR Universal Robot

1. fejezet

Bevezetés

Napjainkban, mikor a negyedik ipari forradalom szemtanúi vagyunk, az olyan technológiai területek, mint a gépi látás és a robotika, egyre komolyabb jelentőséget élveznek. E két ágat kombinálva, a mérnökök képesek autonóm robot rendszereket megalkotni. Az ilyen rendszerek konstruálásához viszont fontos, hogy a robotok pontosan ismerjék a környezetüket.

Az egyik legalapvetőbb technológia az Ipar 4.0-ban az úgynevezett M2M, vagyis a gép-gép közötti kapcsolat: annak érdekében, hogy a gépek hatékonyan átvegyék a bonyolultabb folyamatok irányítását, szükséges, hogy megtanítsuk őket arra, hogy emberi beavatkozás nélkül kommunikáljanak egymással. Például, a gyártósoron dolgozó robotoknak képesnek kell lenniük arra, hogy önállóan kiszolgálják egymást a szükséges alkatrészekkel, vagy megszakítsanak egy folyamatot, ha veszélyt érzékelnek.

Mint már említettem, ahhoz, hogy a robotok biztonságosan tudjanak az emberek mellett dolgozni, fontos, hogy ismerniük kell a környezetüket. Ennek egyik módja, hogy tisztában vannak a körülöttük lévő tárgyak 3D helyzetével. Ezt hat darab számmal tudjuk leírni, ahol három szám a tárgy translációját (eltolását), és további három szám a tárgy rotációját (elforgatását) határozza meg. Ennek fontossága abban rejlik, hogy e póz ismeretében az olyan feladatok, mint például a pick-and-place, vagy átadás-átvétel is megoldhatók.

Mindezen műveletek előtt kritikus fontosságú az adott robot-rendszer kalibrálása, mind a kamera, mind a robotkar oldala felől.

1.1. A dolgozat célja

Számos gyakorlati alkalmazásban az ipari robotok még mindig "vakon", előre programozott bejárési utakkal dolgoznak. Ez azt eredményezi, hogy a robotok és az emberek munkaterülete szigorúan fel van osztva a balesetek elkerülése érdekében, amelyek sajnos néha még mindig előfordulnak. Továbbá a dinamikus környezetben való munkavégzés az azonos munkaterületen osztozó más gépekkel való közvetlen kapcsolat nélkül nehézségekbe ütközhet. Ilyen esetekben gyakoribbak az ütközésfelismerő rendszerek, amelyek azonban nem mindig működnek a várt módon, mint az ütközésmegelőző módszerek. A környezetükkel tisztában lévő robotok [5, 14] azonban egyre elterjedtebbek, mind a kutatásban kifejlesztett, mind maguk a robotgyártók, pl. a Rethink által készített Baxter Robotics [4] robotok.

A dolgozatom első célkitűzése egy olyan algoritmus kifejlesztése és implementálása, mely dinamikusan képes bekalibrálni egy robotkart, amin egy ipari kamera, valamint egy megfogó helyezkedik el.

Az ipari robotokat ma már olyan feladatok elvégzésére alkalmazzák, mint az összeszerelés vagy több robot együttműködésével végzett feladatokra, például hegesztésre. A feladat összetettségének változása, a folyamatidő és a gyakori átállások azonban megkövetelik, hogy egy robot rugalmasan alkalmazkodjon a különböző környezetekhez, az új termékekhez és a feladat elvégzéséhez. A feladatok összetettségének növekedésével a robotok programozása is egyre fáradságosabbá válik, mivel ezek a feladatok egyre bonyolultabbak. Így meglehetősen nagyobb szükség van arra, hogy a robotok rugalmasak legyenek és alkalmazkodjanak az új körülményekhez, hogy egy adott feladatot a kívánt pontossággal tudjanak végrehajtani. Az ipari környezetben a robotok jellemzően rögzítettek, fix helyen helyezkednek el, és a munkafelszerelést a robot munkaterületének megfelelően kell beállítani. Ez segíti a robotot a kívánt pontosság elérésében, de egyben nagyban akadályozza a munkafelszerelés vagy a kívánt feladat bármilyen megváltoztatását.

A dolgozat második célja, már létező objektum detektáló és pózbecslő algoritmusok alkalmazása és fúziója. A cél egy memória modul tálcában való detektálása, pózbecslése, végül megfogása a robotkarra szerelt megfogóval. Ezt követő feladat pedig egy, a munkaterületen lévő alaplap detekciója, pózbecs-

lése, majd az előzőleg megfogott memória modul az alaplap egyik memória foglalatába helyezése. Utolsó cél pedig, hogy az elkészült algoritmusokat alaposan, különböző teszteseteken keresztül teszteljem, az eredményeket pedig dokumentáljam.

1.2. A dolgozat felépítése

A dolgozat felépítése a következőképpen néz ki: a 2. fejezetben az általam felhasznált technológiákról ejtek pár szót, a 3. fejezetben már létező megvalósításokat járok körbe és mutatok be. A 4. fejezetben az Ipar 4.0 történetét dolgozom fel, míg a 5. fejezetben a kamera és robot kalibrálás elméleti hátterét mutatom be. Ezt követően a 6. és 7. fejezetekben a megvalósított programok és algoritmusok implementálása és tesztelése kerül bemutatásra. Végezetül a 8. fejezetben foglalom össze a dolgozat eredményeit. Mérési eredmények a 9. fejezetben kaptak helyet.

2. fejezet

Felhasznált technológiák

Ebben a fejezetben az általam használt eszközöket fogom bemutatni mind hardver, mind szoftver oldalról.

2.1. Hardver oldal

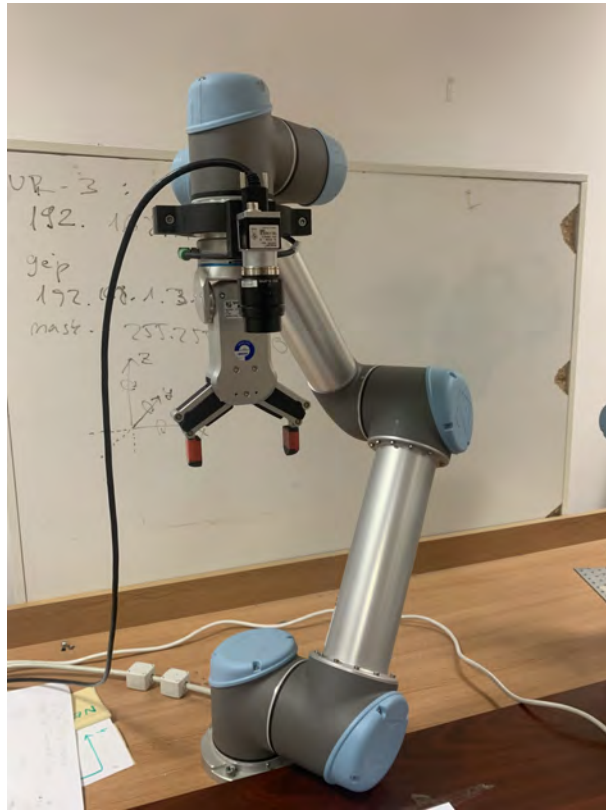
Hardver oldalon egy robotkart, egy kamerát, valamint egy megfogót használtam.

2.1.1. Universal Robots 5

A dolgozatom keretein belül egy középhaladó robotkaron volt szerencsém dolgozni, az Universal Robots 5-ös modelljén (UR5) aminek ugyan vannak hátrányai, de a feladatom elvégzésére megfelelő volt. Ez a robotkar egy úgynevezett kollaboratív robotkar, ami azt jelenti, hogy figyel a környezetére, hogy ne tegyen kárt se magában se pedig a környezetében lévő tárgyakban vagy dolgozóknak. A robotkar legfontosabb adatai a 2.1-es táblázatban találhatók.

2.1.2. OnRobot RG2 megfogó

Egy másik eszköz, amit használtam a feladatom megvalósításához, az OnRobot RG2-es megfogója volt. Ez egy professzionális megfogó, amit az iparban gyakran használnak kisebb tárgyak megfogására. A megfogó főbb paraméterei a 2.2-es táblázatban találhatók.



2.1. ábra. Universal Robots 5 robotkar és OnRobot RG2 megfogó, valamint a Basler kamera

2.1. táblázat. Universal Robots 5 robotkar adatai [16]

Tömeg	18.4 kg
Teherbírás	5 kg
Tartás megismétlése	± 0.1 mm
Foglalat átmérője	149 mm
Szabadági fokai	6
Kommunikáció	TCP/IP 1000 Mbit: IEEE 802.3u
Programozható	PolyScope UI
Használható környezeti hőmérséklet	0-50 °C
A kar max. sebessége	1 m/s
A csukló max. sebessége	180°/s

2.1.3. Basler ace aca2440-35uc

A feladat megvalósításához szükség volt egy kamerára, illetve egy hozzá tartozó objektívre is. Én az iparban is gyakran használt kis méretű, de nagy tel-

2.2. táblázat. OnRobot RG 2 megfogó adatai [12]

Terhelhetőség	2-5 kg
Befogási tartomány	110 mm
Ismétlési pontosság	0.1 mm
Fogóerő	Min: 3 N Max: 40 N
Megfogási sebesség	Min: 38 mm/s Max: 127 mm/s
Méret	213x149x36 mm
Súly	0.78 kg

tesztelési kamerát választottam, a Basler ace acA2440-35uc kamerát és egy Kowa LM12JC5M2 objektívet. Ezekről az eszközökről a legfontosabb adatok az alábbi táblázatokban láthatók:

2.3. táblázat. acA2440-35uc kamera adatai: [2]

Felbontás	2448 × 2048
Kép ráta	35 fps
Mono/színes	Színes
Bajonett	C-Mount
Tömeg	<80g
Ház hőmérséklete üzemeltetés közben	0-50 °C
Teljesítmény felvétel	2.7 W @ 5 VDC

2.4. táblázat. Kowa LM12JC5M2 objektív adatai [8]

Gyújtótávolság	12.5mm
Min. fókusztáv.	100mm
Bajonett	C-Mount
Üzemeltetési hőmérséklet	-10~+50°C
Torzítás	-0.06%
Tömeg	130g

2.2. Szoftver oldal

A robot mozgatójához, illetve a projekt majdnem teljes részéhez a Python programozási nyelvet használtam. A Python programozási nyelv, és azon belül is a Python 3 az egyik legelterjedtebb programozási nyelv a közhasználatban, illetve az iparban egyaránt. Sokan az egyszerűsége miatt szeretik, sokan a változatos könyvtárai miatt. Az én esetemben a választás azért erre a nyelvre esett, mert már egy létező keretrendszerhez kellett programokat írnom, amit Python nyelven implementáltak.

A kódszerkesztő amit használtam a Microsoft által fejlesztett Visual Studio Code volt. Ez gyakorlatilag a leegyszerűsített open source változata a Visual Studionak. Azért ezt a kódszerkesztőt választottam, mivel egyszerű a használata és széles körben alkalmazott.

Ezen felül, fontosnak érezném, hogy megemlítssem pár szóban a felhasznált Python könyvtárakat is, melyek összefoglalva a következők lennének:

2.2.1. Numpy

A Numpy a Python tudományos számítástechnikai programkönyvtára. Nagy teljesítményű többdimenziós tömbobjektumokat és eszközöket biztosít az ezekkel a tömbökkel való munkavégzéshez. Sokoldalúsága és gyorsasága miatt (C programozási nyelven lett implementálva) gyakran használnak az ipar több területén is.

2.2.2. URX

Az Universal Robots robotkarok mozgatójára és irányítására készült open source Python könyvtár. Használati köre az egyszerű megfogóktól a szenzoros alkalmazásokig van kiterjesztve. Hátránya, hogy nem hivatalos könyvtár.

2.2.3. Urllib

Legfőképpen web url-ek és hivatkozások megnyitásához/futtatásához használják. Az én esetemben a program és ezáltal, a robot, illetve a szerver közti kommunikációba játszott szerepet.

2.2.4. Socket

A socket programozás a hálózat két csomópontjának összekapcsolásának módja, hogy kommunikáljanak egymással. Az egyik socket (csomópont) egy adott IP-porton figyel, míg a másik socket eléri a másikat, hogy kapcsolatot hozzon létre. A kiszolgáló alkotja a figyelő socketet, miközben a kliens eléri a szerveret.

Erre a könyvtárra főleg a megfogó és a robotkar közti kompatibilitási problémák miatt volt szükség. Ennek segítségével kitudtam olvasni egy script file-ból az adatokat, amiket a robotkaron keresztül át tudtam küldeni a megfogónak.

2.2.5. OpenCV

Eredetileg az Intel által kifejlesztett szoftverkönyvtár ami a valós idejű gépi látást és egyéb kép manipulációval kapcsolatos folyamatok egyszerűsített függvényeit foglalja magába. Legfőképpen a kamera által készített képek mentésére, azok feldolgozására és újra kirajzolására használtam, mint a kalibrálásnál, mint a tárgyak detektálásánál.

C++, Python, Java és MATLAB nyelveken is implementálták már, és támogatja a Windows, Linux, Android és Mac OS operációs rendszereket. Az OpenCV MMX és SSE utasításokat használ, ha rendelkezésre állnak. Jelenleg fejlesztés alatt van egy teljes értékű CUDA és OpenCL interfész is. Több mint 500 algoritmust és körülbelül tízszer annyi függvényt tartalmaz.

3. fejezet

Létező megvalósítások

A múltban számos módszert és megoldást láttunk már hand-eye kalibrációra. Amint azt a [19] cikkben is taglalják, ezen módszereket két kategóriába sorolhatjuk: olyan megoldásokra amik a hand-eye kalibrációt a hagyományos robot-kinematikai modellekkel összekapcsolódó módszerekre, mint például a [13], és olyan módszerekre ahol a hand-eye kalibrálást elkülönítjük a robot kinematikai modell kalibrálásától, mint például [18]. Ebben a cikkben [19] egy új, a második kategóriába tartozó megoldást is javasolnak, ami gyorsabb és pontosabb is.

A következő alfejezetekben két létező megvalósítást mutatok be automatikus kalibrációra, valamint egy megoldást adaptív összeszerelésre.

