



Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Irányítástechnika és Informatika Tanszék

Képi információ felhasználása robotkar mozgatásához

SZAKDOLGOZAT

Készítette
Széplaki Ádám Zoltán

Konzulens
dr. Kiss Bálint

2015. május 24.

Tartalomjegyzék

Kivonat	4
Abstract	5
Bevezető	6
1. Elméleti bevezető	7
1.1. A képfeldolgozás elméleti háttere	7
1.2. Számítógépes látás	8
1.3. OpenCV	8
1.3.1. Visual Studio 2010	10
1.4. Színterek	10
1.4.1. RGB színtér	10
1.4.2. HSV színtér	11
1.5. Küszöbözés	12
1.6. Gauss-féle elmosás	12
1.7. Morfológia	12
1.7.1. Dilatáció és erózió	13
1.7.2. Nyitás, zárás	14
1.8. Hough-transzformáció	15
2. A Mitsubishi RV-2F-Q robotkar és irányítórendszerének felépítése	18
2.1. A rendszer elemei	18
2.1.1. Robotkar	18
2.1.2. PLC	19
2.1.3. Teljesítményelektronika	22
2.1.4. Betanító pult	22
2.1.5. Pneumatika	23
2.1.6. Személyi számítógép	23
2.2. Fejlesztői környezet	24
2.2.1. GX Works2	24

TARTALOMJEGYZÉK

2.2.2. RT ToolBox2	25
2.2.3. Melfa Basic 5	25
3. A képfeldolgozás eredményének továbbítása	28
3.1. MODBUS kommunikáció	28
3.1.1. Az általam használt Modbus TCP/IP protokoll	29
3.1.2. PLC program	29
3.1.3. Libmodbus	30
3.1.4. Tesztelés	32
3.2. FTP kommunikáció	34
3.3. Kommunikáció mikrokontroller segítségével	35
3.4. Összefoglalás	36
4. Megvalósítás	37
4.1. Alkalmazás ismertetése	37
4.1.1. Palettázás	37
4.2. Alkalmazott optikai eszköz	39
4.2.1. Követelmények	39
4.2.2. A kamera elhelyezése	40
4.3. A képfeldolgozó program	41
4.3.1. A képfeldolgozás lépései	41
4.3.2. A képfeldolgozó alkalmazás által küldött adat	43
4.3.3. Az arány mérése	44
4.3.4. Kalibrációs program	45
4.4. Robotprogram	47
5. Összefoglalás, értékelés	48
Köszönetnyilvánítás	50
Ábrák jegyzéke	52
Irodalomjegyzék	54
Függelék	55
F.1. CD melléklet tartalma	55
F.2. PLC Modbus szerver létraprogramja	56

HALLGATÓI NYILATKOZAT

Alulírott *Széplaki Ádám Zoltán*, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. minden olyan részt, amelyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettettem, egyértelműen, a forrás megadásával megjelöltettem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző, cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelté után válik hozzáférhetővé.

Budapest, 2015. május 24.

Széplaki Ádám Zoltán

hallgató

Kivonat

Az ipari robotika korunk egyik leggyorsabban fejlődő területe. Ennek oka, hogy a robotok egyre több olyan helyen helyettesítik az embert, ahol egy általános, monoton feladatot kell megvalósítani (csavarozás, vágás műveletek, palettázás...). A robotok precizitásával, sebességével, valamint munkabírásával nem versenyezhet a humán munkaerő.

Az automatizált rendszerek terjedésével megnövekedett az igény olyan megoldások implementálására, amelyek valamely a feladat megvalósításának szempontjából hasznos, nemely esetben nélkülözhetetlen emberi tulajdonság alapján (mint például érzékelés, látás...) bővíti az adott architektúra funkcióit.

A számítógépes látás segítségével támogathatjuk egy munkafolyamat elvégzését, de sok esetben fel is válthatja a korábbi technológiai megoldásokat.

Dolgozatom a két említett terüettel foglalkozik. Egy RV-2F-Q Mitsubishi robotkar irányításának alapjait helyeztem számítógépes képfeldolgozás eredményeire. A robotkar a számítógépen futó szoftver által kiértékelt eredmények alapján korongokat mozgat a kamera látószögében lévő terület tetszőleges pontjáról.

Összetett feladatról van szó, amelynek megoldásához ismerni kell a robotkar és irányítórendszerének környezetét, a képi információ kiértékelésének folyamatát, valamint azokat a kommunikációs lehetőségeket, amelyek a rendszer egyes elemei között valósítják meg az információcserét.

Abstract

Robotics is one of the most quickly developing engineering field in our age. The reason is that the robots may replace humans especially in repetitive tasks (e.g.: painting, assembly, etc.). Humans are no match for the precision, speed and working capacity of industrial robots.

With the spread of automation systems a higher autonomy is expected of robotized cells including sensing capacities similar to those of human workers. In a high number of applications this means machine vision.

My thesis deals with the application of machine vision in robot control to solve a sorting problem. The control program of a RV-2F-Q Mitsubishi robot arm uses the results of an image processing software run on a PC. The thesis presents the robot controller, the steps of the image processing and the communication between the robot CPU and the PC.

Bevezető

Számos robotikai alkalmazásban az irányítórendszer képi információt is felhasznál a robot mozgatásához. A képfeldolgozásból eredő információ a megfogni kívánt tárgy pozíciója. Ezen adaton/adatokon alapuló robotirányítási alkalmazás kifejlesztése szolgáltatja szakdolgozatom központi témáját. A rendelkezésre álló Mitsubishi RV-2F-Q robotkar a tanév elején érkezett a tanszékre, az *Önálló laboratórium* c. tárgy keretein belül részt vett a telepítésében, valamint az első működő alkalmazások megírásában.

Feladatom, hogy a robotkar kamera képe alapján munkadarabokat mozgasszon véletlenszerű pozíciókból egy fixen megadott tárolóba, mindezt egy már megvalósult projekt támogató folyamataként implementálva.

A megvalósítás lépéseinak követéséhez szükség van az elméleti alapok bemutatására, ezért az 1. és 2. fejezetben szeretném az olvasóval megismertetni a robotkar és irányítórendszerének fontosabb elemeit, majd a képi információ meghatározásához szükséges alapvető fogalmakat.

Bár fizikailag össze van kötve a számítógép és az irányítórendszer, ennek eredménye, hogy a PC-n létezik megfelelő platform programkód írására, valamint feltöltésére, de minden egy fejlesztők számára kötött program keretein belül. E tekintetben a számítógépen futó egyéb alkalmazások kommunikáció szempontjából elkülönülnek a rendszer más eszközeitől. Ezért a 3. fejezetben taglalom azokat a lehetőségeket és megvalósításuk módját, amelyek segítségével az alkalmazásunk szempontjából tetszőlegesen kezelhetjük az adatcserét.

A 4. fejezetben futólag bemutatom az *Önálló laboratórium* tárgy keretein belül a robotkaron elkészített projektem, majd részletesen elemzem a szakdolgozat keretében megvalósított, ehhez kapcsolódó feladatomat.

Az utolsó fejezetben összefoglalom a program működését és leírom tapasztalataimat, valamint kitérek a fejlesztési lehetőségekre is.

Jelen dokumentum írásakor törekedtem arra, hogy megfelelő arányban legyen az általánosabbnak nevezhető elmélet, valamint a feladatspecifikus gyakorlat bemutatása.

1. fejezet

Elméleti bevezető

1.1. A képfeldolgozás elméleti háttere

A képfeldolgozást tekinthetjük a jelfeldolgozás egy változatának, ahol számítógép segítségével elemezzük és manipuláljuk, valamint kiértékeljük a kívánt adatot. [24] A legtöbb képfeldolgozó művelet megfeleltethető a jelfeldolgozásban megszokott eljárásokkal, azzal a kitételel, hogy a képet úgy kezeljük, mint egy olyan jelet, amely két dimenzióval rendelkezik. [17]

A folyamat általában három lépésből áll [16]:

1. optikai szkenner által küldött, vagy közvetlenül elérhető digitális kép importálása,
2. a kép analízise, manipulálása valamilyen módon. Ez a lépés magában foglalhatja bemenő adat javítását, tömörítését, valamint akár olyan alakzatok, objektumok keresését, amelyek az emberi szem számára láthatatlanok.
3. Az eredmény kiértékelése, amely lehet a bemenő kép módosított változata, vagy információk annak bizonyos tulajdonságai alapján.

Ipari robot vezérlése révén fontos tényező a teljes folyamat ciklusideje, amely meghatározza a termelékenységet. Tehát mindenmellett, hogy a kiértékelést minél rövidebb időszakasz alatt kell megvalósítani, annak minősége, megbízhatósága is fontos tényező. Célom tehát az egyes algoritmusok feldolgozási sebességének, és az eredmény minőségének növelése.

1.2. Számítógépes látás

A gépi látás az emberi látás bizonyos funkcióit valósítja meg. Ezek elsősorban a képi tartalom értelmezésére irányulnak: a látott képből következtet az objektumok alakjára, az objektumok térbeli elhelyezkedésére, egymáshoz való viszonyára, illetve több, időben egymást követő képből a mozgásra. Ahhoz, hogy a robotkar a térben tudjon tájékozódni, elkerülhetetlen a gépi látás beágyazása az irányítórendszerébe. Érdemes megemlíteni, hogy bár a képfeldolgozási műveleteknek rendkívül nagy az erőforrásigényük [21], felhasználási területei rendkívül széles tartományon mozognak, ezekre néhány példa:

- *irányítási folyamatok*: ipari robot,
- *biztonsági rendszerek*: mozgásdetektálás,
- *navigáció*: mobil robotok, autonóm járművek,
- *minőségellenőrzés*: roncsolásmentes anyagvizsgálatok,
- *statisztika készítése*: például forgalomszámlálás.

Az általam megvalósított feladat az irányítási folyamatok csoportba sorolható, amely során a robotkar által felvett korongok pozícióját határoztam meg a gépi látás segítségével. A korongokat szükség esetén színük alapján meg tudjuk különböztetni egymástól, a számunkra szükséges elemeket képesek vagyunk kiválogatni (optikai kiválasztás).

1.3. OpenCV

A rendszer a kamera által küldött képek sorát valós időben dolgozza fel az OpenCV (Open Source Computer Vision Library - Nyílt Forráskódú Számítógépes Látás Könyvtár) segítségével.



1.1. ábra. *OpenCV logó[15]*.

Az Intel által fejlesztett OpenCV elsődlegesen valósidejű képfeldolgozási eljárások implementálására lett kifejlesztve. 2000-ben lett nyílt forráskódú a béta verziója, 2008-ban átvette fejlesztését a Willow Garage. A könyvtár szabadon letölthető és

felhasználható a BSD licenc keretein belül. A gyakorlatban az iparban és tudományos fejlesztésekben egyaránt használják. [15]

Ismertebb támogatott nyelvek például C, C++, Python, Java. [2]

Felhasználási lehetőségei igen sokrétűek: több száz optimalizált algoritmust kínál [24]. Beláthatjuk, hogy ezen algoritmusok segítségével a gépi látás definíciójához tartozó feladatok könnyebben megoldhatók.

A teljes OpenCV könyvtár részletes bemutatása egy külön szakdolgozat témáját is szolgáltathatná, ezért most a teljesség igénye nélkül tekintsük át, hogy milyen, a feladatom számára használható jellemzői és alkalmazásai vannak a könyvtárnak [15]:

- alap adatstruktúrák,
- mátrixok, vektorok,
- dinamikus adatstruktúrák,
- kép és videó input/output,
- beolvassás fájlból (kép vagy videó) és kameráról,
- kiírási lehetőség képként vagy videóként,
- előfeldolgozás,
- színkonverzió,
- morfológiai műveletek,
- Hough-transzformáció,
- kép és videó megjelenítés,
- egyenes, kör, poligon, szöveg rajzolása,
- billentyűzet és egérkezelés.

Választásom oka, hogy az OpenCV a jelenleg elérhető leggyorsabb alternatíva gépi látás terén. Hivatalos weboldala [2] részletes dokumentációt, valamint több száz példaprogramot tartalmaz a képfeldolgozással kapcsolatban.

Érdemes arról is szót ejteni, hogy a nagymértékű felhasználói közösség (nagyjából 47 ezer ember, valamint több mint 9 millió letöltés) által létrehozott tudásbázis nagymértékben segíti az összetett projektek megvalósítását. Hátrányként megemlíthető, hogy az OpenCV-nek nincs beépített fejlesztőkörnyezete.

1.3.1. Visual Studio 2010

A számítógépen futó programot C++ nyelven írtam meg, a címben említett fejlesztőkörnyezetben, mivel a felhasznált OpenCV könyvtárnak nincs erre a célra szolgáló beépített platformja. Keretet ad a program elkészítéséhez, a kód szerkesztő ablakban megírt sorok automatikusan ellenőrizhetők, valamint futtathatók.

A Visual Studio-t a Microsoft fejleszti, több programozási nyelvet támogat, melyek repertoárja évről évre bővül. 2010 áprilisában jelent meg a Visual Studio 2010-es verziója. Integráltan támogatja a Silverlight, Windows 7 platformokra való fejlesztést is, azonban a mobil eszközök esetén már nem támogatja a Windows Phone 7-es verziót régebbi platformon való programozást. [18]

Támogatott programozási nyelvek:

- Visual Basic .NET
- Visual C++ .NET
- Visual C# .NET 4.0
- Visual F# .NET

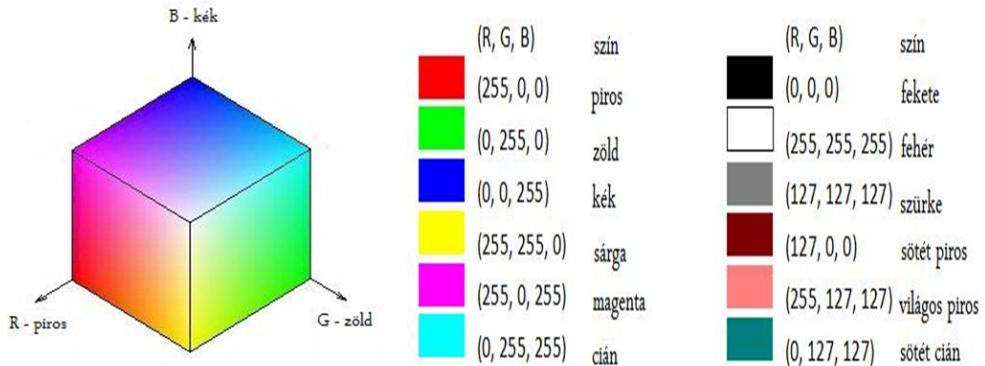
1.4. Színterek

A színtér virtuális térbeli koordináta-rendszer, amely színek ábrázolására használható. Az egyes színek tulajdonságait a rendszerbeli koordinátái fejezik ki. Az ábrázolható színek valamilyen szabály szerint kerülnek elhelyezésre, például az alapján, hogy a színtér alapszíneinek milyen arányú keverésével állíthatók elő. [19]

Bár a gyakorlatban számos színtér létezik (RGB rendszer komplementer színeire támaszkodó CMY, HEX, HSL...), feladatom során az RGB és HSV [6] színterekkel foglalkoztam.

1.4.1. RGB színtér

Az RGB színtér egy olyan additív színmodell, ami piros (Red), zöld (Green) és kék (Blue) színek különböző mértékű keverésével határozza meg a különböző színeket (1.2. ábra). Elsődlegesen elektronikai eszközök és számítástechnika terén alkalmazzák (képernyők, kijelzők, projektor, érzékelők). Alap komponenseinek értéke 0-tól 255-ig vehet fel értéket, ami 256 árnyalatnak felel meg. [12]

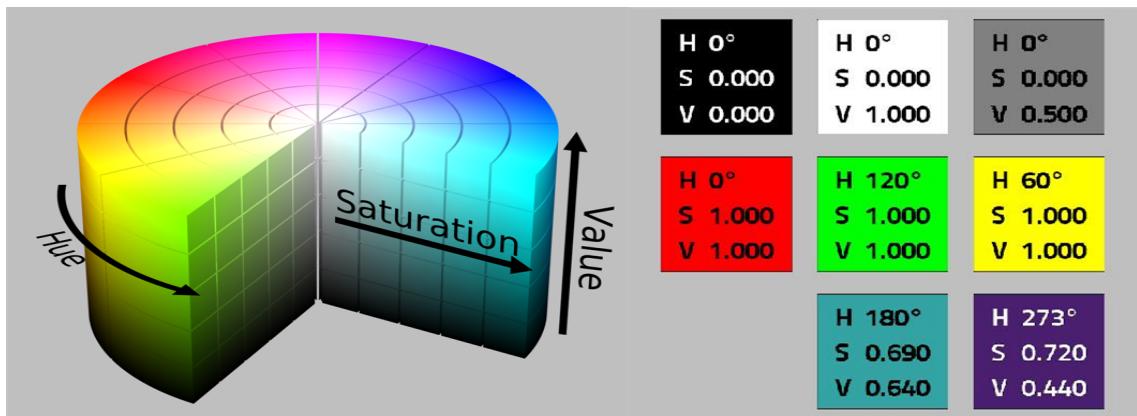


1.2. ábra. RGB színtér ábrázolása, valamint fontosabb színek értékei.

1.4.2. HSV színtér

A RGB színtérből képezhető HSV-t (Hue-színárnyalat, Saturation-színtelítettség, Value-világosság) az 1970-es években kezdték fejleszteni számítógépes grafikai alkalmazásokhoz. Ennek eredményeként manapság a képteret elterjedten használják képszerkesztő programokban, képelemzésnél, valamint számítógépes látásnál. Ez utóbbitnak oka, hogy az RGB-hez képes objektum detektálásához jobban megfelel, mivel a színárnyalat külön van választva a színtelítettségtől, és a világosságtól (1.3. ábra). Tehát kevésbé függ a szűrés eredménye a tárgy felületének színtartománybeli egyenetlenségtől, valamint a megvilágítástól. [12] [20]

Az általánosan használt HSV tartományok az alábbiak: H=0-360°, S=0-100%, V=0-100%, míg OpenCV esetében ezek az értékek: H=0-180, S=0-255, V=0-255.



1.3. ábra. HSV színtér ábrázolása, valamint fontosabb színek értékei. [6]

1.5. Küszöbözés

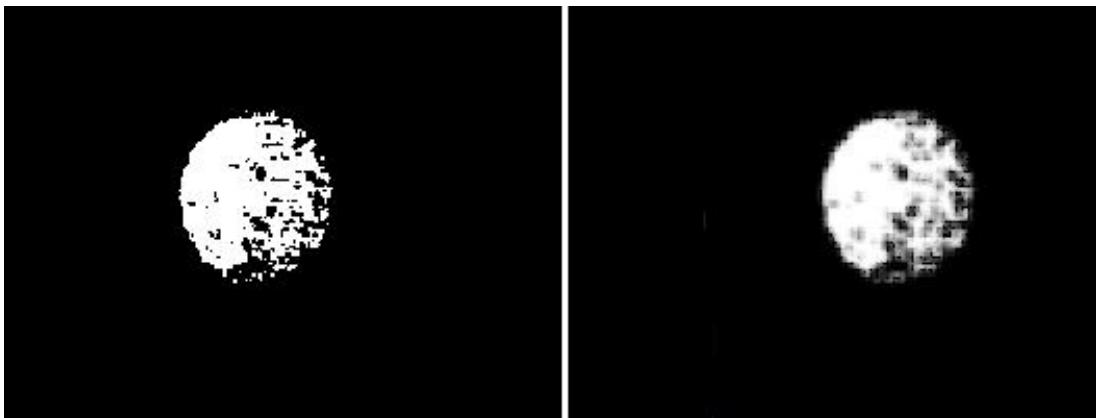
Bináris képet létrehozó operátor segítségével a művelet elvégzése után a kimeneti kép bármely képpontja csak két érték valamelyikét veheti fel [24].

A feladat megvalósításánál egy olyan függvényt alkalmaztam, amely az előre meghatározott maximális és minimális színértékek alapján hozza létre a bináris képet, tehát amely képpont a két érték között található fehér színt kap, amelyik azon kívül esik feketét. Tehát szűrés után nincsenek színárnyalatok.

1.6. Gauss-féle elmosás

Az általános átlagoló szűrő a szomszédos értékekkel átlagolja a vizsgált pixelt. Ennek hatása, hogy a kép elmosott lesz.

Gauss-féle elmosást sűrűn használják képszerkesztésben, funkcióját tekintve az átlagoló szűrés módosított változata. Eredmény szempontjából elmondható, hogy jól szűri a zajt, valamint csökkenti a kép részletességét, mégpedig amiatt, hogy a kernel¹ a Gauss-függvény szerint épül fel, vagyis a közelebbi képpontok nagyobb súlyjal tolják el az átlagot. Emiatt gyakran alkalmazzák előfeldolgozásra. [24] [11]



1.4. ábra. Gauss elmosás hatása.

1.7. Morfológia

Az adott bemeneti kép sok esetben zajos, emiatt a vizsgálni kívánt objektum nehézen azonosítható. A problémára létezik megoldás, morfológiai képfeldolgozásokkal hivatkozhatunk rá. Bár léteznek morfológiai operátorok, melyek szürkeárnyalatos,

¹Képfeldolgozás közben gyakran konvolváljuk a képet egy nxn méretű mátrixszal. Ezt a mátrixot nevezzük kernelnek. $g'(x, y) = g(x, y) * k(n, m)$, ahol az eredeti kép g , módosított kép g' , k a kernel, $*$ pedig a konvolúció.

valamint színes képeknél is használhatóak, feladatom kapcsán csak bináris képekkel való alkalmazásukat mutatom be. Általánosan felhasználható zaj eltávolítására, el-különíthető elemek szétválasztására, valamint ennek ellenére, vagyis az eljárás során tévesen megkülönböztetett objektumok integrálására [24].

Munkám során kétféle hiba került elő:

1. *a bináris képen a céltárgytól távol eső pixelek fehér színűek:* nem tartoznak a koronghoz a képpontok mégis a szűrési tartományon belül esnek,
2. *a vizsgálandó objektumhoz tartozó képpontok közül előfordulnak fekete színűek:* bár a tárgy egésze az emberi szem számára egyértelműen a szűrt színtartományon belül esik, mégis a pixelek szintjén lehetnek eltérések.