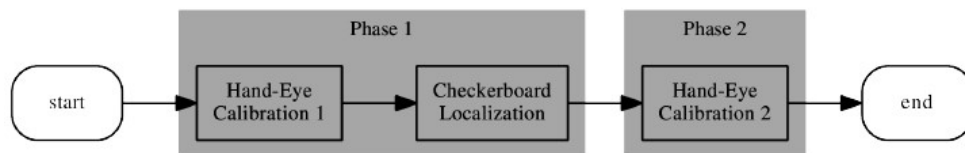
3.1. Robot manipulátor és több 3D kamerarendszer automatikus kalibrálása

A következő cikk [11] egy olyan keretrendszert mutat be, amely egy robotkar manipulátor segítségével automatikusan használható mind belső kameraparaméterek meghatározásához és a hand-eye kalibrációhoz. A működési elve a következő: először egyenletesen mozog egy szabványos négyzetes kalibrációs rács körül. A keretrendszer meglévő és megbízható kalibrációs megközelítéseket használ, de ezek a létező módszerek újszerű kombinálásán alapulnak azért, hogy a kalibrálási folyamat teljesen automatikus és adaptálható legyen tetszőleges számú 3D kamerához. Továbbá, készül egy eredmény a sakktábla eltolásának becslésére is, így különböző végeredmények használhatók. Ez egy időtakarékos és rugalmas megoldás, amely nem igényel semmilyen különleges eszközt a beállítás előkészítéséhez, csupán egy kissé módosított A4-es méretű

nyomatott négyzetes táblát. Az egész keretrendszer a Robot Operating System (ROS)-re alapszik, és a moduláris felépítését, illetve a robotok és érzékelők nagy számának rendelkezésre álló integrációját használja a típusokhoz. Az algoritmus minden egyes része több részre van osztva, ahol a modulok előre meghatározott üzenetformátumok segítségével kommunikálnak egymással. Ennek előnye, hogy a folyamat egyes részeit egyszerűen tudják módosítani anélkül, hogy a feldolgozás többi részét befolyásolnák, valamint, hogyha szükséges akkor további feldolgozási lépéseket tudjanak beépíteni. Továbbá, minden egyes keretmodul újra felhasználható, mivel a bemeneti és kimeneti modulok között a formátum megegyezik. Így a konfiguráció egyszerű módosításával cserélhetővé teszi a tényleges hardvert, a robotot és a 3D kamerákat, feltéve, hogy rendelkeznek a ROS által támogatott illesztőprogramokkal. Ezáltal csak minimális felügyeletre van szükség az egész folyamat során.

3.2. Teljesen automatikus hand-eye kalibrációs rendszer

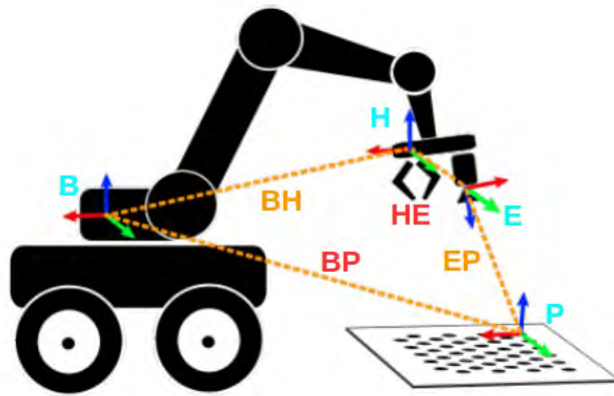
Ez a cikk [1] egy olyan automatizált rendszert mutat be, ami felismeri a kalibráló mintát és elvégzi a hand-eye kalibrációt, így ez egy lépéssel meghaladja a korábban létező rendszerek működési folyamatát. Az eddig megvalósított módszerek legfőképpen arra a megoldásra fókuszáltak, hogy a hand-eye kalibráció minél optimálisabb legyen, ahelyett, hogy megpróbálják teljesen automatikussá tenni a folyamatot, a kezdetétől a végéig. Ez a módszer egy két lépcsős folyamaton alapszik, amelynek mindegyikében elvégeznek egy hand-eye kalibrációt. A második fázis lehetővé teszi, hogy a kamerát a kalibrációs mintához képest mozgatva szabályozott beállítások mentén készítsen képeket. Mind az első, mind a második lépést tesztelték szimulációs és éles környezetben is, ezáltal bebizonyítva, hogy konzisztens eredményeket kapnak a második lépés után.



3.1. ábra. Két fázisból álló végrehajtási lánc [1].

Az automatikus folyamat akkor indul el, ha a robot már egy olyan kezdő-

pozícióra mozgott, ahol a kalibráló minta látható. Ez a folyamat két lépésre bontható, az első lépésben a nyers hand-eye transzformációt **HE** számolják ki, azokból a kinyert képekből amiket a kar a kéz **H** mozgásával készít a robot bázis vagy világ koordináta rendszerében **B** a stop-and-go mozgással. A kapcsolatok **BP** és **HE** között ismeretlenek, szóval ahelyett hogy a kamerát **E** mozgatták a kalibráló mintához **P** képest, csak a kezét **H** mozgatják a világ/robot bázishoz **B** képest.



3.2. ábra. A fő koordináta rendszerek jelölése [1].

Ahogy az korábban említésre került, ez az automatikus művelet olyan kiinduló helyzetből kezdődik, ahol a kalibráló minta a kamera látóterében van (Field of View [FoV]). A robotkar mozgatja a megfogót mindkét tengely mentén addig, ameddig a minta már nem látható. Állomásonként elvégeznek egy ellenőrzést, hogy a minta még mindig detektálható-e, vagyis látható. Ha igen, akkor a kinyert kép felhasználható a sakktábla kalibrálására és megtalálására, illetve a robot tovább mozoghat-e a tengelyeken.

Amikor a rendszer elvégezte az első hand-eye **HE** számítást és a sakktábla **P** látható a munkaterületen, a kamera szabadon mozgatható a tábla fölött, függetlenül a helyzetétől a munkaasztalon egy kijelölt és megismételhető útvonalon.

Ebben a fázisban a robot egy emberi kéz markoló mozdulatát utánozza, Thai és társai [19] javaslata alapján, a változó távolságok és szögek figyelembe vételével. A robot ezután ismét stop-and-go mozgást végez, szakaszosan meg-megállva, hogy újra leellenőrizze, hogy a kalibráló minta látható-e.

3.3. Önadaptív rendszer robot összeszerelési műveletekhez

Ebben a cikkben [10], egy nagy pontossággal (± 2.2 mm) működő adaptív összeszerelésre alkalmas robot rendszert láthatunk. A rendszer használható félig kialakított környezetben egy mozgatható robottal, ami fel van szerelve gépi látásra alkalmas rendszerrel. Ebben a gépi látásra alkalmas rendszerben 3D és 2D kamerák kombinációja van implementálva a tárgyak felismerése érdekében. Azért, hogy a környezethez alkalmazkodjanak, egy 3D térképet használnak ami az *Oct trees*-en alapszik, és egy ütközés mentes útvonalat számol a környezettől függően.

A rendszerben a robot helyzete tetszőlegesnek tekinthető (azaz miközben az adott műveletet végzi aközben a helyzete fix) és a környezet részben kialakított, azaz a robot a feladatát sík felületen végzi el (egy ipari munkaállomáshoz hasonlóan).

A javasolt rendszer alkalmazásának módja összeszerelési feladatokhoz a következő:

- Vigyük a mozgatható robot rendszert a munka környezethez és kézzel állítsuk be a 3D kameránkat hogy a kívánt területre fókuszáljon.
- Keresse meg az új 3D kamera koordináta rendszert a robot koordináta rendszerére hivatkozva hárompontos kalibráció segítségével.
- A munkakörnyezetet kiértékeli a mozgatható robotkarra rögzített 3D kamera és a tárgy helyzetét megközelítőleg kiszámítjuk az objektum kereső technikával.
- Minden objektumhoz egy megközelítési útvonalat számítanak ki, figyelembe véve a környezetben lévő akadályokat és korlátokat.
- Amikor megközelítjük a tárgyat, a 2D kamerát használjuk arra, hogy megkeressük a pontos helyzetét a 2D minta egyeztetésével.
- A 2D-s kamera alapján, a tárgy megfogására való mozgás kiszámítható és elvégezhető.

4. fejezet

Ipar 4.0

A technológiai fejlődés nem csak az emberek mindennapjait könnyíti meg különböző eszközökkel, hanem az iparra is nagy befolyása van. Azokat a munkaköröket, amiket eddig emberek végeztek el a saját hibalehetőségeikkel mára gépek végzik sokkal hatékonyabban és kisebb hibaaránnal. Ezt hívjuk ipari forradalomnak. Ezekből az ipari forradalmakból több is volt az elmúlt 250 évben. Az első, ahol a gőz energiáját hasznosították különféle eszközökhöz, majd a második, ahol már az elektromosság, illetve a tömeggyártás került előtérbe.

Majd a harmadik ipari forradalom, ahol már fél-automatizált, esetenként teljesen automatizált gépeket és gyártási folyamatokat valósítottak meg a programozható memóriák segítségével. Viszont mindezek eltörpülnek a most szemünk előtt megvalósuló negyedik ipari forradalomhoz vagy más néven az Ipar 4.0-hoz képest. Az Ipar 4.0-ban használt eszközök már annyira fejlettek, hogy nem csak egy adott, előre beállított funkciót képesek ellátni, hanem képesek egymással kommunikálni, illetve együtt működni. Ezek a gépek a gyártásoptimalizálás következő lépcsőfokai.

Az Ipar 4.0-ban 5 fő alapelvet igyekszünk követni (lásd 4.1.-es táblázat). Ezt az 5S módszernek nevezzük. A módszer japán eredetű és nem is feltétlenül az informatikához lett kitalálva, viszont ezen a területen is teljes mértékben lehet alkalmazni.

4.1. táblázat. Az Ipar 4.0 öt alapelve

Japán	Angol	Magyar	Magyarázat
Seiri	Organize	Válogatás	Elkülönítjük a kellő és nem kellő részeket (Szerszámokat, alkatrészeket, anyagokat)
Seiton	Set in Order	Rendezés	Rendezzük a megmaradt dolgokat olyan sorrendben, hogy egyszerű legyen használni
Seiso	Cleanliness	Tisztaság	Mindig tiszta környezetben dolgozzunk, a felesleges dolgok csak összezavarhatnak
Seiketsu	Standardize	Rendszerezés	Rendszerezük a karbantartási és tisztántartási feladatokat, betartva a seiri, seiton, seiso pontokat
Shitsuke	Sustain	Fenntartás	Ezeket az elveket felhasználva végezzük a munkánkat

A mai világban a technológia és ezáltal a robotika, az automatizáció, a tartós energiahordozók illetve a mesterséges intelligencia - azon belül a gépi tanulás - is rohamosan fejlődik. Már ma is több mint 2.25 millió robot dolgozik aktívan gyárakban, gyártósorokon, esetenként már a szórakoztató iparban is. Ezt a számot pedig előreláthatólag csak növekedni fogjuk látni. Számítások alapján igen valószínű, hogy 2030-ra több mint 20 millió robot fog dolgozni a világon.

A gyártóvállalatok és automatizált cégek már rengeteg különböző optimalizációs problémára találtak megoldást. A munkaerőhiánytól kezdve a költséghatékonyságon keresztül, a gyártás maximalizálásig. Legtöbbször ezek a robotok egymagukban dolgoztak vagy más robotokkal közreműködve. Viszont az Ipar 4.0-ban egyre nagyobb számban terjednek el az úgynevezett kollaboratív robotok vagy az UR (Universal Robots) által elnevezett kobotok. Ezek a robotok az emberi munkával kölcsönösen és párhuzamosan képesek működni, mivel olyan szenzorokkal vannak felszerelve amik biztonságossá teszik őket, hogy emberek között is működjének. Az egyik ilyen funkció, ami az én munkám során is többször segített, az a roboton lévő ellenállást érzékelő szenzorok. Ha a robot egy falnak vagy az asztalnak ütközik, akkor egy kis ellenállás után megáll és egy Protective Stop üzenetet jelenít meg a kezelőfelületén.

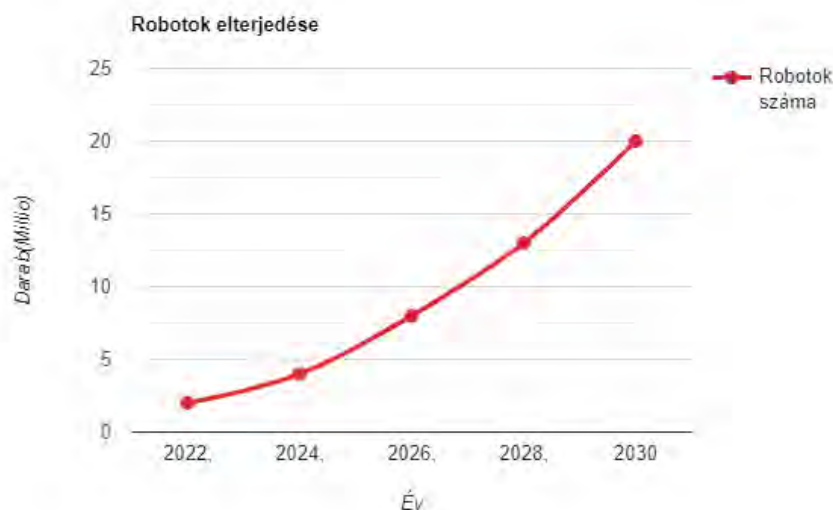
4.1. A robotika és az automatizálás szerepe az Ipar 4.0-ban

A robotika és automatizálás szerepe megkérdőjelezhetetlen az Ipar 4.0-ban [7]. Napjainkban, a cégek közti versengés egyre nagyobb. Pontosan ezért fontos, hogy a lehető legtöbbet hozzuk ki az adott helyzetből. Ez viszont csak úgy le-

hetséges hogyha okos és megfontolt döntéseket hozunk, ez a mentalitás enged teret az újszerű intelligens rendszereknek is, amik akár maguktól is képesek meghozni ezeket az okos döntéseket. Az olyan robotszerű gépezetek már évtizedek óta léteznek, amik egy kifejezett feladatot látnak el. Vegyünk példának egy minőségellenőrző robotot, ami képes kiszűrni a hibás termékeket, de erről még mindig értesítenie kell egy operátort, aki el is távolítja azt az adott hibás terméket a gyártósorról. A mai robotok ennél már fejlettebbek és nagyon nagy szerepet játszanak majdnem minden iparágban. Vegyük az előző példának egy mai változatát; egy minőségellenőrző robot ma már nem csak képes megállapítani, hogy egy termék hibás-e, hanem arra is képes, hogy magától, intelligensen, pontosan, de legfőképpen biztonságosan eltávolítsa azt a gyártósorról, méghozzá oly módon, hogy felismerje mi a hiba az adott terméken és csak a végeredményről értesítse az operátort. Ami viszont talán a legérdekesebb, hogy a robotika és a mesterséges intelligencia még mindig kiforrás alatt álló technológiák, amik az elkövetkező években rengeteget fognak még változni és fejlődni. Ezáltal fogják formálni az iparágakat amik ezekre épülnek és mindezek mellett a társadalmunkat is, amit ezek az iparágak határoznak meg. Hogy alátámasszam eme állításom, ezen a grafikonon (4.1.-es ábra) egy becsült növekedés látszódik a robotok számával kapcsolatban az elkövetkező 8 évre tekintve.

A gépi tanulás, illetve a mesterséges intelligencia fejlődésével a robotika is egy teljesen új szintre emelkedett. Egyre több teret nyernek maguknak a kollaboratív robotok, amik képesek felmérni és érzékelni a környezetüket és eszerint cselekedni, illetve tanulni. Ezzel a fejlődéssel megváltoztak a munkaterületek, körülmények, és a feladatok is, mind az emberek mind a robotok számára. Ez lehetővé tette a cégeknek, hogy képesek legyenek tartani a lépést a folyamatosan növekvő igények kielégítésével.

A technológiai fejlődés több rétegben látható még ma is. Amellett, hogy a mai gyártósorok már nagyon rugalmasak és alkalmazkodók tudnak lenni, megtartják automatizált működésüket is, így alkalmazkodva a vásárlók különböző, de egyre kifinomultabb igényeihez. Továbbá ez a technológiai fejlődés kihatással van az iparágban dolgozó emberekre is. Ezek a tanulásra alkalmas robotok sokkal széleskörűbbek és költséghatékonyabbak lesznek, mint azok a robotok amik most használatban vannak.



4.1. ábra. Robotok számának előirjelzése

4.2. Robotika alkalmazása az Ipar 4.0-ban

Az ipari forradalmak több alkalommal is új mércét állítottak az emberek életkörülményihez, és ez most sincs másképp [7]. A 4. ipari forradalom is hasonló módon változtatja meg életünket már most és a jövőre nézve is. Az Ipar 4.0 egyik alapvető lényege, hogy robot és ember kéz a kézben tudjon dolgozni egymás mellett, olyan módon, hogy az biztonságos és hatékony legyen. A legkézenfekvőbb példa erre, az autonóm robotok, amik már ma is megtalálhatók az iparban. A világunkban, ahol a vásárlók irányítanak és határozzák meg a mércét, egy cég legfőbb célja, hogy a lehető leghatékonyabban gyártsák le a termékeiket és a lehető legtöbb profitot generálják. Pontosan ezért minden hibás terméke egy cégnek, ami kikerül a piacra, komoly sebet okozhat a cég arculatának. Ezeknek a termékeknek a kiszűrése fontosabb, mint valaha.

Igaz viszont az is, hogy az Ipar 4.0 rengeteg más területen is hasznos. Például a következőkön:

- **Összeszerelés.** Sok szakértő állítja, hogy a robotkarok talán legnagyobb előnye az autoiparban, hogy olyan feladatokat is el tudnak látni ami emberek számára veszélyes lenne vagy esetenként fizikailag lehetetlen, mint például a hibrid akkumulátorok beszerelése ami gyakran többet

nyom, mint 100 kilogramm.

- **Önvezető járművek.** Az automatikusan irányított járművek (Automated guided vehicle [AVGs]) már most használatban vannak, főleg raktárakba/raktárakból szállítanak alapanyagokat, de a jövőben sokkal nagyobb szerepük is lehet, például személyszállításban vagy árukiszállításban és hasonlóknban. Ehhez viszont az kell, amit az Ipar 4.0 képvisel, vagyis, hogy ezek a járművek ne csak automatikusak legyenek hanem intelligensek is.
- **Egészségügy.** Több példát is láttunk a közelmúltban erre, viszont talán a legjelentősebb, a gyógyszergyártás területén született, ahol az operátorok és gyógyszerészek robotok segítségével biztonságos távolságból tudnak veszélyes gázokkal és vegyszerekkel gyógyszereket létrehozni amellet, hogy az automatizált gyártósorok hibaértéke és a költségük is kevesebb.
- **Katonai és közbiztonsági ipar.** A technológia gyors változásával a katonai ipar és a közbiztonsági ipar is jelentősen fejlődött. A katonai iparban drónokat használnak évtizedek óta, viszont az újszerű robotokat újonnan a harctéri támogatásban is bevetették, a felmérések elvégzésében és a katonai táborok őrzésében játszanak jelentős szerepet. Ezek a robotok/drónok a céloknak megfelelő mennyiségű fegyverzettel vannak felszerelve és képesek érzékelni a környezetet, így képesek támadni is ha szükségesnek találják.
- **Bányászat.** Robotok által irányított fúrófejeket már használnak bányákban, de ezen felül olyan kis bányász robotokat is, amik automatikusan járják az adott bányát, hogy mintákat nyerjenek ki a bent lévő anyagokról.

5. fejezet

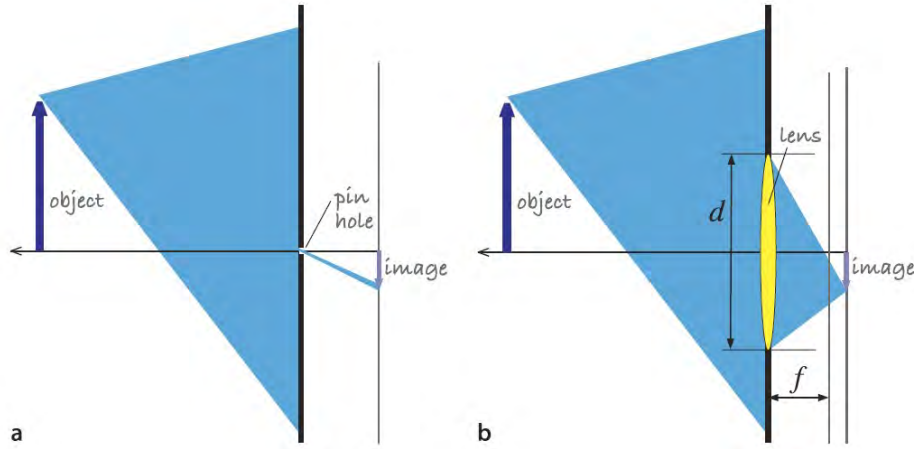
Kamera és robot kalibrálása, pózbecslés

Az alábbi fejezetben a legfontosabb fogalmakat fejtem ki, melyek az elkészült rendszer megértéséhez a későbbiekben nélkülözhetetlenek, melynek alapjául az alábbi könyvek szolgálnak [3], [20].

5.1. Perspektív kamera modell

Képzeljünk el egy apró lyukat a falon egy besötétített szobában. Ez egy alig látható invertált képet vetít a kinti világról a szoba másik oldalán lévő falra. Ezt a jelenséget az úgynevezett *pinhole* lyukkamerának hívjuk. A lyukkamera egy nagyon halvány képet vetít, mivel ezt a kép fénysűrűsége és a lyuk területének a szorzata határozza meg:

A világosabb képek titka az objektívek lencséjében rejlik, ahogy az látszik is a 5.1-es ábrán, amely egy nagyobb területről összegyűjti a fénysugarakat a tárgyról és azt továbbítja egyenesen a képre. Egy konvex lencse ugyan úgy képes képet alkotni, mint egy lyukkamera.



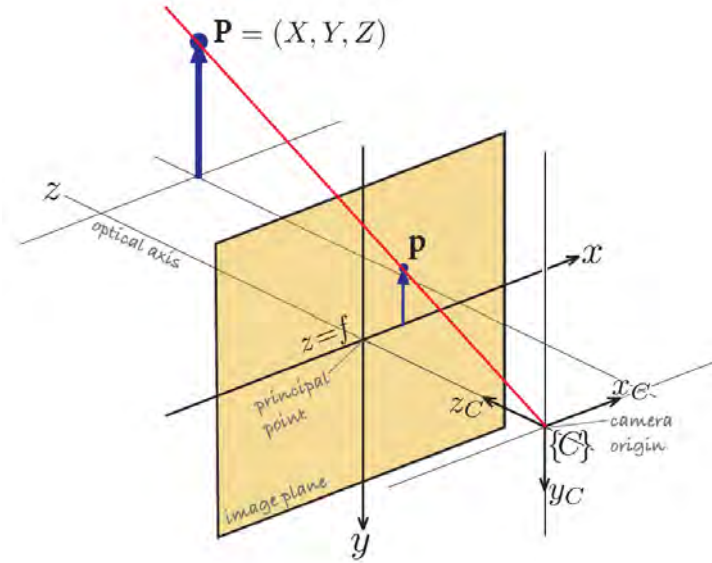
5.1. ábra. A lyukkamera **a** és az objektív **b** fénygyűjtő képessége [3]

A pozitív z tengely a kamera optikai tengelye. Az adott tárgy z koordinátája függ a kamera lencsájének a középpontjától, ez kiszámítható a vékony lencse egyenlettel (thin lens equation):

$$\frac{1}{z_0} + \frac{1}{z_i} = \frac{1}{f} \quad (5.1)$$

ahol a z_0 az adott tárgy távolsága, a z_i az adott kép távolsága és az f a fókusz távolság.

A lencsék hátránya viszont, hogy ügyelnünk kell a fókuszra is. A mi szemünkben is megtalálható egy konvex lencse, ami átlátszó kristályosodott protein láncokból áll, és a szemünk fókuszát egy izom állítja a láncok átformálásával - ezt a folyamatot alkalmazkodásnak (accomodation) hívjuk. Egy jó minőségű kamera több műanyag és üveglencséből áll össze.



5.2. ábra. A központi vetítési modell [3]

A számítógépes látásban gyakran használják a központi perspektivikus képalkotási modellt. A fénysugarak a kamera C origója felé konvergálnak és egy nem invertált képet a képsíkra vetítenek, amely a következő ponton található:

$$z = f \quad (5.2)$$

A z -tengely a képsíkot metszi a főpontban, ami történetesen a 2D-s képkordináta rendszer origója. Hasonló háromszögek segítségével megmutathatjuk, hogy a $P = (X, Y, Z)$ világkoordinátájú pont a $p = (x, y)$ képpontra vetül a következő módon:

$$x = f \frac{X}{Z}, y = f \frac{Y}{Z} \quad (5.3)$$

ami egy projektív transzformáció, pontosabban egy perspektivikus vetítés.

5.2. Perspektív kamera modellezése

A képsíkbeli pont koordinátákat felírhatjuk homogén formában $\hat{p} = (\hat{x}, \hat{y}, \hat{z})$, ahol:

$$\hat{x} = fX, \hat{y} = fY, \hat{z} = Z \quad (5.4)$$

Vagy kicsit kompaktabb mátrix formában:

$$p = \begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \quad (5.5)$$

Ahol a nem homogén képsík koordináták:

$$x = \frac{\hat{x}}{\hat{z}}, y = \frac{\hat{y}}{\hat{z}} \quad (5.6)$$

Ezeket gyakran retinális képsík-koordinátáknak nevezik. Abban az esetben, ahol $f = 1$ a koordinátákat normalizált retinális vagy normalizált képsík-koordinátáknak nevezzük.

Ha a világkoordinátát homogén formában is felírjuk ${}^c\tilde{\mathbf{P}} = (X, Y, Z, 1)^T$, akkor a perspektivikus vetítés lineárisan a következő módon írható le:

$$\tilde{\mathbf{p}} = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} {}^c\tilde{\mathbf{P}} \quad (5.7)$$

vagy

$$\tilde{\mathbf{p}} = C {}^c\tilde{\mathbf{P}} \quad (5.8)$$

Ahol a C egy 3×4 -es mátrix, azaz a kamera mátrix.

5.3. Kamera mátrix

A kamera vetületét általános formában a következőképpen írhatjuk le:

$$\begin{aligned}
 \tilde{\mathbf{p}} &= \begin{pmatrix} f/\rho_w & 0 & u_0 \\ 0 & f/\rho_h & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} ({}^0\mathbf{T}_C)^{-1} \tilde{\mathbf{P}} \\
 &= \mathbf{K}\mathbf{P}_0 ({}^0\mathbf{T}_C)^{-1} \tilde{\mathbf{P}} \\
 &= \mathbf{C}\tilde{\mathbf{P}}
 \end{aligned} \tag{5.9}$$

ahol az összes tagot a C kameramátrixba foglaljuk. Ez egy 3×4 -es homogén transzformáció, amely skálázást, translációt és perspektivikus vetítést végez. Ezt gyakran nevezik vetítési mátrixnak vagy kamera kalibrációs mátrixnak is.

A kamera leírásához 5 belső és 6 külső paraméter áll rendelkezésre, vagyis összesen 11 független paraméter. A kameramátrix 12 elemű, így egy szabadságfok nem korlátozott és tetszőlegesen megválasztható. Ez a gyakorlatra vetítve úgy néz ki, hogy a kamera paramétereket nem ismerjük ezért ki kell számítanunk a kamera kalibrációs folyamat segítségével.

5.4. Lencse torzítások

Egy lencse se hibátlan, és gyakran az olcsóbb, kevésbé minőségi lencsék, amiket például olcsó kamerákban, vagy webkamerákban használnak kifejezetten rossz képeket eredményeznek. A lencse tökéletlenségei különböző torzulásokat eredményeznek, beleértve a kromatikus aberrációt (színhiba), a szférikus aberrációt vagy asztigmatizmust (a fókusz eltérése a képen) és a geometriai torzulásokat, amikor a képsík pontjai eltolódnak a helyükről.

A geometriai torzulás jelenti általában a legnagyobb problémát, amivel találkozunk a robotikai alkalmazásokban, és ez két összetevőből áll: radiális és tangenciális. A radiális torzítás hatása, hogy a képpontok sugárirányú vonalak mentén elfordulnak a főponttól. A radiális hiba jól megközelíthető egy polinommal:

$$\delta r = k_1 r^3 + k_2 r^5 + k_3 r^7 + \dots \tag{5.10}$$

ahol r a képpont távolsága a főponttól.

Hordó torzulás akkor fordul elő, amikor a nagyítás csökken a főponttól való



5.3. ábra. Különböző torzítások

távolsággal, amelynél a kép széléhez közeli egyenes vonalak kifelé görbülnek.

A tűpárna torzítás akkor következik be, amikor a nagyítás a fókusponttól való távolsággal növekszik, és a kép széléhez közeli egyenes vonalak befelé görbülnek.

A tangenciális torzulás vagy decentralizáló torzulás a sugarakra merőlegesen jelentkezik, de általában kevésbé jellemző, mint a radiális torzulás.

5.5. Kamera kalibráció

A kamera kalibrálása a kamera saját paramétereinek meghatározása és a külső paraméterek meghatározása a világkoordináta-rendszerhez viszonyítva. Ezeket a paramétereket használhatjuk a lencse torzításának korrigálására, egy tárgy méretének világegységben történő mérésére, vagy a kamera helyének meghatározására a világban. Ezeket a módszereket olyan alkalmazásokban használják, mint például a gépi látás vagy objektumok észlelése és mérése. A robotikában, a navigációs rendszereknél és a 3D-s munkaterületek felmérésére is használják őket.