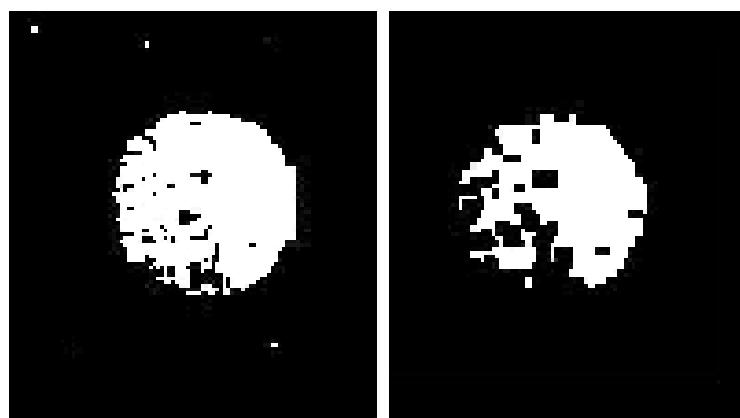
Ezen zajok kiküszöbölése, és ezáltal a kép javítása céljából használtam az alábbi operátorokat.

1.7.1. Dilatáció és erózió

Morfológiai operátorok alapelemei. Általában olyan esetekben szokás használni, amikor kismértékű zajt szeretnénk kiküszöbölni [7].

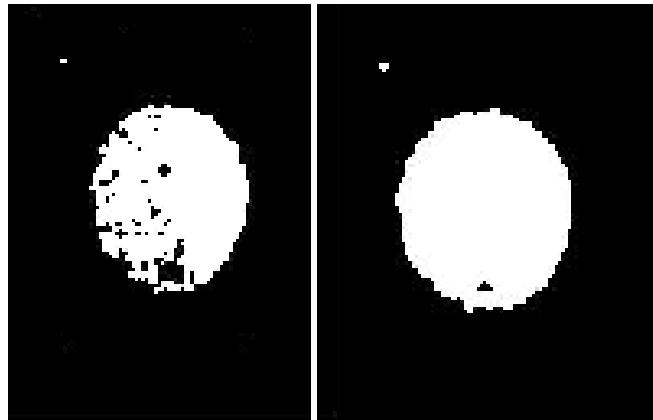
A műveletek az adott képpont környezetén belüli pixeleket vizsgálják. Amennyiben értékük eltér a lokális központban lévő képponttól, helyükre behelyettesíti annak értékét. Bináris képek esetén ez fekete, vagy fehér színértéket jelent. Az, hogy milyen mértékű környezetet vizsgálunk, az alkalmazott kernel mérete adja.

A gyakorlatot tekintve az erózió lényege, hogy a képen látható fekete területek méretét növeljük, ezzel kiszűrhetjük azokat a hibás pixeleket, amik nem tartoznak a megfigyelt objektumhoz. Az erózió alkalmas a zaj eltávolítására, és különböző elemek egymástól való elkülönítésére.



1.5. ábra. Erózió hatása.

A dilatáció hatását tekintve az erózió ellentétes művelete, tehát a bináris képen a fehér képpontokat sokszorosítjuk környezetükben. A dilatáció alkalmas az objektumon támadt lyukak befoltozására.



1.6. ábra. *Dilatáció hatása.*

1.7.2. Nyitás, zárás

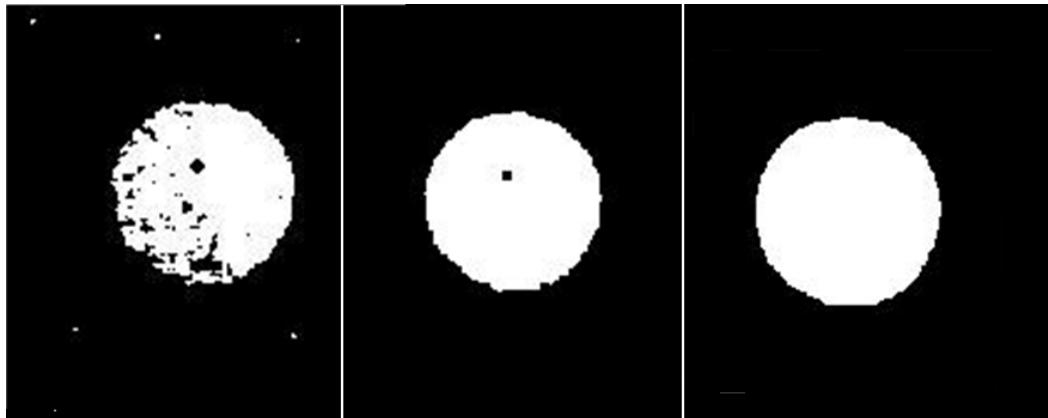
Ha az eróziót és a dilatációt kombináljuk, akkor különböző transzformációkat tudunk megvalósítani [23].

Meg kell említenem, a két művelet nem inverze egymásnak, így a dilatációt követő erózió megegyező kernel mérettel, illetve a műveletek fordított sorrendje nem adja vissza tökéletesen az alaphalmazt [5]. Az eltérés feladatom során figyelmen kívül hagyható mind mértéke, mind pedig amiatt a tény miatt, hogy egy kör középpontjának koordinátáit nem befolyásolja annak kerületváltozása. Ellenben a feldolgozott képen az objektum kitöltöttsége növekszik, megfelelő sorrendű alkalmazásuk esetén a fekete és fehér zajok eltűnnék.

A nyitást úgy valósítjuk meg, hogy eróziót végzünk a képen és azután dilatációt. A háttérben megjelenő kis fehér felületek eltüntetésére alkalmas. Feladatom során az efféle hibák elenyésző mértékben jelentek meg a homogén háttér miatt.

A zárás egy dilatációnak és egy azt követő erózióknak felel meg. Ezzel a transzformációval a kis fekete felületeket tüntetjük el a vizsgált objektumról, tehát a zárást arra használtam, hogy megnöveljem annak lehetőségét, hogy a vizsgált objektum összefüggő fehér felületű legyen.

A 1.5. és 1.6. ábrákon bemutatott képek a feladat megvalósítása során készültek. Következtethető belőlük, hogy a zárás transzformációt kellett alkalmaznom, mivel első lépésben tökéletesíti a kört, második lépésben pedig képes eltüntetni a háttérzajt.



1.7. ábra. Zárás művelete egyszer, valamint többször ugyanazon a forráson alkalmazva.

1.8. Hough-transzformáció

Az alfejezet lényege, hogy bemutassam az algoritmust, amellyel detektálhatjuk a kívánt objektumot, amely munkám során a színes korongokat jelenti.

A szűrt, valamint morfológiai operátorokkal feldolgozott bináris képen előáll a számunkra érdekes pontok halmaza. Bár, amennyiben megfelelő volt az előfeldolgozás az ember számára egyértelmű a megfeleltetés a tárggyal, de a számítógépnek ez már kevésbé egyszerű feladat. Egy korong felülnézetből szabályos körként látszódik. Ezt a kört találja meg a kimeneti képen egy körkereső algoritmus.

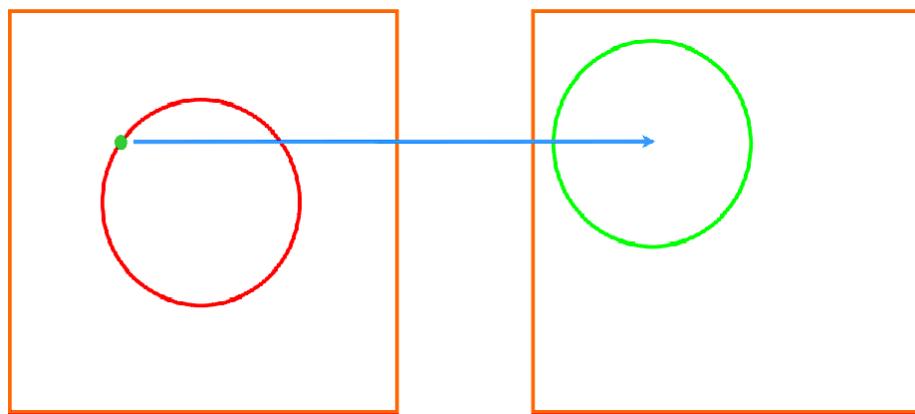
A Hough-transzformáció néhány paraméterrel leírható alakzatok detektálására alkalmas, mint például egyenes, kör, vagy ellipszis. Ezen tulajdonsága miatt széles körben használják, mint például szemmozgás követő rendszereknél, légi felvételek kiértékelésekor, vagy mint esetében optikai adat alapján történő irányításnál [10].

Hátránya, hogy bonyolultabb alakzatok esetén már nem hatékony, valamint érzékeny a zajos bemeneti képre, és többnyire csak kisebb felbontás mellett alkalmazható. Utóbbi tetszőlegesen állítható az alkalmazásomban, a fejezetben említett zavarsszűrés, valamint képjavítás hatására a transzformáció bemenete csak minimálisan zajos, valamint ahogy már említettem köröket, tehát egyszerű alakzatokat detektál a program, így az algoritmus hátrányai a projektem során nem kerülnek előtérbe. Ezzel szemben előnye, hogy gyorsan értékeli ki az eredményt, valamint, hogy létezik rá beépített OpenCV függvény.

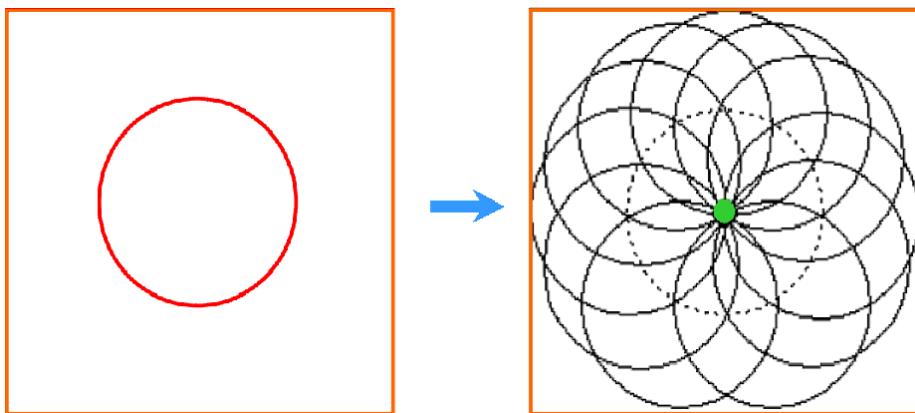
A módszer leírását kezdjük az eredeti Hough-transzformáció megértésével, amelyet egyenesek megtalálására használtak. A bináris képtérben lévő 1-es pontokat

megfeleltetjük egy szinusz görbének a Hough térben. Ebben a térben minél több görbe érintkezik azonos metszésponton, annál nagyobb a valószínűsége, hogy ezen görbék képtérben lévő reprezentánsai – az (él)pontok – egy egyenest alkotnak. Vagyis a Hough-tér küszöbölésével megkapjuk a képtér egyenesait.

Amennyiben előre meghatározott sugarú kört keresünk, a paraméter-tér 2 dimenziós lesz. Az 1.8.. ábrán látható, hogy egy él-kép minden egyes pontja egy, a potenciális középpontokat tartalmazó körnek felel meg a Hough-térben. A vizsgált kör középpontja nem lesz más, mint a körök metszéspontja által meghatározott maximumhely a Hough-térben (1.9. ábra) [22].