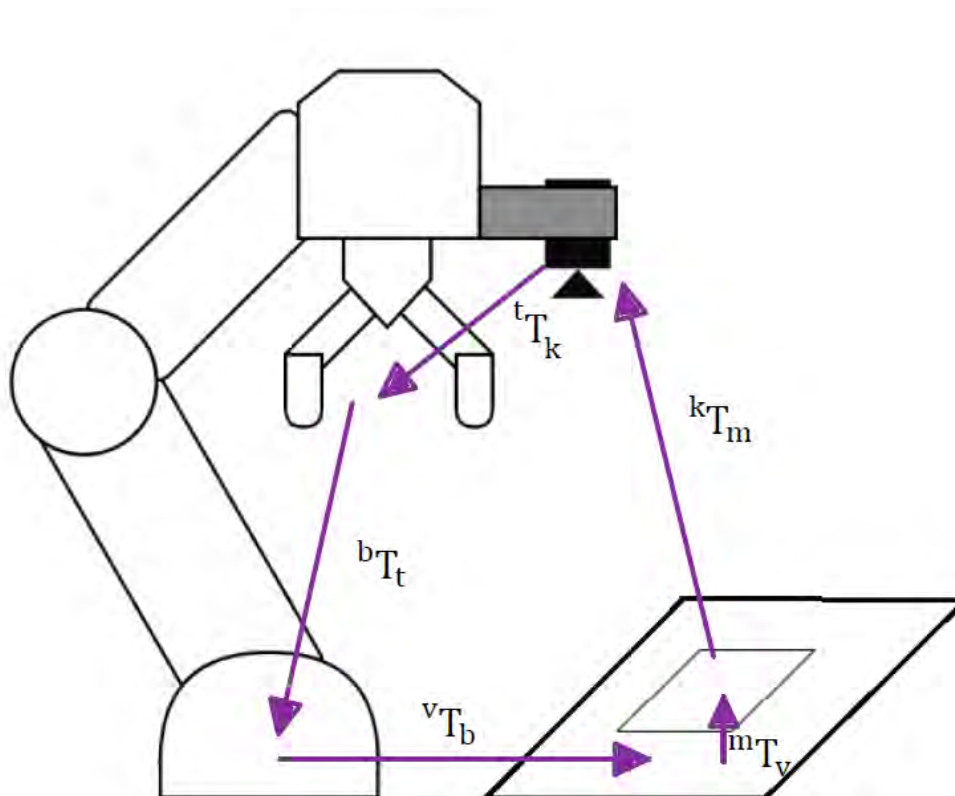
A kalibrációs technikák olyan 3D-s világpontok halmazára támaszkodnak, amelyek relatív koordinátái, és az ezeknek megfelelő képsíkbeli 2D-s koordinátái is ismertek. A legmodernebb technikákhoz egyszerűen szükség van egy síkbeli sakktáblás kalibrációs mintára (általában egy sakktáblára) és számos képre (amin a kalibrálás folyamata elvégezhető). Ebből megbecsülhetők a belső paraméterek (beleértve a torzítási paramétereket is), valamint a sakktábla relatív pozíciója az egyes képeken. Régebbi kalibrációs technikáknál egyet-

len képre volt szükség egy 3 dimenziós kalibrációs céltárgyról, de nem voltak képesek a torzítási modell kiszámítására.

Az egyik leggyakrabban alkalmazott technika kamera kalibrálásra az úgynevezett Gold Standard algoritmus, melynek lépései a következők:

- Lineáris (algebrai) megoldás
- Geometriai hiba minimalizálása: az előző lépésben kapott megoldással inicializálva minimalizáljuk (pl. Levenberg-Marquardt algoritmussal)
- Denormalizálás

5.6. Hand-eye kalibráció



5.4. ábra. Koordináta-rendszerek

A hand-eye (vagy magyarul kéz-szem) kalibráció lényege a meglévő koordináta-rendszerek közötti átmeneteket biztosító transzformációs mátrixok megállapítása, kiszámolása. Esetemben öt koordináta-rendszert állapítottam meg:

- modell - m
- kamera - k
- tcp/megfogó - t
- bázis - b
- világ - v

A különböző koordináta-rendszerek közötti átszámoláshoz a transzformációs mátrixok T meghatározása szükséges. Ezek egy 3×3 méretű rotációs mátrixból R és egy 3×1 méretű translációs vektorból t , valamint egy homogenitást biztosító $\begin{pmatrix} 0 & 0 & 0 & 1 \end{pmatrix}$ sorvektorból tevődnek össze:

$$\mathbf{T} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix} \quad (5.11)$$

Ezt a kalibrációs folyamatot szokás úgy is emlegetni, mint az $\mathbf{AX} = \mathbf{ZB}$ egyenlet megoldása, ahol - a 5.4.-es ábra figyelembevételével:

- $A \iff {}^kT_m$
- $B \iff {}^vT_b$
- $C \iff {}^tT_k$
- $D \iff {}^bT_t$

A kalibrálási folyamat a következő: egy kalibrációs mintát használunk a világ és a kamera koordináta-rendszerek közötti transzformáció becslésére, a robot megfogóját többféle pózba mozgatjuk. Minden póznál rögzítjük a megfogó és a robot koordináta rendszere közötti homogén transzformációt, valamint minden póznál rögzítésre kerül a homogén transzformáció a világi és a kamera koordináta rendszere között.

5.7. Robot kinematika

A robotkar működésének első lépése a céltárgy pozíciójának meghatározása, valamint a robotkar pontos helyzetének meghatározása a céltárgyhoz képest. Más szóval, meg kell határozni a köztük lévő orientációs kapcsolatot. A koordinátákra vonatkozó szükséges információk megszerzése után a koordinátatranszformáció egyes szakaszai elvégezhetők. A kinematika egy mozgásban lévő tárgy követésének leírására szolgál, és magában foglalja az előremenő és az inverz kinematikát is. Az előremenő kinematika az egyes csuklók forgási szögén keresztül határozza meg a robotkar pozícióját és végpontját. Az inverz kinematika a robotkar pózából és a végpontjának helyzetéből visszaszámítja az egyes tengelyek csuklóinak a szögét. Az előmenő kinematika az alapja az inverz kinematikának, és használható az inverz kinematika pontosságának ellenőrzésére. Az ipari alkalmazásokban gyakran a végpont pozíciók sorozata már előre meghatározott. Az inverz kinematikát arra használják, hogy meghatározzák az egyes tengelyek egyes pozíciókhoz tartozó csuklóknak a szögét, és ellenőrizzék a motorok vezérlését, hogy az egyes csuklók elérjék a meghatározott pozíciót. Ez lehetővé teszi, hogy a robotkar minden pozícióba képes legyen elmozogni [6].

5.8. Pózbecslés

A pózbecslés célja a célkoordináta pózának $\{T\}$ meghatározása a kamerához képest. A célpont geometriája ismert, azaz ismerjük az $(X_i, Y_i, Z_i), i \in [1, N]$ pontok helyzetét a célpozícióhoz a $T\{T\}$ viszonylatában. A kamera saját paraméterei is már ismertek. A képet rögzítjük, és a megfelelő képsíkbeli koordinátákat (u_i, v_i) számítógépes látás algoritmusok segítségével meghatározzuk.

A pózbecslését az $(u_i, v_i), (X_i, Y_i, Z_i)$ segítségével, illetve a kamera belső paramétereinek használatával, Perspective-n-Point problémának vagy röviden PnP-nek nevezzük. Ez egy jóval egyszerűbb probléma, mint a kamerakalibrálás vagy a dekompozíció, mivel a kevesebb paramétert kell megbecsülni.

6. fejezet

A feladat megvalósítása

Ebben a fejezetben azokat a függvényeket és általam használt módszereket fogom leírni, amiket a dolgozatom megvalósítása során használtam.

6.1. Mozgatások

Az egyik legelső megvalósítandó feladat a dolgozatomban a robot mozgatása volt. Ezt nagy mértékben leegyszerűsítette a Pythonban implementált URX könyvtár. Ezen szoftverkönyvtár elérhetővé tette számomra a *move()* függvényt, amit továbbfejlesztettem, és így használtam. A *move* függvény 8 változót vár; x, y, z translációs és R_x, R_y, R_z rotációs értékeket, továbbá egy *vel* (velocity) avagy maximum sebesség változót és egy *acc* (acceleration) vagyis gyorsulás változót. Az általam módosított függvény már csak az x, y, z koordinátákat várta és opcionálisan megadhatók maradtak az R_x, R_y és R_z koordináták. Ha nem kapnak értéket, akkor egy előre meghatározott érték állítódik be nekik. A *vel* és *acc* változókat egy osztályban tárolt változó határozza meg, amit a konstruktorban előre beállíthatunk.

Ezen felül főleg a debugolás szempontjából létrehoztam egy úgynevezett *WASD()* függvényt, aminek elindításával a robotot tudtam mozgatni fix értékkel az x, y és z -tengelyeken, oly módon, hogy a kamera kép szerint a *W* "előre" mozgatja a robotkart (az y -tengelyen negatív irányba a robotkar bázisa szerint) az *S* ugyanezt végzi el, csak "hátra" (bázis szerint pozitív irányba az y -tengelyen). Az *A* és *D* "balra" illetve "jobbra" mozgatja a robotkart (azaz az x -tengelyen pozitív és negatív irányban).

Továbbá létrehoztam ismételten a debugolás érdekében olyan függvényeket is, amiknél az R_x, R_y és R_z játszotta a fő szerepet, ugyanis a robotkart a dőlési

pozícióira mozgattam, ezáltal biztosítva, hogy jó helyről kezdje meg a döntött palettázást.

6.2. Palettázás

Ahogy arról már eddig többször szó esett a dolgozatomban, mindennek az alapja a kamera és a hand-eye kalibrációk pontos kivitelezése, mivel ezen kalibrációk elvégzése után lehet meghatározni, hogy a robot hol helyezkedik el a különböző koordináta rendszerekben. Ehhez viszont először sok képre van szükség egy jól látható kalibrációs mintáról. Ez a minta nálam egy sakktábla mintázatú, 6×8 -as kalibráló tábla - fekete és fehér négyzetekkel - volt. Ideális esetben erről a tábláról 5 különböző állásban (pontosabban egy sík helyzetben és négy döntött helyzetben) szerettem volna 5×48 , azaz 240 képet készíteni, viszont ez nem volt lehetséges, mivel a robot elhelyezkedése és fesztávja ezt nem tette lehetővé. Ez abból eredt, hogy az y -tengelyén a robot nem tudott megfelelő mértékben hátrafelé mozogni ahhoz, hogy eredményesen képeket tudjon készíteni a kalibráló tábláról a 4. döntött állapotában. Így a képek számát leredukáltam 192-re, vagyis 4×48 képre. Ez a mennyiség is elégnek bizonyult, egy sikeres kalibráció eléréséhez, ugyanakkor meg kell jegyeznem, hogy több képpel a kalibráció pontosságának esélye megnő.

Az egyetemen található Képfeldolgozás Kutatólaboratorium tette elérhetővé számomra a már a 2. fejezetben említett hardvereket. Ezekkel az eszközökkel egy olyan megoldást dolgoztam ki, ahol az UR5-ös robotra egy 3D nyomtatott bilincsel rögzítettem a Basler ace aca2440-35uc kamerát, így a robotkar szemszögéből tudtam képeket csinálni. A 3. fejezetben bemutatott folyamatok és módszerek közül is használhattam volna egyet, de úgy gondoltam, hogy ez a megoldás a legkézenfekvőbb a rendelkezésemre álló eszközökkel.

Amint rákerült a kamera a robotra, azonnal problémákba ütköztem, mivel a kamera kábele elakadt a robotban, illetve, ha még ennek ellenére is mozgott a robot, akkor a karra szerelt bilincs elmozdult, ezzel elmozdítva a kamerát is. Ezt hamar orvosoltam oly módon, hogy a kábelt többé-kevésbé lazán elvezettem a robotkar mentén, és a bilincs belsejét két oldalú ragasztóval a robothoz rögzítettem. Ezután már használható képeket készítettem a sakktábláról, viszont még ezek se voltak tökéletesek, de ezeknek az okáról egy kicsit később

tesztek említést.

A palettázást a *Palette()* függvényem végzi el, ez a függvény 6 paramétert vár, sorban x, y, z és R_x, R_y, R_z értékeket. Az értékek segítségével bemozog egy kezdőpozícióra és kezdő dőlésbe, ezeket a koordinátákat a *full dinamic()* függvényem határozza meg. Innen kiindulva pedig egy előre meghatározott lépésszámú mozgást tesz az x és y -tengelyeken. Ez a *rows* - azaz sorok - és *cols* - azaz oszlopok - változóiban vannak elmentve, és az értékük 6 és 8 volt, a kalibráló tábla sorai és oszlopai alapján. A robot mozgásának értékét az x -tengelyen az x_step változóval míg az y -tengelyen az y_steps változóval állítottam be. Ezeknek a változóknak a mértékét a kalibráló tábla sorainak és oszlopainak egyenlő részre való felbontásával számítottam ki.

A robot megkezdi a mozgást egyenletesen a tengelyeken. Először készít egy képet a kiindulópozíciójából, majd végig mozog az x -tengelyen még 7 képet készítve. A 8. kép elkészítése után a robotkar lefele halad egy lépést az y -tengelyen y_step mértékében és ugyanezt elvégzi visszafelé is. Így kapjuk meg egy dőlési pozícióban a 6×8 mennyiségű képet. Ennek a palettázásnak a végső formája viszont három felé lett szeparálva, mivel a debugolásnál nem mindig volt szükséges az összes pozícióban lefuttatni. Így a három külön funkció:

- csak horizontális; a kamerát teljesen bedöntés nélkül, csak a kalibráló lapra merőlegesen mozgatja,
- csak a 3 döntött állapot; a kamerát bedöntve mozgatjuk, és más perspektívákból csináljuk a képeket,
- a kettő kombinációja: a 4 palettázást lefuttatjuk egymás után, ami a 192 képet eredményezi.

Eleinte viszont akadtak problémák ugyanis az x_step és y_step nem volt tökéletesen beállítva, így finomhangolásra volt szükség. Azonban lefuttatni minden egyes alkalommal a palettázást körülményes lett volna idő szempontjából (körülbelül egy futtatás 13-14 percet vesz igénybe) így létrehoztam a *Palette_Corners()* nevű függvényemet, ami csak a kalibráló tábla sarkaira mozog el, ($col \times x_step$ az x -tengelyen és $row \times y_step$ az y -tengelyen) és egy kapcsolóval állíthatóvá tettem, hogy akár ezekbe a pozíciókba képeket is készítsen a kamera. Ezeket a palettázás mintájára végül 3 külön funkcióba szedtem szét.

6.2.1. Felmerült problémák

Amikor elkezdtem a dolgozatomban akkor nem gondoltam volna, hogy a legnagyobb problémát nem az ismeretlen nyelv, robotkar vagy akár a szerver fogja okozni, hanem a környezeti körülmények. Ebben az alfejezetben ki szeretném fejteni azokat a környezeti tényezőket, amik nagyban megnehezítették a palettázás folyamatát és ezáltal a robot pontosságát is. Először is fontosnak érzem, hogy megemlítssem magát a robotkart aminek - gyártói dokumentációk szerint - 0.5-0.6 mm a pontatlansági értéke, ami azt jelenti, hogy a robotkar ugyan abba a pozícióba körülbelül fél milliméter pontossággal tud elmozogni, többszöri ismétlés esetén. Ez a fél milliméter ugyan nem hangzik soknak, de amikor egy alig 1 milliméteres tárgyat akarunk megfogni, majd ezt behelyezni egy szintén alig több mint egy milliméteres foglalatba, akkor jelentős ez a hiba. Egy másik probléma már az egész feladat elején az asztallap volt, amin dolgoztam. Több szempontból is problémát okozott, ugyanis egyenletlen volt, valamint csúszkált is. Ezt végül megoldottam azzal, hogy egy egyenletesebb asztallapot raktam a munkaterületre, és azt gyorszorítókkal az eredeti asztallapra szorítottam, így kaptam egy (ugyan érezetes, de) síkfelületet.

Az egyenletlenség nem csak az asztallapnál jelentett problémát, hanem magánál a kalibráló táblánál is. A (közel) tökéletes kalibrálás érdekében ugyanis a kalibráló táblának teljesen sík állapotban kell lennie, ez alá tartozik az is, hogy az összes levegőt ki kell szorítani alóla, ami sokkal nagyobb kihívásnak bizonyult mint azt eredetileg gondoltam volna. Több módszerrel is megpróbáltam kiszorítani alóla az összes levegőt, de sose sikerült teljesen.

A képkészítésnél is akadtak problémáim, ugyanis nagyon nagy szerepet játszott a képek felhasználhatóságában a laborban elérhető fény mennyiség. A laborban a lámpák fel és le kapcsolásával próbálkoztam először és sikerült is egy olyan helyzetet kialakítanom, ahol elfogadható fényviszonyok között tudtam képeket készíteni, viszont mivel a terem nem teljesen sötétíthető be, ezért a kintől beszűrődő fény még így is problémát okozott olykor-olykor a palettázásban. A képkészítés másik problémája a kamera elmozdulása volt. Ahogy azt már az egyik előző fejezetben említettem, a kamerát egy 3D nyomtatott bilincsel rögzítettem a robotkarra, viszont ez enyhe húzásra is ami a kamera kábeléből adódott, elmozdult és a kalibrálás onnantól használhatatlanná

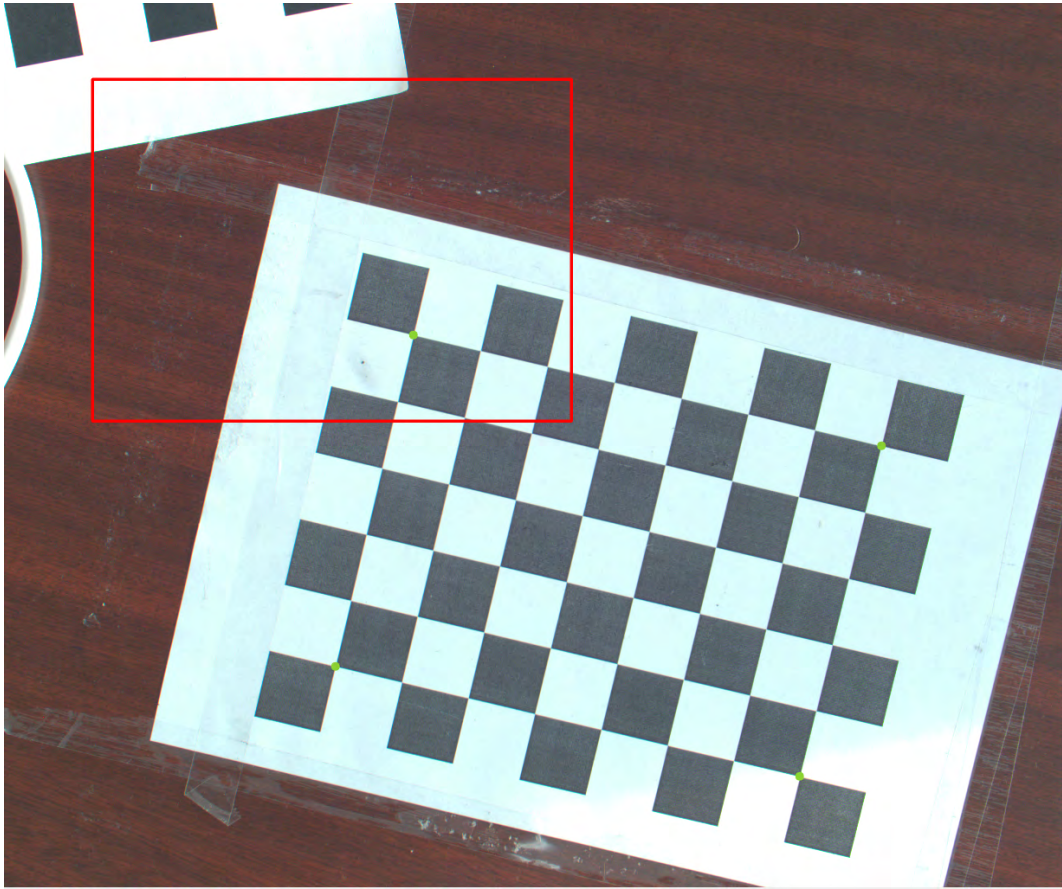
vált. Ezt sikerült megoldani azzal, hogy a bilincs belsejére kétoldalú ragasztót ragasztottam, így már kevésbé tudott elmozdulni.

Adódott a robotból két másik probléma is, miszerint valamely - számomra érthetetlen - okból fakadóan néha nem engedte a csatlakozást a gépemnek, hogy futtatni tudjam a kódokat. Ezt a "connetion timeout error" sikerült kiküszöbölni oly módon, hogy egy ciklusba beletettem a csatlakozáshoz szükséges kódrészletet és ezt a ciklust addig futtattam, ameddig a kapcsolat sikeres nem lett. A másik probléma az volt, hogy a robot huzamosabb használatba vétel után elkezdett nem rendeltetés szerűen viselkedni. Nem hajtott végre mozgást, vagy nem olyan mértékben, mint ahogy azt én meghatároztam neki.

6.3. Dinamikus megközelítés

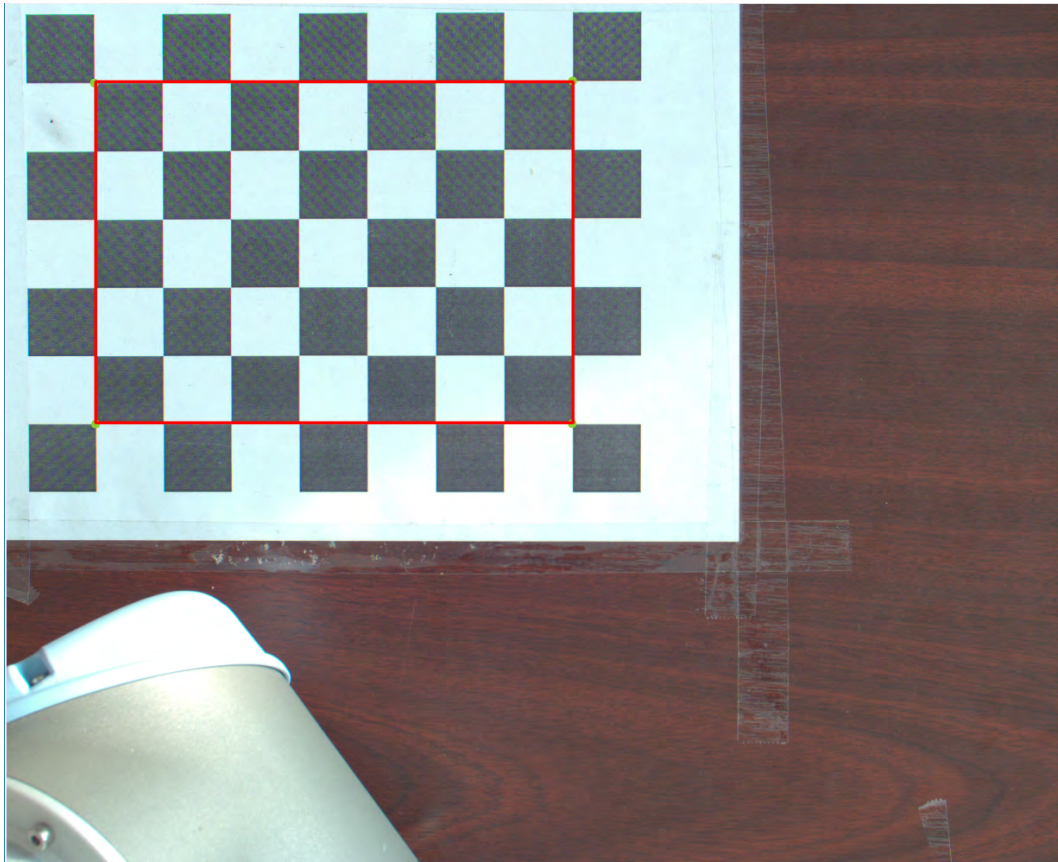
A palettázás megkezdéséhez a robotkarnak és az arra felszerelt kamerának a megfelelő kezdőpozícióra és állapotba (dőlésbe) kell mozognia. Ezt a dinamikus detektálással - *full_dinamic()* - függvénnyel oldottam meg.

Ezt a műveletet úgy végeztem el, hogy a képből kinyert pontok y koordinátáit kivonva egymásból, a különbséget abban az esetben, ha 100 pixel alatt van, megszoroztam egy fix értékkel, és a *step_y*-ba tároltam el. Így, ha a két-tő között nagy a távolság akkor nagyot mozgunk, ha kisebb akkor kevesebbet. Abban az esetben, ahol 100 pixel fölötti a különbség, akkor fix értékkel fordulunk el. Ezt addig ismételjük, ameddig egy kellően jó értéket kapunk. Ezt a kellően jó pixelkülönbséget a *difflimit* határozza meg. Az y -ra való beforgás után még leellenőrizzük az x -re is, viszont ez csak biztonsági művelet, mivel, ha az 1. és 3. pont y koordinátái megegyeznek, akkor az 1. és 2. pontok x koordinátái is meg fognak egyezni.



6.1. ábra. Dinamikus beállítás kezdő pozíció

A dinamikus detektálás oly módon működik, hogy ha a kamera képében megtalálható a kalibráló tábla 4 belső sarokpontja (ahol 2 fehér és 2 fekete négyzet találkozik) akkor azoknak a pixelkoordinátái alapján a robot először beforgatja magát az y -tengelyén úgy, hogy az 1. és 3. pont y koordinátája ideális esetben ugyan annyi kell, hogy legyen.



6.2. ábra. Dinamikus beállítás vég pozíció

A beforgás után az előző módszerhez hasonlóan beállunk az x majd az y -tengelyekre, ismételten oly módon, hogy ha az 1. és 3. pont x koordinátája nem egyezik az elvárt etalon koordinátával, ahol szeretnénk, hogy legyen, akkor kivonja az etalon x koordinátánkból az aktuális állapotban lévő x koordinátáját az 1. pontnak. Ha a különbség 200 pixel alatt van, akkor a különbséget megszorozzuk egy fix értékkel, majd ezt a $step_x$ változóba tároljuk el. Ezt ismételjük addig ameddig elfogadható értéket nem kapunk. Ezt az elfogadható értéket a $diff_limit$ változó határozza meg. Ezt 10-re állítottam az x és y esetében is. Ha sikeresen beálltunk az x -tengelyre akkor ugyanezt a műveletet megismétljük az y -tengelyre is. Ami ezt a folyamatot képes megszakítani az az, hogy ha megszakad a kapcsolat a szerverrel, ahonnan a képeket frissítjük minden mozgás után, vagy ha a képen nem találhatóak meg a pontok. Ezt okozhatja az is, hogy a robot a mozgásából adódóan nem találja, vagy a fényviszonyok

miatt nem található egy vagy több pont.

A sikeres beállítás után a koordinátákat elmentem, így nem kell a következő alkalommal megcsinálni újfent a dinamikus beállást (addig ameddig el nem mozgatjuk a kalibráló táblát), hanem csak betöltjük a file-ba a már kimentett adatokat és onnan tudjuk futtatni a palettázást.

6.4. Megfogó

A megfogóval való kommunikáció nehézkesen indult mivel a robotkar és a megfogó alapvetően nem kompatibilis. A socket könyvtár segítségével sikerült végül egy script file-t átadni a robotkaron keresztül nyers formában a megfogónak. A megfogónak 3 állapotot állítottam be;

- *grripper_Open*
- *grripper_Hold*
- *grripper_Close*

Egy másik hasznos dolog, amit elvégeztem a megfogó segítségével, az az, hogy a programom detektálja, ha a megfogó rajta van-e a roboton, és aszerint dinamikusán beállítja a z értékünket. Ezt eredetileg manuálisan akartam megoldani egy kérdéssel, és operátor inputtal, de végül úgy döntöttem a dinamikus jobban illik a dolgozatom témájához. A program ezt úgy hajtja végre, hogy kicsit megmozgatja a megfogót, így biztosítva, hogy kapcsoljon be az is, viszont a kompatibilitási probléma, miatt a kommunikációnk csak egy oldalú. Viszont, ha a megfogó rajta van a robotkaron, akkor a z koordinátánk a megfogó hosszával lentebb lesz, mint alapértelmezetten lenne, így megtudjuk állapítani, hogy rajta van-e. Az egyetlen probléma ami ilyenkor adódhat, hogy ha a megfogó rajta van a robotkaron de a csatlakozásával valami probléma adódott. Ebben az esetben a robotkar működése veszélyes lehet a megfogóra nézve.

6.5. Kamera és hand-eye kalibráció

A kamera kalibrálásának lényege, - ahogy azt már a 5. fejezetben taglaltam - hogy megadja a kamera belső és külső paramétereit, amiket később felhasználunk.

nálva a további feldolgozás során pontos méréseket végezhetünk. A kamera kalibrálásához a kezdeti lépés egy sakktábla többszöri lefényképezése volt.

Kétféleképpen készíthetünk képeket: a sakktábla rögzítve marad, majd pedig több képet készíthetünk a kamera mozgatásával. A másik módszer, hogy a kamera fixen marad, és helyette csak a sakktáblát mozgatjuk.

A képek készítésénél arra törekedtem, hogy a sakktábla minél jobban ki legyen tolvá a kamerakép széléig. Ez létfontosságú volt, mivel a nemlineáris torzítások jellemzően jelentősebbek az objektív határai közelében.

Az OpenCV biztosít egy függvényt a sakktábla sarokpontjainak megtalálásához, *findChessboardCorners()* néven. A függvény a sarkok képpontjait keresi meg és adja vissza. A függvényhez a keresett minta mérete is szükséges – ezúttal egy 8×6 -os rácsról van szó. Ha a program megtalálja a sakktábla összes sarkát a képen, akkor hozzáadja az objektumpontokat egy listához.

Ezenkívül a jó eredmények eléréséhez elengedhetetlen, hogy a sarkok elhelyezkedését szubpixel szintű pontossággal határozzuk meg. Az OpenCV-ben erre is találtam egy megfelelő függvényt, ez a *cornerSubPix()*. Úgy működik, hogy elkészíti az eredeti képet a sarkok elhelyezkedésével együtt, és az eredeti hely egy kis szomszédságában keresi a legjobb sarok helyét. Vegyük figyelembe, hogy az algoritmus iteratív jellegű, ezért meg kell határozni egy befejezési kritériumot (például az iterációk számát és/vagy a pontosságot).

Miután a program feldolgozta az összes képet, következhetett a kalibrálási folyamat. Ehhez a *calibrateCamera()* függvényt használtam. A kamera kalibráció elvégzéséhez átadom az ismert 3D pontok értékeit és az ezeknek megfelelő, detektált 2D pontokat. A függvény visszaadja a belső (kamera mátrix, torzítási együtthatók) és a külső paramétereket (forgatási és transzlációs vektorok) is.

A belső paraméterek ismeretében így már képes voltam a képek torzításmentesítésére. Ez azért volt fontos, mert a kamera lencsének torzítása van, amelyet torzításmentesítéssel szüntethetünk meg. Az OpenCV két függvényt biztosít ehhez a művelethez, de mindkettő ugyanazt az eredményt adja vissza. Az én esetemben az *undistort()* függvényt használtam.

Végül már csak a hand-eye kalibrációt volt szükséges végrehajtani. Az OpenCV ehhez is biztosított egy függvényt, *calibrateRobotWorldHandEye()* néven. Használatához biztosítani kell a kamera és a világ, valamint a bázis

és a TCP közötti transzformációs mátrixokat. Kimenetként pedig visszatér a TCP és a kamera, valamint a bázis és a világ közötti transzformációs mátrixokkal. Fontos, hogy minimum 3 mérésre szüksége van.