1.8. ábra. Kör pontjának leképezése a paraméter-térbe. [22]



1.9. ábra. Kör középpontjának meghatározása Hough-transzformáció segítségével. [22]

A feladatomban felhasznált algoritmus ettől annyiban tér el, hogy nem állandó r sugarú körökre futtatjuk le, vagyis a paramétertér 3 dimenziós lesz. A bináris képen a megfigyelt pontok 1-es, míg a vizsgálat szempontjából lényegtelen háttérrel alkotó pixelek 0-s értékűek. A bejárás során 1 értékű pixelt találva jóval nagyobb számítási igény áll elő, mint előre meghatározott sugarú körök esetében.

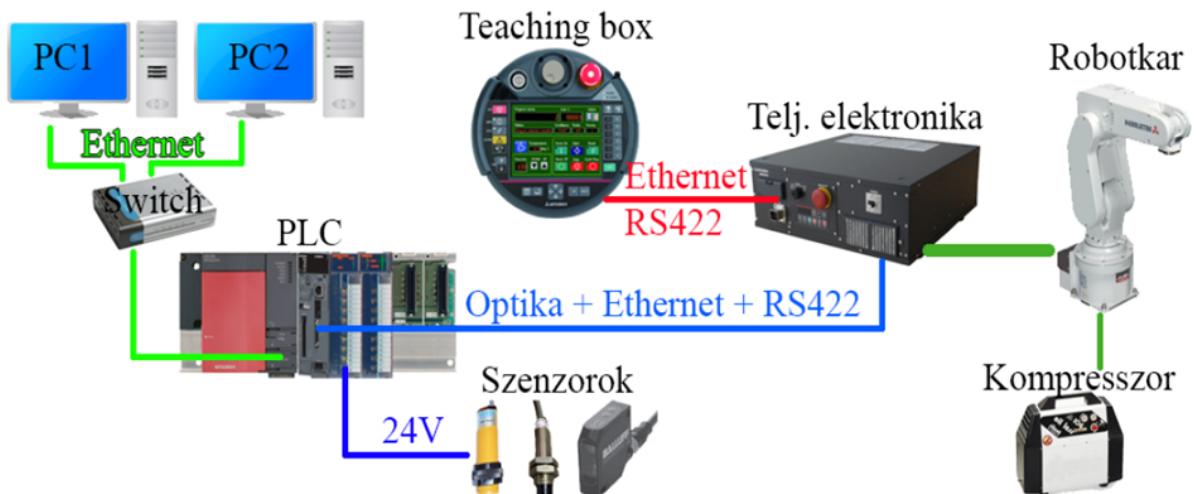
Az összes lehetséges r sugárparaméterre meg kell határoznunk a potenciális körök középpontjait. A számítási idő csökkenthető, ha nagyjából ismerjük a keresett objektum méretét. Ekkor a keresés korlátozódhat egy előre definiált r_{min} és r_{max} tartományon belül eső körökre.

2. fejezet

A Mitsubishi RV-2F-Q robotkar és irányítórendszerének felépítése

2.1. A rendszer elemei

Összetett architektúra révén ahhoz, hogy egy komplex feladatot meg tudjunk valósítani, amelyben minden elem részt vesz, szükséges a részegységek ismerete. Az alábbiakban a 2.1. ábrán látható struktúra elemeinek bemutatása következik.



2.1. ábra. Robotkar és irányítórendszere. [4]

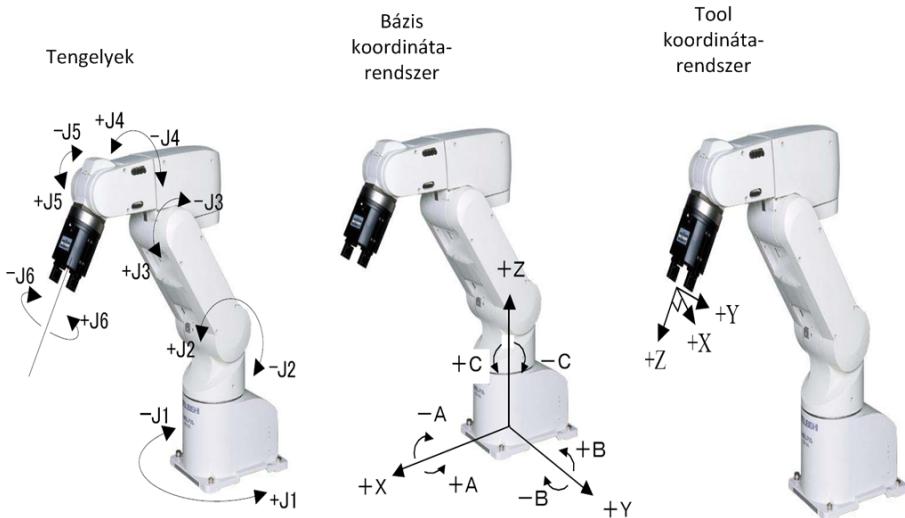
2.1.1. Robotkar

A függőleges (Elbow) felépítésű robotkar típusa Mitsubishi RV-2F-Q. Összesen 6 tengellyel rendelkezik. Ezekre a csuklótengelyekre alulról felfelé számozva hivatkozhatunk (J1, J2, ..., J6). A J2, J3, valamint J5 csuklók rendelkeznek fékkal. A tengelyek maximális sebessége eltér egymástól. A kar maximális teherbírása 2 kilogramm.

Az egyes tengelyek meghajtását AC motorok végzik hullámhajtóműs áttétellel, amik abszolút enkóderrel vannak felszerelve. A robot kikapcsolt állapota esetén az enkóder értéket a beépített akkumulátor tárolja, melynek lemerülése esetén a robotot újra kell kalibrálni. A robotkar megfelelő csuklóállások mellett maximum 0,504 méter kitérésű. [14]

A robot helyzetének megadási módjai

A robotkar pozíciójának meghatározását többféle módon végezhetjük (2.2. ábra). Hivatkozhatunk az egyes csuklók elfordulására, vagy megadhatjuk a keret relatív pozíóját és orientációját a térben, és a robot mozgásáért felelős processzor a hátterben kiszámolja a megfelelő csuklóállásokat. Annak érdekében, hogy az adott pozíció betanítása a legkényelmesebb módon történhessen, választhatunk *bázis/XYZ* koordinátarendszt (ekkor az egymásra merőleges irányvektorok a robot aljának középpontjához vannak rögzítve), vagy a végberendezéshez illesztett, úgynevezett *tool* koordinátarendszt.



2.2. ábra. Mozgási irányok, melyek paraméterezésével megadható a robotkar pozíciója, orientációja. [14]

2.1.2. PLC

A PLC moduljai a System Q moduláris PLC rendszer/család tagjai. A 8 slotos alaplaphoz csatlakoznak: tápegység modul, processzor kártya beépített ethernet csatlakozóval, egy Motion CPU, valamint egy 16 digitális vonallal rendelkező bemeneti és egy kimeneti modul [8].



2.3. ábra. System Q termékcshaládhoz tartozó PLC.

Alaplap: QD38DB

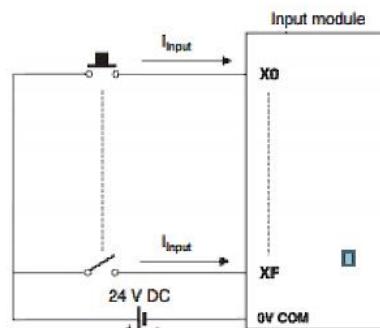
A PLC alapegysége. Funkciója: felületet szolgáltat a modulok szerelésére, összekötésére és nagy sebességű kommunikációt biztosít az egyes modulok között. Az alaplapon elhelyezhető modulok: CPU modulok, tápmódul, bemeneti modul, kimeneti modul, hálózati modul és speciális funkciójú modulok.

Tápegység: Q63P

Bemenete 24 VDC, míg kimenete 5 VDC / 6 A. Tápellátást biztosít az összes többi elemnek a hátlapon. A PLC előlapján LED-es indikátor található a működési állapotáról.

QX80

16 darab digitális bemenete van, amik optocsatolós galvanikus leválasztással rendelkeznek. A bemeneti jelszint 24 VDC (4mA), ezt a jelszintet alakítja át digitális jellé. Led-es indikátorok jelzik a bemeneti állapotokat negatív logikával (2.4. ábra).



2.4. ábra. Digitális bemenet.

QY80

16 darab tranzisztoros kimenet. A kimeneti jelszint 20V.

Q03UDVCPU (PLC CPU)

A PLC processzorkártyája, kezeli a digitális kimeneteket/bemeneteket, a hálózati kártyát, és utasításokat küldhet a robotvezérlő CPU-nak.

Jellemzői:

- univerzális Q CPU modul,
- 30k memória,
- 20ns végrehajtási idő utasításonként,
- Ethernet és USB csatlakozás,
- iQ rendszerrel való kompatibilitás,
- összekapcsolható a motion CPU-val,
- rendkívül gyors bit feldolgozás: 9.5ns,
- nagy sebességű adathozzáférés,
- szó (16bit) memóriaszervezés.

Q172DRCPU (Motion CPU)

A robotvezérlésért felelős feldolgozó egység.

Jellemzők:

- egy rendszeren belül 3 robotot is képes irányítani,
- rugalmas kezelhetőség a megosztott iQ hálózaton, I/O, valamint az intelligens modulokon keresztül,
- gyors ciklusidő az inter-CPU megosztott/közös memória buszán,
- bites memóriaszervezés.

Közös memóriaterület

A két említett CPU kommunikációja egy megosztott memóriaterület igénybevételével valósul meg. Ezek a közös regiszterek minden két oldal számára elérhetők, azonban címzésük eltérő (2.5. ábra), mivel a processzorok memóriaszervezése eltér egymástól.

2.1. A RENDSZER ELEMEI

PLC output address		Robot adress	PLC device TO robot		PLC device FROM robot		Robot adress	PLC input address	
Word address	Bit address	1st Robot inputs					1st Robot outputs	Word address	Bit address
U3E0\G10000	U3E0\G10000.0	M_in[10000] BIT	USED	Name	Name	USED	M_out[10000]	U3E1\G10000	U3E1\G10000.0
	U3E0\G10000.1	M_in[10001] BIT					M_out[10001]		U3E1\G10000.1

	U3E0\G10000.F	M_in[10015] BIT					M_out[10015]		U3E1\G10000.F
U3E0\G10001	U3E0\G10001		USED	Name	Name	USED	U3E1\G10001	U3E1\G10001	U3E1\G10001

	U3E0\G10009						U3E1\G10009		U3E1\G10009
U3E0\G10010	U3E0\G10010.0	M_in[10160]	D500.0			D1000.0	M_out[10160]	U3E1\G10010	U3E1\G10010.0
	U3E0\G10010.1	10161	D500.1			D1000.1	10161		U3E1\G10010.1
	U3E0\G10010.2	10162	D500.2			D1000.2	10162		U3E1\G10010.2
	U3E0\G10010.3	10163	D500.3			D1000.3	10163		...
	U3E0\G10010.4	10164	D500.4			D1000.4	10164		...
	U3E0\G10010.5	10165	D500.5			D1000.5	10165		...
	U3E0\G10010.6	10166	D500.6			D1000.6	10166		...
	U3E0\G10010.7	10167	D500.7			D1000.7	10167		...
	U3E0\G10010.8	10168	D500.8			D1000.8	10168		...
	U3E0\G10010.9	10169	D500.9			D1000.9	10169		...
	U3E0\G10010.A	10170	D500.A			D1000.A	10170		...
	U3E0\G10010.B	10171	D500.B			D1000.B	10171		...
	U3E0\G10010.C	10172	D500.C			D1000.C	10172		...
	U3E0\G10010.D	10173	D500.D			D1000.D	10173		...
	U3E0\G10010.E	10174	D500.E			D1000.E	10174		...
	U3E0\G10010.F	10175	D500.F			D1000.F	10175		U3E1\G10010.F
U3E0\G10011		10176	D501			D1001	10176	U3E1\G10011	
U3E0\G10012		10192	D502.0			D1002.0	10192	U3E1\G10012	

2.5. ábra. Megosztott memória címzése.