6.6. Alaplap detekció

Az alaplap detektálását a témavezetőm által tervezett algoritmus és program hajtja végre [17]. Futtatását egy másik programom végzi, amit a fő program csak elindít. A program két fő részből áll, az egyik a jellemző pontok kinyerésére szolgál, míg a másik az alaplap pózbecslésére.

Az alaplap esetében a kulcspontok az mATX szabvány specifikációja szerinti rögzítési furatoknak felelnek meg. A program célja, hogy megtalálja ezeket a kör alakú pontokat, majd elmentse a körök középpontjaként ábrázolt képpont koordinátákat. A feature matching azt jelenti, hogy két hasonló adatkészletből kell megtalálni a megfelelő jellemzőket. Esetemben ez azt jelentette, hogy meg kellett találnom az objektum modellpontjainak megfelelő képpont koordinátákat. A feldolgozási lánc jellemzőkinyerésének lépései öt fő lépésből állnak

A folyamat a NYÁK-ról vett kép kulcspontjainak manuális kiválasztásával kezdődik. Nevezzük ezeket tesztképeknek. Ezek a kulcspontok az alaplap felhasználási pontjai. Miután kiválasztottam ezeket a képeket, kivágom őket, és ROI-képeket (region of interest) készítek belőlük. A következő lépés a SIFT jellemző illesztő és leíró algoritmus alkalmazása. A SIFT segítségével a két kép – a teszt és a ROI kép – között jellemzőillesztést hajtok végre, és kulcspontokat vonok ki. Ezt követően kiszámítom a homográfiai mátrixot, hogy megtaláljam a két kép közötti koordináta transzformációt. Ezután a ROI képen megtalálom a PCB rögzítőfuratainak közepét a Hough-kör transzformációval. Végül a perspektivikus transzformáció alkalmazásával kiszámítom a próbaképen a kép furatának középpontjának koordinátáit. Ennek az egész folyamatnak az volt a célja, hogy a tesztképen megtaláljuk az alaplap rögzítési pontjainak közepét.

A kamera abszolút pózbecslése azt jelenti, hogy meghatározzuk a helyzetét és az orientációját egy referencia 3D világkoordináta rendszerhez képest. A pózbecslést általában a Perspective-n-Point algoritmussal (vagy röviden PnP-vel) hajtják végre.

A PnP lényege, hogy megtalálja a pozíciót és az orientációt adott számú

3D pont és a hozzájuk tartozó 2D vetületek között. A megvalósítás a következőképpen történik: a program az előző lépésben kinyert képkoordinátákat és a hozzájuk tartozó 3D modell koordinátákat igényli, valamint egy képet az alaplapról.

Először megbecsülöm a tárgy pózát a kamerához viszonyítva a PnP algorit-mussal, így kapok hat darab számot; egy három elemű vektort, ami a tárgy translációját, míg a másik három elemű vektor, ami az alaplap rotációját hivatott leírni. Ezt követően a rotációs vektort Rodrigues transzformációval 3×3 -as mátrixszá alakítom át, és hozzáfűzöm a translációs vektor transzpo-náltját. Végző lépésben kiegészítem az így 3×4 -es mátrixot egy 1×4 -es sor-vektorral (melynek értékei: $\begin{pmatrix} 0 & 0 & 0 & 1 \end{pmatrix}$). Ezekkel a lépésekkel megkaptam a tárgy pózát a kamera koordináta rendszerben, valamint már ismerem a transz-formációs mátrixot a két koordináta rendszer (modell - kamera) között, ahogy az az 5.11-es egyenletben is látható.

```
def move_to_slot(self, z):
    self.gripper_close()
    time.sleep(self.wait_between-3)
    actual_pose = self.rob.get_pose()
    save_pose = np.hstack((actual_pose.orient.array, np.
reshape(actual_pose.pos.array * 1000, (-1, 1))))
    save_pose = np.vstack((save_pose, np.array([0,0,0,1])))
    print(save_pose)
    np.save("Actual_pose", save_pose)
    time.sleep(self.wait_between-2)

    os.system('python feature_matching.py')
    time.sleep(self.wait_between-2)
    os.system('python pose_est.py')
    time.sleep(self.wait_between-2)

    slot_coords = np.load("Slot_Coords.npy", allow_pickle=
True)
    print("Slot_Coords (loaded) : ",slot_coords)
    self.initial_pose(slot_coords[0]/1000, slot_coords
[1]/1000, 0.200, 3.155, 0.000, 0.012)
```

6.1. Kódrészlet. Foglalatra való mozgatus

6.7. Memória modul detekció

Ahogy az alaplap pózbecsléséért felelő program sem az én munkám, úgy a memória modulokat detektáló program sem. Az általam használt algoritmusért [15] egy, a Képfeldolgozás Kutatólaboratoriumban nyári szakmai gyakorlatát töltő francia hallgatót illeti az elismerés. A detekció főbb lépései az alábbiak:

- ArUco markerek detekciója: 4 ArUco markert helyeztünk el a memóriatálcánk sarkaiban. A 2D koordinátáik megismeréséhez detektálnunk kell az ArUco kódokat a képen.
- Pózbecslés: A korábban kiszámított ArUco marker koordinátákkal, a memóriatálca modellel és a belső paraméterekkel kiszámoljuk a kamera külső paramétereit, amelyek lehetővé teszik a korábban kiszámított 3D pontjaink kivetítését.
- Pontok vetítése: Ebben a lépésben kinyertük a kameránk belső és külső paramétereit. Most már képesek vagyunk kivetíteni a szükséges 3D-s pontokat, hogy kisebb képeket nyerjünk ki a slotjainkról.
- Osztályozás használata memóriamodulok észlelésére: Több címkézett adatkészletet hozunk létre, és egy bináris osztályozási modellt tanítunk annak észlelésére, hogy egy kisebb kép tartalmaz-e memóriamodult vagy sem.

Ezek után a programot kiegészítettem - az alaplap detekciójához hasonlóan - annak érdekében, hogy meghatározzam a memória tálcát, és a benne lévő modulok pozícióját.

6.8. Összeszerelés

Végső lépésként a már meglévő programok és függvények felhasználásával a célom az volt, hogy a robotkar sikeresen kivegyen egy memóriát a memória tálcából és behelyezze azt az alaplap egyik memória foglalatába. Az eredeti elképzelésemben ezt egy 3D nyomtatott kis megfogó ujjvéggel akartam megoldani viszont ezzel problémák adódtak, méghozzá olyan mértékben, hogy egy új

megoldást kellett keresnem. Így arra jutottam, hogy jobb lenne, ha a robotkar merőlegesen a memória közepére fogna rá, mivel így kisebb az esélye annak, hogy a RAM akármilyen irányba is elmozduljon. Itt megemlíteném, hogy a jövőben elképzelhetőnek látom, hogy egy jobb 3D nyomtatott megoldással, véghez lehessen vinni ezt a feladatot nagyobb pontossággal, viszont számomra ez az opció időhiányból eredően nem állt fenn.

Első lépésként a robotkar elmenti a memória tálca pozícióját illetve az alaplappozícióját magához mérten a `pose_save()` függvénnyel (6.2.-es kódrészlet). Erre technikailag, ha csak egy memóriát akarnánk behelyezni nem is lenne szükség, viszont a jövőre való tekintettel, beleírtam a kódba, hogy képes legyen működni akkor is ha több memóriát akarnék behelyezni. A pozíciók mentése után a RAM tálca fölé mozgunk, ahol lefuttatjuk a memória detekciót ezzel kinyerve, hogy mely foglalatokba vannak a memória moduljaink. Ezt követően odamozgunk az első memória foglalathoz, majd leereszkedve a `gripper_Hold()` paranccsal megfogjuk azt. A memóriával együtt átmozog a robotkar az alaplapp fölé. Amit már tud, hogy hol helyezkedik el az alaplapp - az előzőleg már file-ba elmentett pozícióból. Amint a megfelelő helyzetbe került a robotkar, le is futtatjuk az alaplapp detektáló programot (6.6. kódrészlet). Ez vissza fogja adni a memória foglalat pozícióját, de a rotációját nem. Ezt úgy oldottam meg, hogy a rotációs vektorból meg tudtam állapítani a pontok elforgatását és azt átszámolva a robothoz való beforgatását.

```
def pose_save(self, z, motherboard = True):
    self.gripper_close()
    time.sleep(self.wait_between-2)
    pos = self.rob.get_pos()
    print(np.shape(pos))

    if motherboard:
        np.save("Motherboard initial pose:", pos[0], pos
[1])
        print("Pose of the motherboard is saved!")
    else:
        np.save("RAM tray initial pose: ", pos[0], pos[1])
        print("Pose of the RAM tray is saved!")
```

6.2. Kódrészlet. Pózok kimentése

Miután megvan a memória foglalat helye, illetve rotációja is, a robotkar leereszkedik a foglalat fölé és beleejti a memóriát. A foglalatba helyezett memória viszont nem tökéletes még, mivel nincsen teljesen jól belenyomva így arra a megoldásra jutottam, hogy a robotkart kicsit felmozgatva, a *gripper_Close()* függvénnyel összecsukom teljesen a megfogó ujjait és visszaengedem a memória tetejére, ezzel egy kis nyomást helyezve felülről a memóriára, ami éppen elég ahhoz, hogy belepattanjon a foglalatba.

7. fejezet

Tesztek és mérési eredmények

Ebben a fejezetben azokat a tesztek fogom kifejteni amiket a robottal végeztem el, a pontosság és ismételhetőség megméréséhez. Szó lesz még arról is, hogy az adott tesztre miért volt szükség, hogy alkottam meg a tesztesetet, és természetesen végül az eredmények kiértékelése is szóba kerül.

7.1. Robot pontossági teszt

A robot pontossági tesztje alatt az ismételhetőség pontosságát értem, ugyanis a 2.2-es táblázatban - ahol az UR5-ös robot technikai adatai találhatóak - a gyártó azt állítja, hogy ± 0.1 mm a pontatlanság. Ezt le akartam tesztelni magamnak, ugyanis az előző bekezdéshez hasonlóan, ha ilyen vékony tárgyakkal akarunk dolgozni, akkor a pontosság és megismételhetőség kifejezetten fontos. Ehhez egy egyszerű tesztet gondoltam ki: veszek egy objektumot, az én esetemben a kalibráló táblát, - de ez lehetett volna az alaplap vagy a RAM tálcá is akár - ezután két tetszőleges pontot kiválasztok ezen az objektumon, majd e között a két pont között elkezdem mozgatni a robotkart oda-vissza, többször egymás után, különböző sebességekkel. Eredmények a 9. fejezetben találhatók.

7.2. Kalibráció pontosságának a tesztelése

A kalibráció pontosságának meghatározása volt a legfontosabb, ugyanis jelentős különbség van pl. 0.7 és 0.3 mm közötti eltérés között, amikor egy ilyen kis méretű tárgyat szeretnénk behelyezni egy alig nagyobb foglalatba. A teszteset ebben az esetben egyszerű volt, mivel csak le kellett futtatnom a már meglévő

kalibrációs folyamatot többször, több fényviszony és környezeti tényező között.

Első lépésként lefuttattam egy palettázást, majd egy hand-eye kalibrációt. Az ebből kiértékelődött MSE egy jó kiinduló érték volt arra, hogy mennyire pontos a kalibráció, viszont én úgy döntöttem, hogy egy fizikai "éles" tesztet is lefuttatok, mivel az elméleti pontatlanság és a gyakorlati pontatlanság se mindig ugyanannyi.

A palettázás folyamatát már említettem, de a teszteléshez létrehoztam egy másik függvényt is, amit *mb_points()*-nek - a 7.1-es kódrészleten látható - neveztem el. Itt a kinyert képből az alaplap detektáló program - ami a 6.6. alfejezetben lett bemutatva - visszaadja a foglalatok koordinátáit. A robotkart odamozgatva megtudtam nézni, hogy mekkora pontossággal mozog a kinyert pontok koordinátáira.

Eredmények a 9.2. alfejezetben.

```
def mb_points(self, z):
    base_coords = np.load("base_coords.npy", allow_pickle=
True)
    for i in range(8):
        self.initial_pose(base_coords[i][0]/1000,
base_coords[i][1]/1000, z, 3.144, -0.121, 0.005)
        time.sleep(pal.wait_between-2)
        self.initial_pose(base_coords[i][0]/1000,
base_coords[i][1]/1000, 0.200, 3.144, -0.121, 0.005)
        time.sleep(pal.wait_between-2)
        w = input()lf.initial_pose(slot_coords[0]/1000,
slot_coords[1]/1000, 0.200, 3.155, 0.000, 0.012)
```

7.1. Kódrészlet. Furatpontok megadása

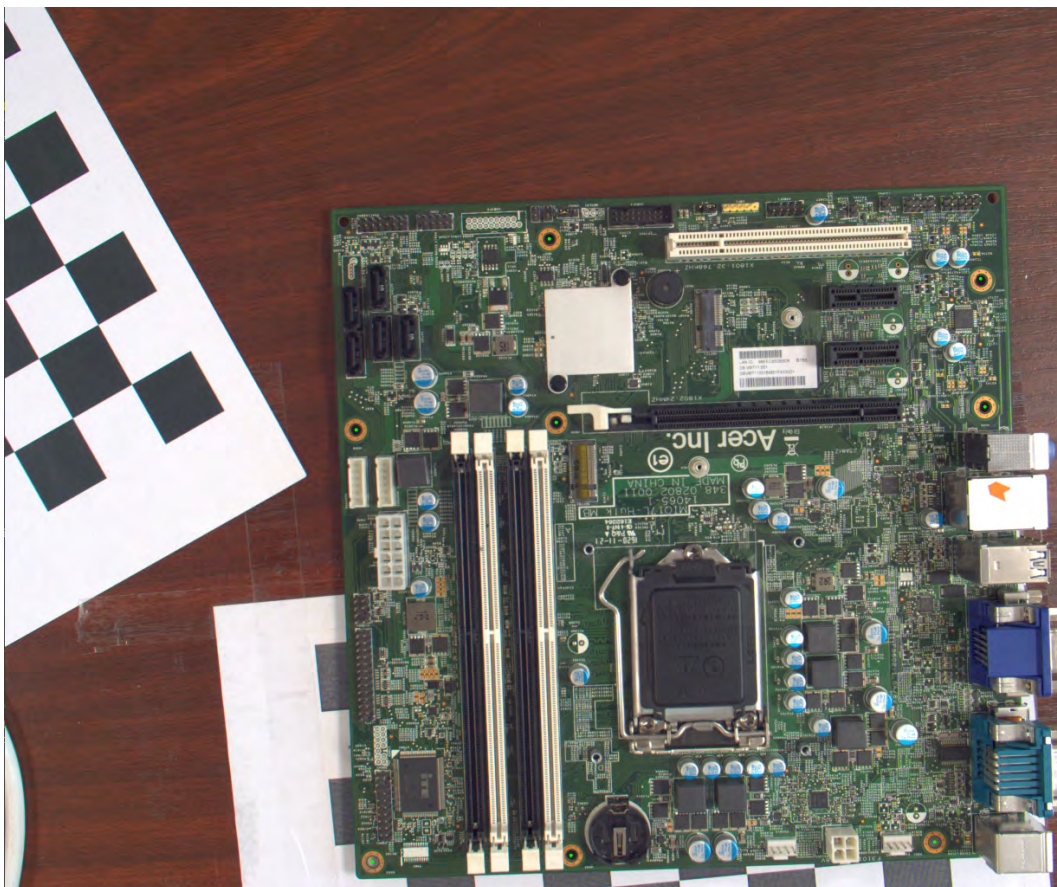
7.3. Detektálás pontosságának tesztelése

A detektálási folyamat pontossága is kifejezetten fontos, mivel a robot pontatlansága nem javítható nagy mértékben, így ha a detektálás is pontatlanságot eredményez akkor ezeknek az együttes pontatlansága már meggátolja a memória kivételét és/vagy behelyezését. Ezért láttam fontosnak erre a funkcióra is tesztek futtatni, különböző fényviszonyokban és elforgatásokban. Ehhez nem kellett egy különösen bonyolult kódrészletet csinálnom, mivel a

feature_matching.py és a *pose_est.py* programok visszaadnak minden olyan képet és értéket amire szükségem van ehhez a teszthez, úgyhogy ezeket csak futtatnom kell egy kódrészletben. Ehhez összesen ennyit kellett létrehoznom az *os* könyvtár segítségével.

```
def detect_test():  
    os.system('python feature_matching.py')  
    os.system('python pose_est.py')
```

7.2. Kódrészlet. Detektáló teszt kód



7.1. ábra. Feature matching példa az alaplapon. A zöld pontok a furatok közepén jelölik a megtalált pontokat.

Eredmények a 9.3-as alfejezetben foglalnak helyet.



7.2. ábra. Pózbecslés példa. A zöld pont a memória foglalaton jelöli a kívánt pont helyzetét.

7.4. Memória megfogás és behelyezés

A dolgozatom legvégső és egyben legfontosabb lépése a memória behelyezése az alaplapha, így ennek pontosságát megmérni elengedhetetlen. Teszteset amit erre a célra terveztem az felettébb egyszerű, de annál fontosabb. A behelyezésre szolgáló program részletet többször is lefuttattam több pozícióra és megnézem, hogy sikerül-e behelyezni, illetve, ha nem akkor hol van a probléma, x esetleg y -tengelyen vagy az elforgatásba.

8. fejezet

Összefoglalás

Végül pedig szeretném összefoglalni, hogy mivel foglalkoztam a dolgozatom készítése során. A TDK dolgozatom fő témája és célkitűzése az volt, hogy egy olyan algoritmus fejlesszek ki és implementáljak, mely dinamikusan képes bekalibrálni egy robotkart, amin egy ipari kamera, valamint egy megfogó helyezkedik el. Továbbá a robotkar képes kellett, hogy legyen egy memória modul kivenni egy tálcából és behelyezni azt egy alaplap tetszőleges foglalatába, mindezt úgy, hogy a hardver komponensek bárhol lehetnek a munkaterületen. A megvalósításhoz több lépés vezetett.

Első feladatomban a dolgozatom megírásához szükséges háttérirodalom megismerése volt. Ezek közé tartozott az Ipar 4.0 fogalmának és főbb tulajdonságainak a feltárása, a gépi látással és robotikával kapcsolatos fogalmak megértése, és a használandó eszközök elsajátítása.

Második feladatomban a kamera és hand-eye kalibráció automatizálása, dinamikussá tétele volt. Ehhez egy olyan algoritmust fejlesztettem ki, és implementáltam, mely képes a robotkart és az arra felszerelt kamerát a megfelelő kezdőpozícióra és állapotba (dőlésbe) mozgatni. Teszi mindezt automatikusan, bármiféle emberi beavatkozás nélkül. Ezután tudtam elvégezni a palettázás műveletét, ahol képeket készítettem a sakktábláról és elmentettem a robot aktuális pózát is a kamera és a hand-eye kalibrációhoz. Végül elvégeztem az említett kalibrálásokat.

Harmadik feladatomban már létező objektum detektáló és pózbecslő algoritmusok alkalmazása és fúziója volt. Erre azért volt szükség, hogy mind az alaplapot, mind pedig a tálcát, benne a memória modulokkal felismerjem, és képes legyek megállapítani a pózukat. Mindezt sikeresen elvégeztem, ahogy a tesztek is bizonyították.

Negyedik fő feladatomban az elkészült programok tesztelése volt. Négy különböző tesztet állítottam fel és végeztem el, melyeket táblázatokkal és képekkel dokumentáltam (lásd: 7. és 9. fejezet).

Az általam megvalósított szoftverről elmondható, hogy robusztus, továbbá megbízhatóan és nagy pontossággal működik, viszont további apróbb fejlesztések, tesztelések szükségesek lehetnek, ha ipari környezetben is alkalmazni szeretnénk.

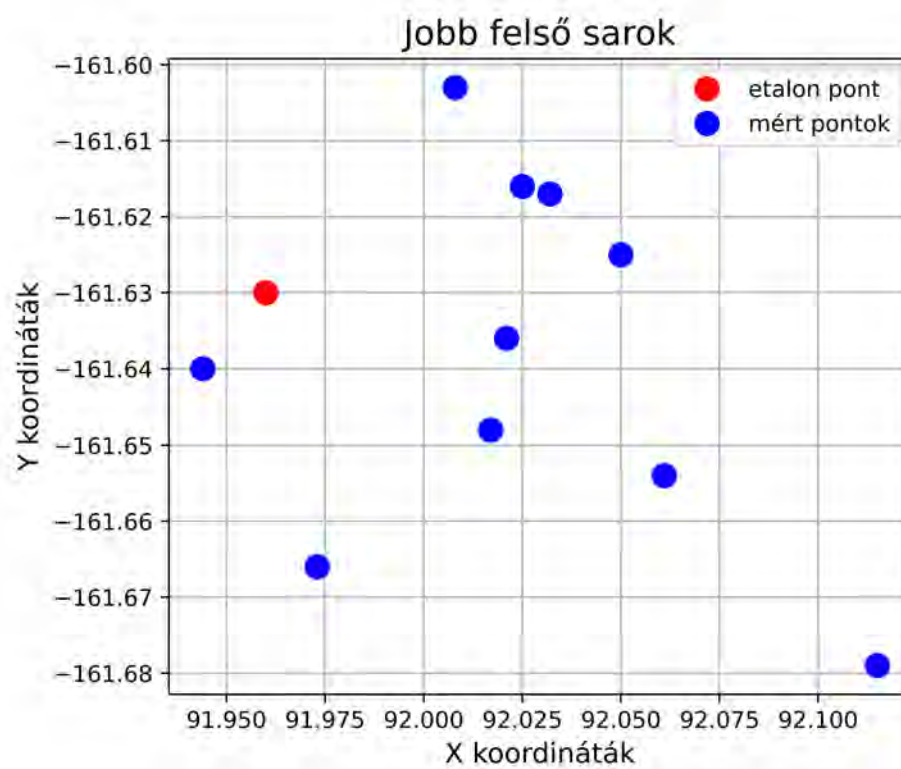
Továbbfejlesztésre is van lehetőség; az alaplap detektáló program viszonylag lassúnak mondható, egy 5 MP-es kép feldolgozása - még úgy is, hogy a műveletek párhuzamosítva vannak - 3 másodpercet vett igénybe. Ezt lehetne gyorsítani azzal, hogy az OpenCV-t forrásból telepítjük fel Cuda támogatással, ezáltal egy GPU végezhetné el a számításigényesebb feladatokat.

További továbbfejlesztési lehetőség lehetne a detektáló és pózbecslő programokat kiváltani egy mindenre kiterjedő neurális hálózattal [9]. Jó fejlesztési irányoknak mutatkozna egy igényes grafikus kezelőfelület készítése is az elkészült keretrendszerhez.