2.1.3. Teljesítményelektronika

A teljesítményelektronikai átalakítók az ipar szinte minden területén megtalálhatók, ahol szükség van az elektromos energia átalakítására (például változtatni akarjuk a feszültséget, áramát, vagy frekvenciáját). Mivel működésének részleteibe menő ismerete nem kapcsolódik az általam megvalósított feladatokhoz, ezért kizárolag felhasználói szempontból megközelítve tanulmányoztam. A teljesítményelektronika vázának kijelzőjén nyomon követhető a roboton futó program címkéje, valamint a kód éppen végrehajtott sorának száma. A kijelző mellett megtalálható kulcs segítségével tudjuk a rendszert kézi, vagy automata üzemmódba kapcsolni. Kezelés szempontjából a legfontosabb különbség, hogy kézi módban feloldja a betanító pult letiltását, míg automata módban a PC kap engedélyt robotprogram futtatására.

2.1.4. Betanító pult

Kézi vezérlés esetén a betanító pult (TB, Teaching Box) segítségével tudjuk manipulálni a robotkar térbeli helyzetét, valamint orientációját. A betanítás elkezdése előtt a *Teach* módnak kell aktívnak lennie, valamint a *Servo* kapcsolót kell benyomni. A megtalált pozíció mentését végezhetjük a pulton (*Jog* menü), vagy a számítógépen, amennyiben éppen fel vagyunk csatlakozva a robotra. Az érintőképernyőn paramé-

terek változtatásával mozgathatjuk a kart a 2.2. ábrán látható megadási lehetőségek alapján (a csuklótengelyek szögét egyenként állítjuk, vagy *XYZ* és *TOOL* koordinátarendszer tengelyei mentén mozgatjuk, vagy forgatjuk a végberendezést). Ipari kialakítás révén biztonsági okokból a pult hátulján lévő gombok egyikének (dead man button) a betanítás ideje alatt folyamatosan benyomva kell lennie.



2.6. ábra. *Betanító pult.*

2.1.5. Pneumatika

A robotkar végberendezése egy sűrített levegővel nyitható/zárható megfogó elem. Az erőkifejtéshez szükséges nyomást egy kompresszor biztosítja. A solenoid szeleptelep vezérlése gyárilag megoldott. Vezérlését végezhetjük a betanító pult *Hand* menüpontból manuálisan, valamint a 2.2.2. fejezetben ismertetett fejlesztőkörnyezeten belül robotprogram írásakor utasításként hivatkozva. A munkám megvalósítása során az egyik legnagyobb kihívást okozta, hogy a megfogó végelem minden összes 5 mm-t zár össze, a korong megfogásakor megengedett tolerancia 2 mm-re tehető, így a folyamat minden elemét precíz pontossággal kellett telepítenem.

2.1.6. Személyi számítógép

A felsorolásból nem hagyhatók ki említés szintjén a robotkar rendszeréhez kapcsolt modern számítógépek. Nagy műveletvégzési sebességüknek köszönhetően gyors programkezelést, kód fordítást, feltöltést biztosítanak a felhasználók számára.

2.2. Fejlesztői környezet

2.2.1. GX Works2

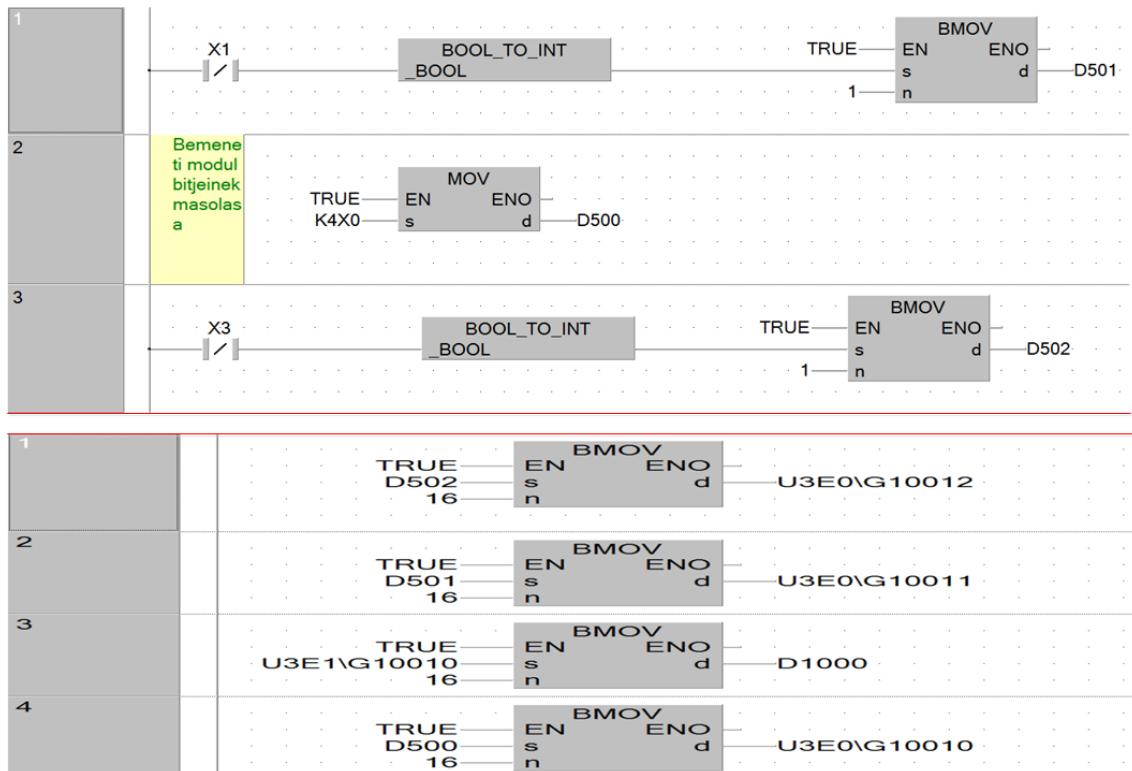
A GX Works2 program segítségével lehet programozni a PLC CPU-t. Támogatott programozási nyelvek:

- ST (strukturált programnyelv),
- SFC (sorrendi folyamatábra),
- létradiagram,
- strukturált létradiagram.

Csatlakozni tud USB-n és Etherneten keresztül is a PLC beépített hálózati modul-jához.

Adatcsere program

A processzorok közötti kommunikációt megvalósító létradiagram a 2.7. ábrán látható.



2.7. ábra. Adatküldés megvalósítása két lépésben.

A bemeneti modul meghatározott regisztereinek, valamint a jelenlegi hálózati elrendezésben a külső eszköztől (például PC) érkező információk adatcsere szempontjából megkövetelt kezelése megegyezik. Általános esetben ez említett adatok a PLC processzor számára elérhetőek, programozási feladat ezeknek a közös memóriaterületre való másolása.

2.2.2. RT ToolBox2

Keretet ad a robot mozgásáért felelős Motion CPU programozásához.

A programnak három fő állapota van:

1. *Offline üzemmód*: lokálisan írhatunk programokat, amik a PC-n a projekt mapájában tárolódnak,
2. *Online üzemmód*: a program csatlakozik a PLC-n keresztül a Motion CPU-hoz, elérhetővé, szerkeszthetővé, és futtathatóvá válnak a roboton lévő programok,
3. *Szimuláció*: a szoftver egy olyan környezetet biztosít a felhasználó számára, amely megegyezik az *Online* üzemmód valamennyi sajátosságával. Ebből kiindulva még az Offline írt programot is szükséges felmásolni a virtuális robotra, amennyiben futtatni szeretnénk. A robot viselkedése 3D Monitor segítségével szimulálható.

A *Layout* menüben lehetőségünk van mind a robot végberendezését, mind a robot környezetét (például a mérési panelt) alapvető geometriai alakzatokból álló építőelemek segítségével megépíteni. Megfelelő beállítás esetén ezek megjelennek mind *Online*, mind pedig *Szimuláció* üzemmódban is.

Debug open

Legyen szó szimulátorról, vagy a tényleges robotkarról, a felcsatlakozás után a *Debug* üzemmódban tudjuk a megírt kódsorokat lefuttatni. Az *OVRD* paraméter segítségével lehet a robot mozgásának a sebességét felülríni, tehát ha a programban például 100 százalékot állítottunk be, de a debug ablakban 10 százalék van beállítva, akkor a kettő szorzata lesz a tényleges mozgási sebesség. A *RESET* mezőben taláható gombokkal lehet a program elejére ugrani (program számláló reset) és a hibaüzeneket nyugtázni (error reset). A *Jump* címkéjű szövegdobozba írt érték segítségével tudunk a programkód megfelelő sorára ugrani.

2.2.3. Melfa Basic 5

A programozási nyelvet a Mitsubishi fejlesztette ki gyártósori robotok programozására. Egy basic típusú nyelv, melyben kommentet a ', címkét a * speciális karak-

ter jelöl. A program elejére praktikus a *ServoOn*, értelemszerűen a szervómotorok bekapcsolásának utasítását írni, valamint meghatározni az egyes csuklók maximális sebességét százalékos arányban az *OVRD* funkció segítségével. Ez a maximális sebesség J1, J2, J3... esetén külön-külön eltér. A várakozó utasítást *Dly* gyakran igénybe kell venni, mivel a megfogás előtt és után időt kell hagynunk a pneumatikus rendszernek.

Mozgató utasítások

A munkám során használt mozgató utasítások: *MOV*, valamint az *MVS*. Paraméterként egy pozícióértéket tartalmazó változót várnak, ami lehet előre betanított, de futás közben is adhatunk neki értéket. A két parancs között a különbség, hogy előbbi esetén a robot két pont között egy véletlenszerű¹ pályát ír le, míg utóbbi segítségével lineáris pályán mozoghatunk. Ez alapján elmondható, hogy kritikusabb pozíció (például asztalhoz közel, vagy két tárgy között lévő...) esetén az *MVS* utasítást érdemes használni. [14]

Memória olvasása és írása

A Melfa Basic V nyelven belül az *M_in(cím)*, valamint az *M_out(cím)* szintaktika segítségével tudunk hivatkozni a memóriakezelő műveletekre. Ennek jelentősége a már említett közös memóriaterületen való kommunikáció kezelése a robot oldaláról. [13]

Pozíciók

A megnyitott programkód sorai alatt megtalálhatjuk az általunk felvett pozíciókat. Amennyiben a robotra fel vagyunk csatlakozva az *Add* menü *GetCurrentPosition* gombjára kattintva lekérhetjük és elmenthetjük a kar jelenlegi pozíójának értékeit, azzal az egyetlen kitételel, hogy a neve *p* betűvel kell kezdődjön (így automatikusan globális változóként lesz definiálva). Azonban nem szükséges minden esetben kezdeti értéket adnunk, a program futás közben is változtathatja a pozíciók paramétereit². Az elmentett értékeket manuálisan is átírhatjuk az *Edit* menüben.