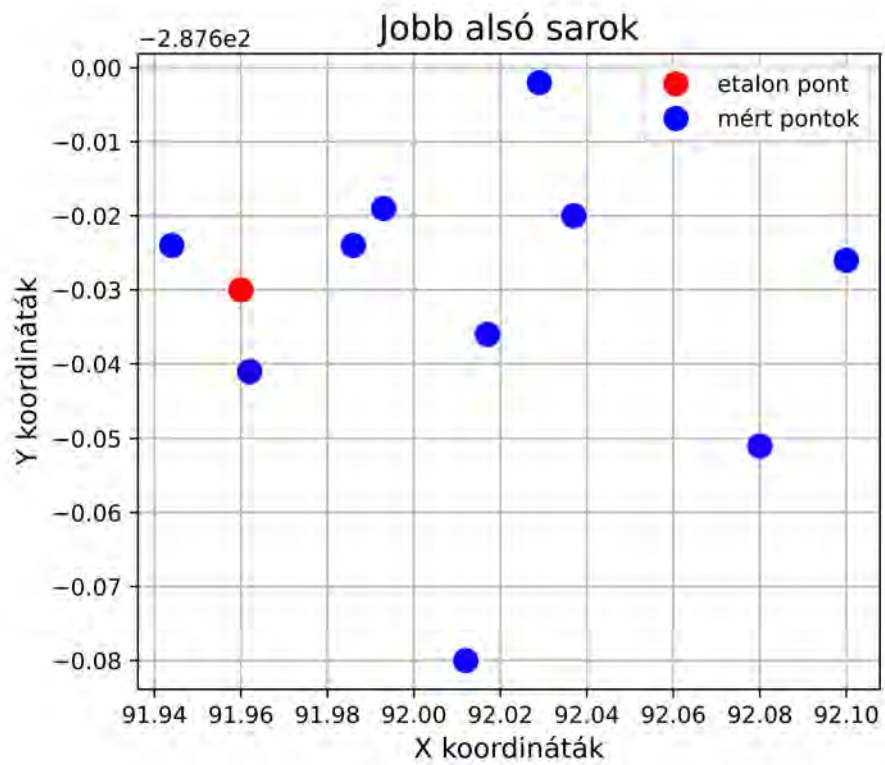
9. fejezet

Függelék

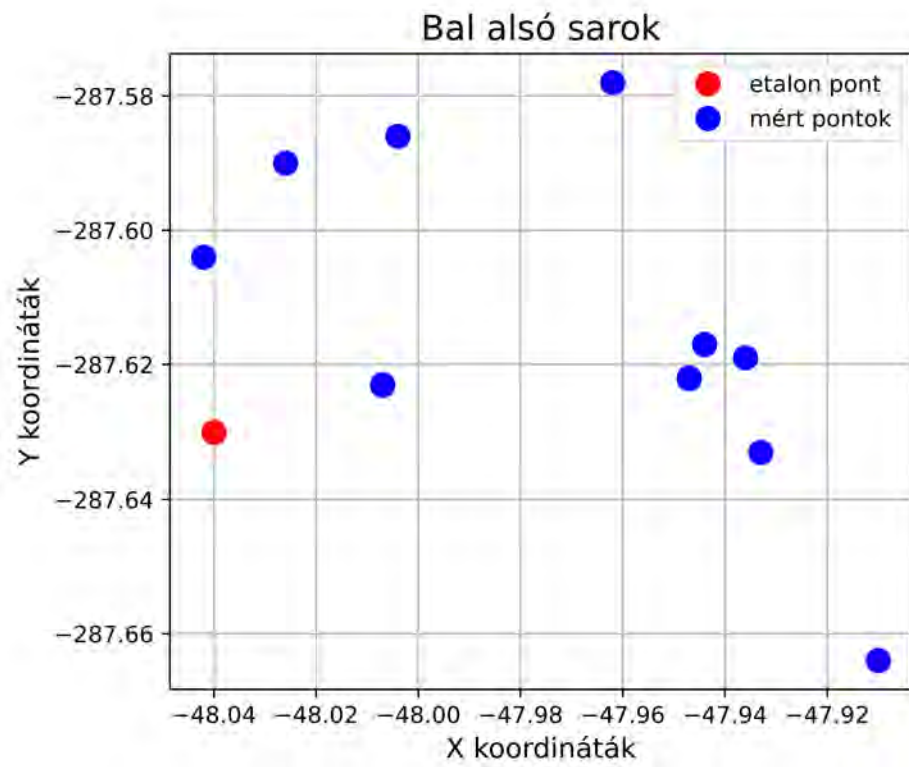
9.1. Robot pontossági teszt



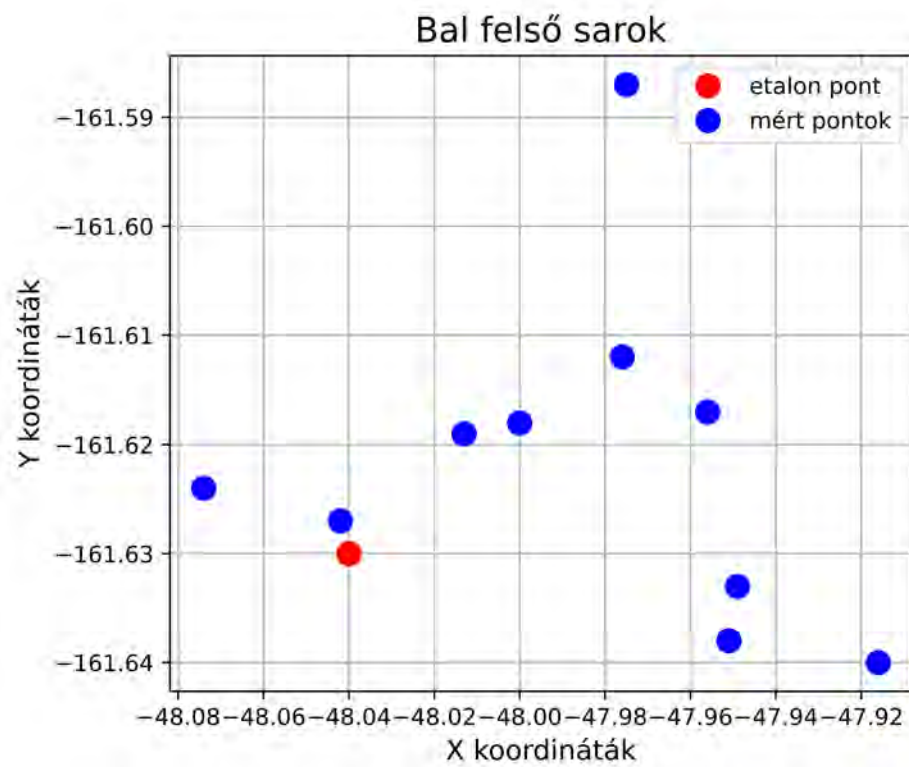
9.1. ábra. Jobb felső pontok eltérése



9.2. ábra. Jobb alsó pontok eltérése



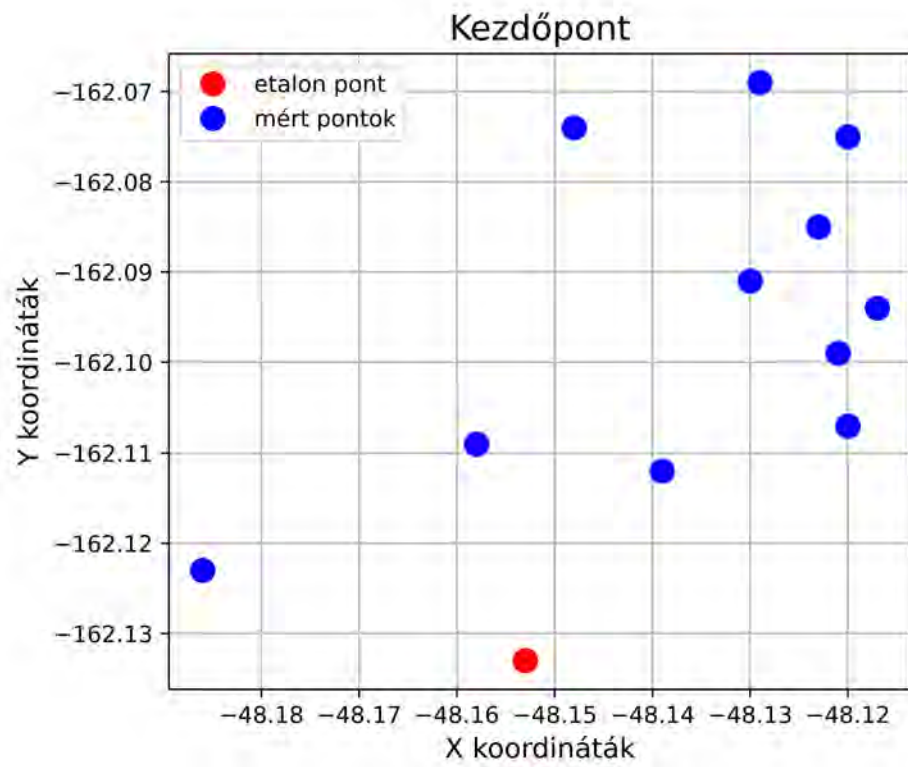
9.3. ábra. Bal alsó pontok eltérése



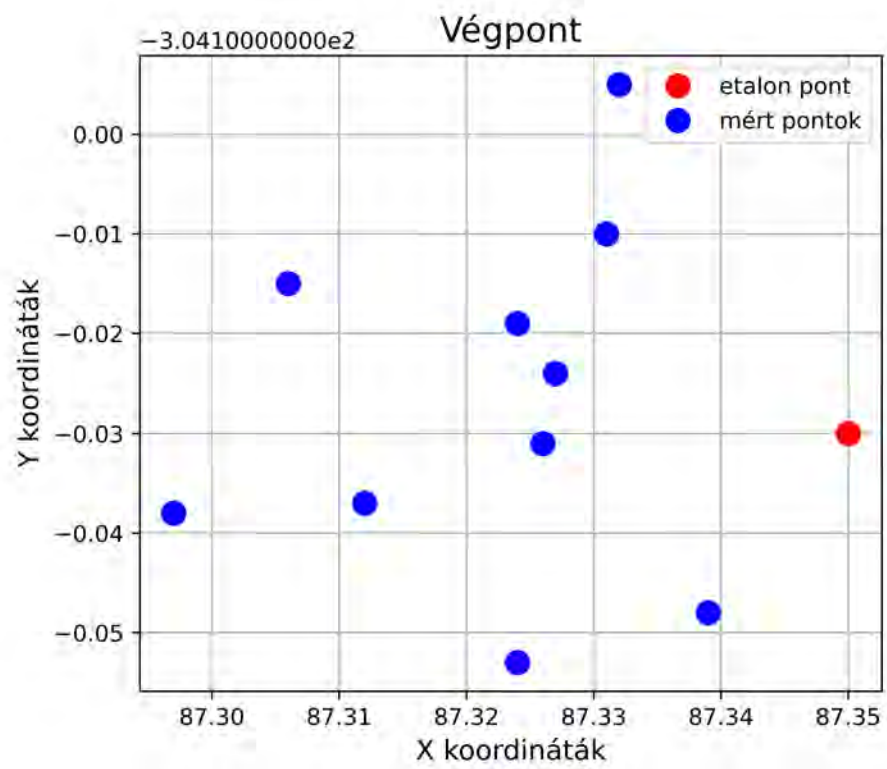
9.4. ábra. Bal felső pontok eltérése

9.1. táblázat. Mérések, sarokok

	Jobb felső sarok		Jobb alsó sarok		Bal alsó sarok		Bal felső sarok	
	X	Y	X	Y	X	Y	X	Y
<i>Eltérés, 1. mérés</i>	-0.155	0.049	-0.120	0.021	0.130	0.033	-0.124	0.009
<i>Eltérés, 2. mérés</i>	-0.101	0.024	-0.140	-0.004	-0.103	-0.010	-0.091	0.003
<i>Eltérés, 3. mérés</i>	-0.090	-0.004	-0.052	0.050	-0.095	-0.012	-0.088	0.008
<i>Eltérés, 4. mérés</i>	-0.072	-0.013	-0.077	-0.0099	-0.092	-0.007	-0.083	-0.013
<i>Eltérés, 5. mérés</i>	-0.057	0.018	-0.057	0.006	-0.106	0.002	-0.064	-0.018
<i>Eltérés, 6. mérés</i>	-0.061	0.006	-0.069	-0.028	-0.077	-0.052	-0.064	-0.043
<i>Eltérés, 7. mérés</i>	-0.048	-0.026	-0.026	-0.0059	-0.033	-0.007	-0.0399	-0.012
<i>Eltérés, 8. mérés</i>	-0.065	-0.013	-0.033	-0.010	-0.0139	-0.040	-0.027	-0.010
<i>Eltérés, 9. mérés</i>	-0.013	0.036	-0.002	0.011	-0.036	-0.043	0.002	-0.002
<i>Eltérés, 10. mérés</i>	0.015	0.009	0.0159	-0.005	0.002	-0.02	0.033	-0.006
<i>Átlagos MSE</i>	0.0033		0.0028		0.0037		0.0026	
<i>Szórás</i>	0.0467	0.0239	0.0486	0.0214	0.0450	0.0253	0.0476	0.0152



9.5. ábra. Kezdőpont



9.6. ábra. Végpont

9.2. táblázat. Sarok

	Kezdőpont		Végpont	
	X	Y	X	Y
<i>Eltérés, 1. mérés</i>	0.0330	-0.0100	0.025	0.023
<i>Eltérés, 2. mérés</i>	-0.024	-0.064	0.024	0.000
<i>Eltérés, 3. mérés</i>	-0.031	-0.034	0.0189	-0.019
<i>Eltérés, 4. mérés</i>	-0.033	-0.026	0.037	0.007
<i>Eltérés, 5. mérés</i>	-0.030	-0.048	0.010	0.018
<i>Eltérés, 6. mérés</i>	0.005	-0.024	0.022	-0.005
<i>Eltérés, 7. mérés</i>	-0.022	-0.042	0.018	-0.034
<i>Eltérés, 8. mérés</i>	-0.004	-0.058	0.025	-0.010
<i>Eltérés, 9. mérés</i>	-0.013	-0.021	0.043	-0.014
<i>Eltérés, 10. mérés</i>	-0.036	-0.039	0.052	0.007
<i>Átlagos MSE</i>	0.0011		0.0006	
<i>Szórás</i>	0.0216	0.0171	0.0129	0.0179

9.2. Kalibráció pontosságának a tesztelése

9.3. táblázat. Kalibráció pontosságának tesztelése különböző környezeti viszonyok között

Esetek	Környezeti viszonyok	Hibás képek	Jó képek	Kam. kalib. MSE	Hand-eye MSE
Eset 1	Term. fény	1	191	0,04441	0,04231
Eset 2	Term. fény	1	191	0,04686	0,15547
Eset 3	Sötétebb	1	191	0,05021	0,66412
Eset 4	Túl. Exp.	175	17	0,07346	N/A
Eset 5	Hullámos kalib. tábla	2	190	0,08442	1,43948
Eset 6	Ideális	1	191	0,04647	1,15982
Eset 7	Ideális	6	186	0,0494	0,79403
Eset 8	Ideális	7	185	0,04884	0,43915
Eset 9	Ideális	5	187	0,04764	1,14018
Eset 10	Ideális	0	192	0,04367	0,44466

9.3. Detektálás pontosságának tesztelése

9.4. táblázat. Feature matching eredmények, ideális fényviszonyban

	Eset 1	Eset 2	Eset 3	Eset 4	Eset 5
Furat \ Fény- viszony	Ideális	Ideális	Ideális	Ideális	Ideális
A	Jó	Jó	Jó	Jó	Jó
B	Jó	Jó	Jó	Jó	Jó
C	Jó	Jó	Jó	Jó	Jó
D	Jó	Jó	Jó	Jó	Jó
E	Jó	Jó	Jó	Jó	Jó
F	Jó	Jó	Jó	Jó	Jó
G	Jó	Jó	Jó	Jó	Jó
H	Jó	Jó	Jó	Jó	Jó

9.5. táblázat. Feature matching eredmények, nem ideális fényviszonyok esetén

	Eset 6	Eset 7	Eset 8	Eset 9	Eset 10
Furat/ Fény- viszony	Term. fény	Term. fény	Túl. Exp.	Túl. Exp.	Alul Exp.
A	Jó	Jó	Jó	Jó	Célon kívül
B	Jó	Jó	Jó	Perem	Célon kívül
C	Jó	Jó	Jó	Jó	Célon kívül
D	Jó	Jó	Jó	Célon kívül	Célon kívül
E	Jó	Jó	Jó	Elcsúszott	Jó
F	Jó	Jó	Jó	Jó	Célon kívül
G	Jó	Jó	Jó	Jó	Célon kívül
H	Jó	Perem	Jó	Perem	Jó

9.6. táblázat. Feature matching eredmények összesítve

	Esetszám	%
Jó	69	86,25
Perem	3	3,75
Elcsúszott	1	1,25
Célon kívül:	7	8,75
Össz	80	100

9.7. táblázat. Pózbecslés a memória foglalaton

	Fényviszony	Slot
Eset 1	Ideális	Jó
Eset 2	Ideális	Jó
Eset 3	Ideális	Jó
Eset 4	Ideális	Jó
Eset 5	Ideális	Jó
Eset 6	Term. fény	Jó
Eset 7	Term. fény	Jó
Eset 8	Túl. Exp.	Jó
Eset 9	Kiégett	Jó
Eset 10	Alúl Exp.	Elcsúszott

9.8. táblázat. Pózbecslés eredmények összesítve

	Esetszám	%
Jó	9	90
Elcsúszott	1	10

9.4. Memória megfogás és behelyezés

9.9. táblázat. Memória behelyezés

Eset száma	Fényviszony	Hiba X-tengelyen	Hiba Y-tengelyen	Forgatási hiba	Sikeres-ség
Eset 1	Ideális	0.2 mm	0.1 mm	0.3°	Igen
Eset 2	Ideális	0.6 mm	0.4 mm	1°	Nem
Eset 3	Ideális	0.1 mm	0.1 mm	0.2°	Igen
Eset 4	Ideális	0.3 mm	0.2 mm	0.4°	Igen
Eset 5	Ideális	0.7 mm	0.4 mm	0.2°	Igen
Eset 6	Alul. Exp.	1 mm	0.4 mm	0.4°	Nem
Eset 7	Túl. Exp.	1.2 mm	0.8 mm	1°	Nem
Eset 8	Túl. Exp.	0.9 mm	0.6 mm	0.7°	Nem
Eset 9	Természetes	0.5 mm	0.3 mm	1.4°	Nem
Eset 10	Természetes	0.6 mm	0.2 mm	0.8°	Igen

Irodalomjegyzék

- [1] Morris Antonello, Andrea Gobbi, Stefano Michieletto, Stefano Ghidoni, and Emanuele Menegatti. A fully automatic hand-eye calibration system. In *2017 European conference on mobile robots (ECMR)*, pages 1–6. IEEE, 2017.
- [2] Basler. Basler aca2440-35uc. https://twiki.cern.ch/twiki/pub/Sandbox/EquipmentAndInstallationLaserSetup/Ace_USB_3_0_Users_Manual.pdf.
- [3] Peter I Corke and Oussama Khatib. *Robotics, vision and control: fundamental algorithms in MATLAB*, volume 73. Springer, 2011.
- [4] Cliff Fitzgerald. Developing baxter. In *2013 IEEE Conference on Technologies for Practical Robot Applications (TePRA)*, pages 1–6. IEEE, 2013.
- [5] Fabrizio Flacco, Torsten Kröger, Alessandro De Luca, and Oussama Khatib. A depth space approach to human-robot collision avoidance. In *2012 IEEE international conference on robotics and automation*, pages 338–345. IEEE, 2012.
- [6] Bo-Lin Jian, Chi-Shiuan Tsai, Ying-Che Kuo, and Yu-Sying Guo. An image vision and automatic calibration system for universal robots. *Journal of Low Frequency Noise, Vibration and Active Control*, 40(1):347–366, 2021.
- [7] Pimal Khanpara and Sudeep Tanwar. Additive manufacturing: concepts and technologies. In *A Roadmap to Industry 4.0: Smart Production, Sharp Business and Sustainable Development*, pages 171–185. Springer, 2020.

- [8] Kowa. Kowa lm12jc5m2. <https://www.kowa-lenses.com/en/lm12jc5m2-5mp-industrial-lens-c-mount>.
- [9] Yann Labbé, Justin Carpentier, Mathieu Aubry, and Josef Sivic. Cosypose: Consistent multi-view multi-object 6d pose estimation. In *European Conference on Computer Vision*, pages 574–591. Springer, 2020.
- [10] Duc Tho Le, Mayur Andulkar, Wenchao Zou, J Philipp Städter, and Ulrich Berger. Self adaptive system for flexible robot assembly operation. In *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–5. IEEE, 2016.
- [11] Justinas Miseikis, Kyrre Glette, Ole Jakob Elle, and Jim Torresen. Automatic calibration of a robot manipulator and multi 3d camera system. In *2016 IEEE/SICE International Symposium on System Integration (SII)*, pages 735–741. IEEE, 2016.
- [12] OnRobot. Onrobot rg2 specifications. [https://onrobot.com/sites/default/files/documents/RG2_User%\\$20_Manual_enEN_V1.9.2.pdf](https://onrobot.com/sites/default/files/documents/RG2_User%$20_Manual_enEN_V1.9.2.pdf).
- [13] G. Puskorius and L. Feldkamp. Calibration of robot vision. In *IEEE Int. Conf. on Robotic and Automation*. IEEE, 1987.
- [14] Panjawee Rakprayoon, Miti Ruchanurucks, and Ada Coundoul. Kinect-based obstacle detection for manipulator. In *2011 IEEE/SICE International Symposium on System Integration (SII)*, pages 68–73. IEEE, 2011.
- [15] Remi Orveau. Recognition and pose estimation of memory modules in a memory tray with a monocular camera, 2022. Internship in the Image Processing Research Laboratory at the University of Pannonia in Veszprem, Hungary.
- [16] Universal Robots. Universal robot 5 specifications. https://www.universal-robots.com/media/50588/ur5_en.pdf.
- [17] Rádli Richárd Bence. Optical Pose Estimation in Robotics Applications. Kari Tudományos Diákköri Konferencia, 2020. Pannon Egyetem.

- [18] Yiu Cheung Shiu and Shaheen Ahmad. Calibration of wrist-mounted robotic sensors by solving homogeneous transform equations of the form $ax = xb$. 1987.
- [19] Roger Y Tsai and Reimar K Lenz. Real time versatile robotics hand/eye calibration using 3d machine vision. In *Proceedings. 1988 IEEE International Conference on Robotics and Automation*, pages 554–561. IEEE, 1988.
- [20] Kató Zoltán and Czúni László. *Számítógépes látás*. Typotex, 2011.