¹A pontosság kedvéért nem teljesen véletlenszerű pályáról van szó, a kiszámíthatatlannak tűnő mozgás annak következménye, hogy a robotkar csuklótengelyeinek maximális fordulási sebessége nem egyezik meg

²Ezekre a paraméterekre a kiválasztott koordinátarendszertől függően hivatkozhatunk, például position.x = 100

Megfogó végberendezés vezérlése

A sűrített levegővel működő munkahengert a *HOpen1* utasítással nyitjuk, valamint a *HClose1* parancsal zárjuk. [13]

3. fejezet

A képfeldolgozás eredményének továbbítása

A képfeldolgozást végző program a PC-n fut, így a szükséges adatot valamilyen kommunikációs csatorna segítségével el kell juttatnunk a robotkar mozgásáért felelős vezérlő egységhez (Motion CPU).

A rendelkezésre álló lehetőségek:

- MODBUS kommunikáció,
- FTP kommunikáció,
- kommunikáció a rendszerbe ágyazott mikrokontroller segítségével.

3.1. MODBUS kommunikáció

A protokollt 1979-ben kezdte fejleszteni a Modicon cég ipari automatizálási rendszerek, és programozható irányítóberendezések kommunikációjára. Mára az egyik legelterjedtebb szabványosított ipari módszer I/O információk és regiszter adatok küldésére az ipari berendezések, valamint a monitorozó állomások között. A Modbus-ra jellemző a kliens-szerver architektúra. Egy időben minden eszköz kezdeményezhet tranzakciót (szerver). A hálózat többi eleme köteles visszaküldeni a kért adatot, vagy véghezvinni a kapott utasítást. Kliens lehet bármely olyan konstrukció, amely képes információt feldolgozni, és azt kimenetén elküldeni a szervernek. A szerver meg tud címezni különálló klienseket, de képes küldeni úgynevezett broadcast üzenetet is a teljes hálózatra. [9]

3.1.1. Az általam használt Modbus TCP/IP protokoll

Alapvető tulajdonsága, hogy rajta keresztül bináris adatcsomagokat tudunk cserélni két állomás között. TCP (Transmission Control Protocol) feladata, hogy az összes elküldött adat hiba nélkül megérkezzen, az IP (Internet Protocol) pedig biztosítja, hogy az üzenet a megfelelő címzetthez jusson el. Egy összetett felépítésről van szó, amely kombinálja a fizikai hálózat (Ethernet), a hálózati szabvány (TCP/IP), és az információkezelés (Modbus) elemeit. Ezek alapján elmondható, hogy a Modbus TCP/IP üzenet egy egyszerű Modbus kommunikációval feleltethető meg, amely Ethernet TCP/IP protokollstruktúrába van csomagolva [9].

A mi esetünkben a személyi számítógép Etherneten keresztül kommunikál a System Q PLC-vel, melyre fel van töltve egy Modbus szervert megvalósító létraprogram, majd a PLC a robot mozgatását végző processzornak továbbküldi a fogadott adatot. A továbbküldés módja: közös memóriaterületre másolás, majd a Motion CPU kiolvassa azt a megfelelő címről.

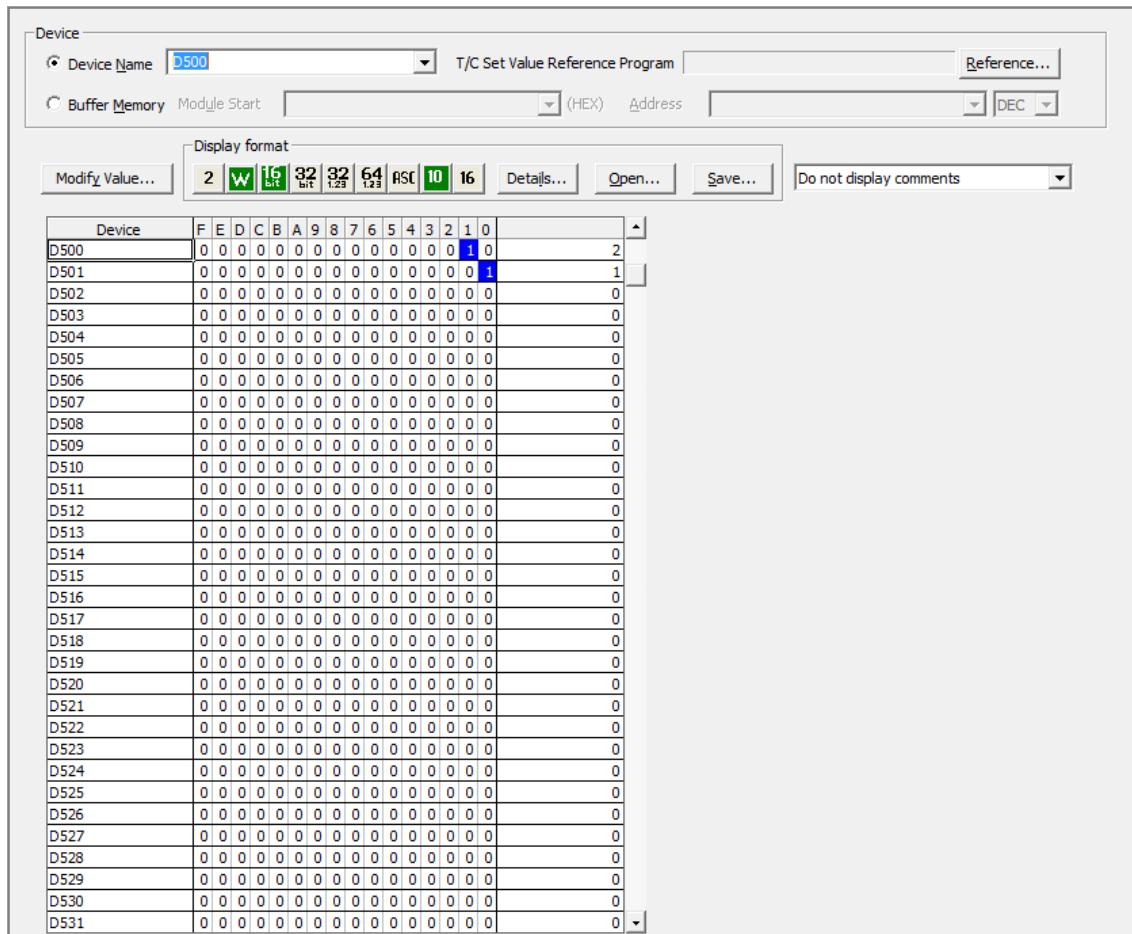
3.1.2. PLC program

A PLC CPU oldalán egy Modbus szerver található, amely fogadja a külső klienstől érkező adatokat, majd átmásolja azokat a megadott memóriaterületre. A szerver megvalósítását szolgáló létraprogram már létezett a Mitsubishi cég fejlesztése által, felhasználása során minden paramétert kellett meghatároznom (melyik regiszterbe írjon adatot, milyen címen elérhető...). Emellett egy másoló blokk van feltöltve a PLC-re, amely a bemenetként megadott címről, megadott mennyiségű adatot másol a kimenetént beállított címre. Ez a kimeneti blokk nem más, mint a Motion CPU-val közös memóriaterület.

PLC monitorozás

A GX Works 2 programon belül kiválaszthatjuk a számunkra érdekes memóriablokokat, és ezeket valós időben meg tudjuk figyelni, valamint értéküket is megváltoztathatjuk (3.1. ábra).

3.1. MODBUS KOMMUNIKÁCIÓ



3.1. ábra. *PLC monitor funkció.*

3.1.3. Libmodbus

A Libmodbus egy ingyenes szoftverkönyvtár, amely adatok küldését és fogadását hivatott megvalósítani a Modbus protokollhoz igazodva. A könyvtár C nyelven íródott és támogatja az RTU (soros), valamint TCP (Ethernet) kommunikációkat.

A Libmodbus kezelése

A könyvtár a számítógép oldali adatküldést/fogadást valósítja meg. Visual Studio 2010 fejlesztőkörnyezetben a libmodbus projektként való felvétele, majd futtatása után a forrásmappában egy „modbus.lib” valamint egy „modbus.dll” fájl generálódik. Ezeket kell beépítenünk a saját projektünkbe. A saját C vagy C++ kódunk fejlécébe a „modbus.h” header fájlra való hivatkozás által hozzáférhetünk a könyvtár elemeihez, függvényeihez. Az elvégzett lépések után az alábbi kódok segítségével márás jól működő Modbus protokoll szerinti kommunikáció áll rendelkezésünkre.

A kapcsolat felépítése

`modbus_t *modbus_new_tcp(const char *ip, int port);` a függvény inicializálja és allokálja a modbus_t struktúrát, hogy megvalósítsa a kommunikációt a Modbus TCP/IP szerverrel. Az ip argumentum meghatározza a szerver IP címét, amellyel a kliens létre akarja hozni a kapcsolatot. A port argumentum értelemszerűen a használt TCP port száma. Az argumentumként megadható MODBUS_TCP_DEFAULT_PORT az alapértelmezett 502-es portot fogja beállítani. Sikeres lefutás esetén egy a modbus_t struktúrára mutató pointerrel tér vissza a függvény. Egyébként NULL pointer a visszatérési érték és beállítja a megfelelő hibaüzenetet. A libmodbus könyvtár a felhasználó által hibásan megadott IP cím kezelésére van felkészítve.

`void modbus_free(modbus_t *ctx); void modbus_close(modbus_t *ctx);` fontos, hogy a megfelelő helyen végezzük el a szükséges memóriaterületek felszabadítását, valamint a kapcsolat bezárását.

A kommunikációt megvalósító, és a projekt során felhasznált írás és olvasást végző függvények

`int modbus_write_register(modbus_t *ctx, int addr, int value);` a ctx paraméter tárolja a távoli eszközzel való kapcsolódás adatait. Amennyiben sikeres volt a kommunikációs csatorna megnyitása az addr sorszámú tároló regiszterbe írja a value egész értékét. A függvény visszatérési értéke 1, ha sikeres volt az írás, egyébként -1, és beállítja errno¹ változót.

`int modbus_read_registers(modbus_t *ctx, int addr, int nb, uint16_t *dest);` A távoli eszkösről nb számú tároló regiszter tartalmát olvassa ki a függvény. Az olvasás eredményét elmenti a dest tömbbe szó értékekkel (16 bit). A függvény a sikeresen kiolvasott regiszterek számával tér vissza. Amennyiben ilyen nem volt a visszatérési érték -1 és beállítja a megfelelő hiba számát (errno változó). Alapvető szabály, hogy allokálunk elegendő memóriát, az eredmény dest tömbbe való elmentéséhez (legalább nb x sizeof(uint16_t)).

Az alábbi programrészlet (3.2. ábra) működése: A megfelelő porton lévő és létező IP-című eszközzel megnyitja a kommunikációs kapcsolatot, kiolvas értékeket a céleszközöből, majd számadatot küld a meghatározott regiszterbe. Végül bezárja a kapcsolatot és felszabadítja a szükséges memóriaterületeket.

¹Hibaüzenetet tartalmazó változó.

```

modbus_t *ctx;
uint16_t tab_reg[64];
int rc;
int i;

ctx = modbus_new_tcp("127.0.0.1", 1502);
if (modbus_connect(ctx) == -1) {
    fprintf(stderr, "Connection failed: %s\n", modbus_strerror(errno));
    modbus_free(ctx);
    return -1;
}

rc = modbus_read_registers(ctx, 0, 10, tab_reg);
if (rc == -1) {
    fprintf(stderr, "%s\n", modbus_strerror(errno));
    return -1;
}

for (i=0; i < rc; i++) {
    printf("reg[%d]=%d (0x%X)\n", i, tab_reg[i], tab_reg[i]);
}

int modbus_write_register(ctx, 500, 1);

modbus_close(ctx);
modbus_free(ctx);
    
```

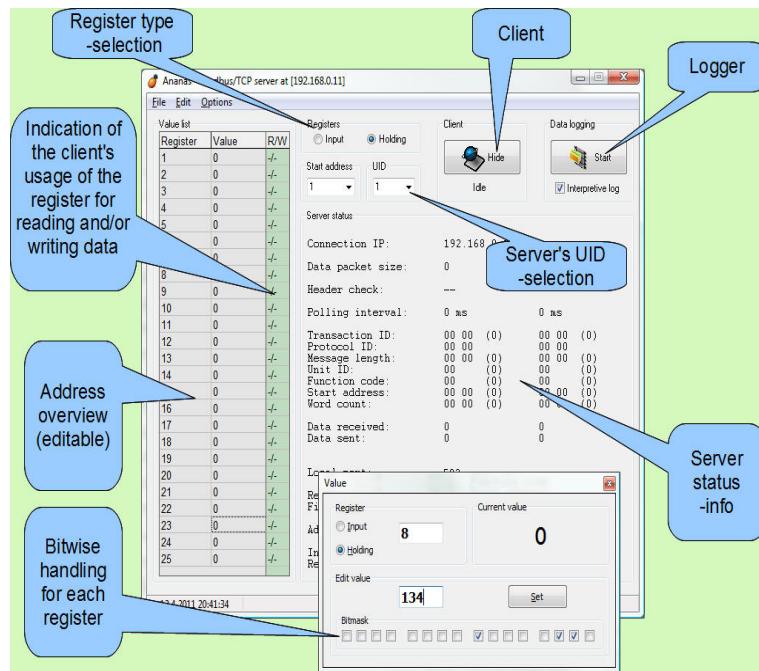
3.2. ábra. Modbus példaprogram.

3.1.4. Tesztelés

A Modbus kommunikáció tesztelésére az Ananas 1.8.1 programot használtam. Windows rendszereken futtatható, közvetlenül Windows API-t használ, nem szükséges hozzá MS Framework vagy Java Runtime Environment [1]. A program feltételezi, hogy egy Modbus regiszter előjel nélkül 16 bites egész számot, vagy 16 bites szót tartalmaz. A libmodbus függvényeinek működése alapján ez megfelel nekünk. Még mielőtt a végső PC-PLC kommunikáció letrejönne, a kapcsolat minden oldalán lévő eszközök teszteljük az Ananas segítségével. A programot először Modbus szerverként fogjuk használni, egyelőre helyettesítve a robotkar rendszerébe illeszkedő PLC-t. Az Ananas kiírja nekünk az első IP címet, amelyet a PC-n futó operációs rendszertől kap, és ezen a címen is lesz elérhető. Több IP cím is rendelkezésre áll, ezeket nem írja ki a program, de szükség esetén beállítható az alapértelmezettől eltérő érték is. A saját programunk a fent ismertetett függvények segítségével egy Modbus TCP/IP klienst hivatott megvalósítani. Sikeres beállítás esetén az Ananas, mint virtuális Modbus szerver regiszter táblázatában megjelenik az elküldött érték, valamint ezen

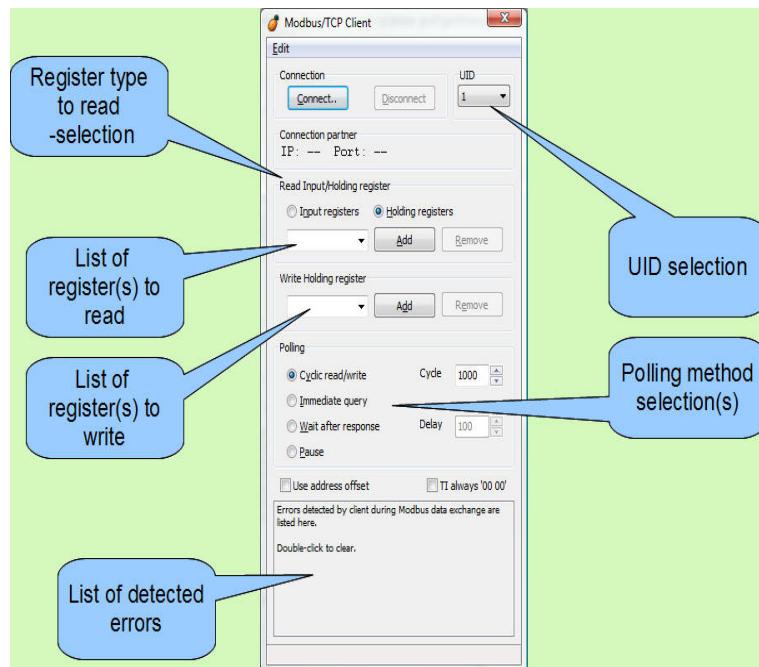
3.1. MODBUS KOMMUNIKÁCIÓ

értékek kiolvashatóak a kliens számára.



3.3. ábra. Virtuális Modbus szerver. [1]

A tesztelés második fázisa, hogy az Ananas programot kliensként futtatjuk és kapcsolódni próbálunk a PLC-hez. Ezt megtehetjük az alábbi képen látható platformon a *Connect* gombra kattintás után az IP cím valamint a port szám megadásával. Hiba esetén a kapcsolat nem jön létre, a program a detektált hibát kiírja a felhasználói felületre.



3.4. ábra. Virtuális Modbus kliens. [1]

3.2. FTP kommunikáció

A Modbus kommunikációtól eltérően nem szükséges először a PLC CPU-nak elküldeni az adatot, majd onnan egy a Motion CPU által elérhető memóriarekeszbe átmásolni azt, hanem közvetlenül létesítenénk kapcsolatot a robotkarral. A program írását az RT Toolbox 2 programmal végeztem, amely segítségével a robotra feltölthetünk egy olyan programot, amely futása közben képesek vagyunk megnyitni file-t, vagy kommunikációs csatornát. Ezt a fájlnév megadásakor kell definiálnunk a 3.1. táblázat alapján.

Többféle módszer is rendelkezésre áll, hogy miként kezeljük a kapcsolatot létesítő vonalat:

- *bemenet*: olvasás a meglévő fájlból,
- *kimenet*: új fájl létrehozása, és adat írása,
- *hozzáfűzés*: kimeneti mód, de a kiírandó információt a már létező fájl végéhez illeszti,
- *az elérési metódus elhagyható (véletlenszerű mód)* : kommunikációs vonal használatakor van rá lehetőség.

3.1. táblázat. *Melfa Basic V - Open utasítás lehetőségei.*

Fájl típus	Fájl név	Elérési metódus
File	Maximum 16 karakterrel definiálható.	bement,kimenet,hozzáfűzés
Kommunikációs vonal	COM 1...8: szabványos RS232C Ethernet	véletlenszerű Mxt utasítás

Mxt utasítás egy valósidejű külső elérésű irányító funkció Etherneten keresztül. A robot direkt módon mozdul, amennyiben megfelelő abszolút pozíció adat érkezik. [13]

A megoldást gyakorlatban is megvalósítottam. Szimulátor módban tökéletesen működött az információjácsere, azonban valós körülmények között sajnos nem alkalmazhattam, mivel a robot és annak irányítórendszerére egy előre meghatározott és jelenleg nem átalakítható módon kapcsolódik az egyetemi hálózatra. A robotra feltöltött program futás közben létrehozta a szükséges kommunikációs fájlt és fel is töltötte megfelelő adatokkal, azonban ez jelen hálózati kialakításban a számítógép oldaláról nem volt elérhető.

3.3. Kommunikáció mikrokontroller segítségével

A mozgást szabályozó hurok működhet folyamatos képi visszacsatolás alapján (visual servoing). Ebben az esetben a robotra szerelt kamera képe alapján négy bináris információt érdemes meghatározni, mégpedig hogy a megfigyelt tárgy pozíciójának koordinátái (x , és y) eltérnek-e és ha igen, milyen irányban attól a pozíciótól, amely a feladat megvalósításához szükséges (esetünkben a korongok megfogásának pozíciójától).

Felvetődött annak lehetősége, hogy a számítógépen kiértékelt adatokat Bluetooth használatával eljutassuk egy Arduino Uno mikrokontrollerhez, amely digitális kimenetein a PLC digitális bemeneteihez kapcsolódott volna. A folyamat innentől megegyező a Modbus kommunikációval, vagyis a PLC-be beérkező adatok (bár ezúttal digitális) a közös memóriaterületre másolásával elérhetőek a Motion CPU számára.

A megoldás előnye, hogy a mikrokontroller felszerelhető további áramkörökkel (pl. RTC, Led-ek, Ultrasonic Sensor, LCD Display...), amelyek tovább bővítenek a projektben rejlő lehetőségeket. Hátránya, hogy működését tekintve jóval lassabb a többi kommunikációs megvalósításhoz képest. Szükség lenne egy illesztő áramkörre, mivel a rendelkezésre álló mikrokontroller (Arduino Uno) 5V-os kimenettel rendelkezik, míg a PLC QX80 bemeneti kártyáján 24 VDC feszültségszintet vár, és ezt konvertálja digitális jellé. A késleltetés miatt a végpozíció körüli ingadozás várható, ezért szükséges lenne egy szabályzást is implementálni.

Amennyiben a megvalósítás előnyeit sikerül a rendszer egyéb eszközeivel helyettesíteni (LCD kijelző valamint RTC a PC, LED indikátorok a PLC segítségével), úgy ez a megoldás kevésbé optimális az előzőeknél. Ebből kifolyólag gyakorlati szinten minden össze egy egyszerű soros kommunikációt valósítottam meg a számítógép és a mikrokontroller között, de további fejlesztésekbe ezen a vonalon nem kezdtem.

3.4. Összefoglalás

A másik két jobbnak ígérkező lehetőség (Modbus, FTP) során milliszekundumos időkről, időkésleltetésekéről beszélünk. Az FTP közvetlen összeköttetéssel járó előnye (lehető leggyorsabb kommunikációs ciklusidő) a Modbus kommunikációval szemben jelen projekt esetében nem releváns. A rendszer elrendezését megváltoztatva a számítógépet közvetlenül a robotkarral kéne összekötni, hogy megvalósítható legyen az FTP kommunikáció.

Így jelenleg a legoptimálisabb választás a Modbus TCP/IP protokoll használata.

3.2. táblázat. Kommunikációs lehetőségek összefoglalása.

Kommunikáció típus	Előny	Hátrány
Modbus	gyors	nehezebben implementálható
FTP	gyors,könnyen programozható	jelenlegi elrendezéssel nem megvalósítható
Mikrokontrolleren keresztül	gyors, általa bővíthető a rendszer	nagyobb késleltetés,illesztő áramkör szükséges hozzá,szabályzás nélkül kétes a működése

4. fejezet

Megvalósítás

4.1. Alkalmazás ismertetése

Előzmény

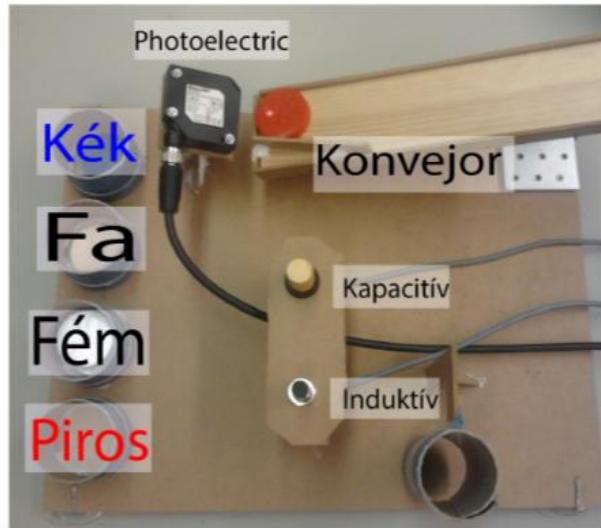
Az *Önálló laboratórium* c. tárgy keretein belül a robotkar és a hozzá tartozó szoftveres és hardveres környezet megismerése volt a feladat, valamint az összetett rendszer felhasználása egy komplex probléma megoldására. A félév során egy palettázó folyamatot valósítottunk meg, ahol a robotkar egy tárolóba érkező korongokat szortírozott azok anyaga, valamint színe alapján [8]. A mérési elrendezés a 4.1. ábrán látható.

A szakdolgozatom feladata ennek a palettázó folyamatnak a bővítése. A robotkart a gépi látás lehetőségeivel keresztezve létrehoztam egy olyan alkalmazást, amely képes meghatározni az asztalon véletlenszerűen elhelyezkedő korongok pozícióját, majd elmozdítani azokat. Jelenleg a palettázó folyamat előfeladataként funkcionál, viszont olyan esetekben, amikor kizárálag szín alapján válogatunk, akár helyettesítheti is azt. Ennek oka, hogy programom a korongokat a háttértől színük alapján különbözteti meg, tehát ha nem számít az anyagi minőség, kizárálag a korong színe meghatározó, akkor felválthatja a palettázás során részinformációt szolgáltató fotoelektromos színérzékelőt.

4.1.1. Palettázás

A robot számára lehetséges palettákat definiálni az alábbi módon [14].

Def[] Plt[] <Pallet No.>, <Start Point>, <End Point A>, <End Point B>, [<Diagonal Point>], <Quantity A>, <Quantity B>, <Pallet pattern>



4.1. ábra. Palettázó panel elrendezése. [8]

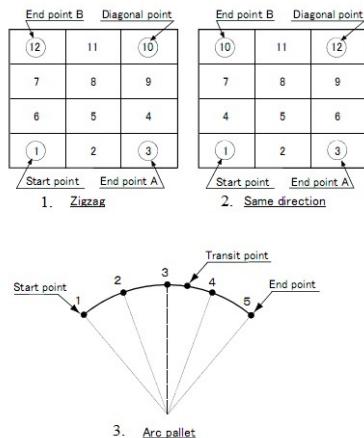
A paraméterek:

- *Pallet No.* - paletta száma (egyszerre több palettát is tud kezelní, konstans számot vár 1-8-ig),
- *Start Point* - kiinduló pozíció,
- *End point A,B* - a végpont pozíciók,
- *Diagonal Point* - kiindulási pozícióval átlóban lévő végpont pozíciója (tetszőlegesen adhatjuk meg a pontosság növelése érdekében),
- *Quantity A,B* - a kiindulási és a megfelelő végpont vonalán elhelyezkedő paletták összege, nullánál nagyobb egész számot vár paraméterként,
- *Pallet Pattern* - paletta minta, értékként egy számot vár a 4.2. ábra szerinti elrendezések alapján.

Palettára való hivatkozás

Plt[] <Pallet No.> , <Grid No.>

A kiválasztott, és előre definiált paletta (*PalletNo.*) megadott rácspontjának (*GridNo.*) pozíció értékével tér vissza. A hivatkozás paraméterei megadhatóak konstans, valamint változó értékekkel is. Érdemes megjegyezni, hogy a mozgató utasításnál közvetlenül nem alkalmazhatjuk a fenti szintakszist, mint bemeneti paramétert, a



4.2. ábra. Paletta minták.

program hibát jelezne. Azonban, ha egy definiált pozíció típusú változót egyenlővé teszünk vele, azt már felhasználhatjuk a robotkar irányításához. Tehát a $MOV(P1, 5)$ utasítás hibát okoz, míg $pPosition = (P1, 5)$ után a $MOVpPosition$ gond nélkül lefut.

4.2. Alkalmazott optikai eszköz

A feladat megvalósításához egy Logitech C905 webkamerát használtam. Az általa rögzített video képkockái szolgáltatják a képfeldolgozó alkalmazás bemeneti adatait.

4.2.1. Követelmények

A eszköz kiválasztásánál elsődleges szempontok voltak:

- könnyen rögzíthető kialakítás - a feladat megkezdésekor még nem volt definiált, hogy hol lesz később elhelyezve a kamera,
- lehetőség szerint nagy felbontás - növeli a pontosságot, azonban túl nagy felbontás esetén kritikusan megnövekszik a számítógép feldolgozási ideje,
- kompatibilitás a PC-n futó operációs rendszerrel, elérhető driver - a képfeldolgozás robosztussága miatt elengedhetetlen az egyes kamerák automatikusan bekapcsolt autófókusz funkciójának letiltása, ennek kikapcsolása nem lehetséges OpenCV környezetben.

A kamerát USB-n csatlakoztatva automatikusan felismerte az operációs rendszer, a feltelepített segédprogram segítségével képes voltam az autófókuszt leállítani. Az

eszköz két megapixeles, maximális felbontása 1600 x 1200 pixel, és másodpercenként 30 képkockát képes továbbítani [3]. Ezen paraméterek és tapasztalatok alapján választottam a fent említett típust (4.3. ábra).

Megkötések a kamera által vett területre:

- nem lehet tükröződő felületű,
- a felvételen a háttér egy nagy homogén felület legyen,
- átlagos fényviszonyok érjék.



4.3. ábra. Logitech C905 webkamera. [3]

4.2.2. A kamera elhelyezése

Mivel a folyamat szempontjából kulcsfontosságú volt a kinyert információk pontos-sága, az optikai eszközt a lehető legnagyobb precizitással kellett telepítenem. Egy mikrofontartó állvány végére erősítettem rá, melyet a térben minden irányba tudtam állítani. Végső pozíciója a robotkar kikapcsolt állapotban felvett helyzete fölött lett beállítva, mivel az alkalmazás során a robot számára ez a z irányú legnagyobb kitérés, így nem áll fent a veszélye, hogy esetlegesen a kamerának ütközik.

A szükséges mérések:

- robot maximális kartávolsága megfogási pozícióban,
- a kamera képének az előbb meghatározott területre való pozicionálása az asztallal való párhuzamosságának, valamint x,y irányú (robot XYZ koordináta-rendszer szerinti) illeszkedés figyelembevételével,
- $mm/pixel$ viszonyszám meghatározása (később kifejtve a 4.3.3 fejezetben).

4.3. A képfeldolgozó program

A szükséges információkat meghatározó applikációt Visual Studio 2010 fejlesztő-környezet C++ nyelven írtam az OpenCV könyvtár beépített függvényeinek felhasználásával. Ebben a fejezetben végigvezetem az olvasót a program működésének fontosabb lépésein.

4.3.1. A képfeldolgozás lépései

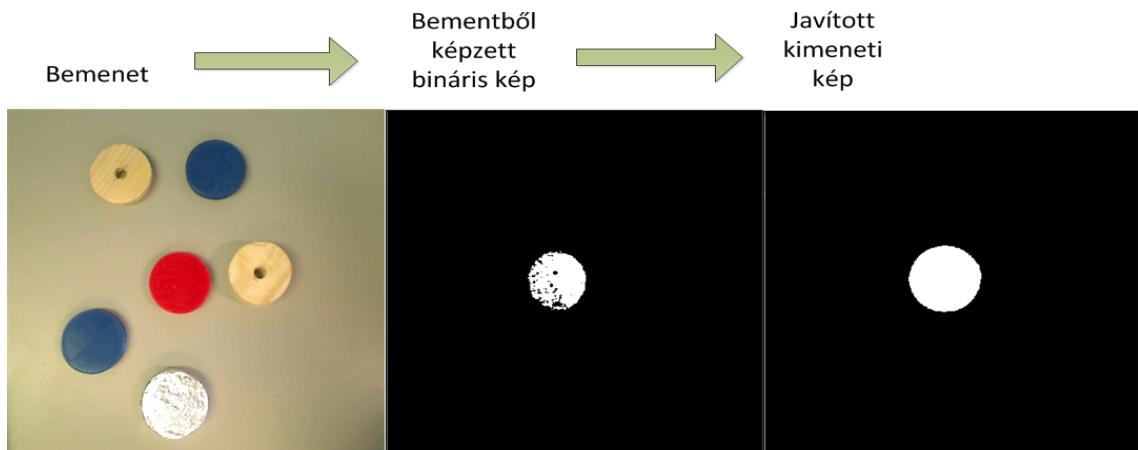
Az átláthatóbb tematikai felépítés, valamint a jobb olvashatóság érdekében elhagy-nám a későbbi soronkénti hivatkozást ebben az alfejezetben. A bemutatott algorit-mus elemeinek elméleti ismertetése megtalálható a 1. fejezetben.

A szükséges globális, valamint lokális változók definiálása, és a Modbus kapcsolat megnyitása után a program egy *while* ciklus keretein belül folyamatosan fut. Az alkalmazás első lépésként egy képkockát olvas be a kamerától, majd ezt a bemenetet menti egy *Mat* típusú n-dimenziójú tömb osztályba (OpenCV osztály, kép esetén a pixelek színértékeit tárolja). Az így létrehozott változón egyszerűen végezhetünk feldolgozási műveleteket. Attól függően, hogy melyik képtér alapján szeretnénk elvé-gezni a forrásunk bináris képpé való átalakítását, meghívhatjuk a *cvtColor()* függ-vényt, amely három paramétert vár: bemenet,kimenet,konverzió típusa. Amennyiben kihagyjuk ezt a lépést alapértelmezett BGR formátumban kell megadnunk a szűrés feltételeit. Az *inRange()* rutin meghívása során a forrás és célobjektum megadása mellett egy felső, valamint egy alsó határt definiálhatunk. Ezeket színkódok alapján határozhatjuk meg, értékét tekintve a tartományon belül eső pixelek a kimeneten fehér színűek, míg a tartományon kívül esők feketék lesznek. Amennyiben többféle szín szerint is szeretnénk a korongokat látni a kimeneten, egy többdimenziós tömbbe eltárolhatjuk a kellő adatokat és a *while* ciklus elején léptethetjük a tömb indexét. Bár a működés gyakorlatilag nem lesz párhuzamos, de a végrehajtási sebesség miatt azt a látszatot kelti a grafikus ablakban.

A kialakított bináris képen előfordulhatnak hibák (például a kamera által küldött kép, vagy a szűrés feltétele nem tökéletes...), ezek kiküszöbölésére használjuk a Gauss féle elmosást, majd a zárás műveletét. A javított adatsoron (4.4. ábra) fut-tatjuk a Hough-transzformációt, majd a program egy *for* ciklusba lép, amely addig fut, amíg a program elején definiált adatstruktúrát fel nem tölti az összes megtalált kör három darab paraméterével (középpont koordinátái, és a sugár pixelben megha-tározva).

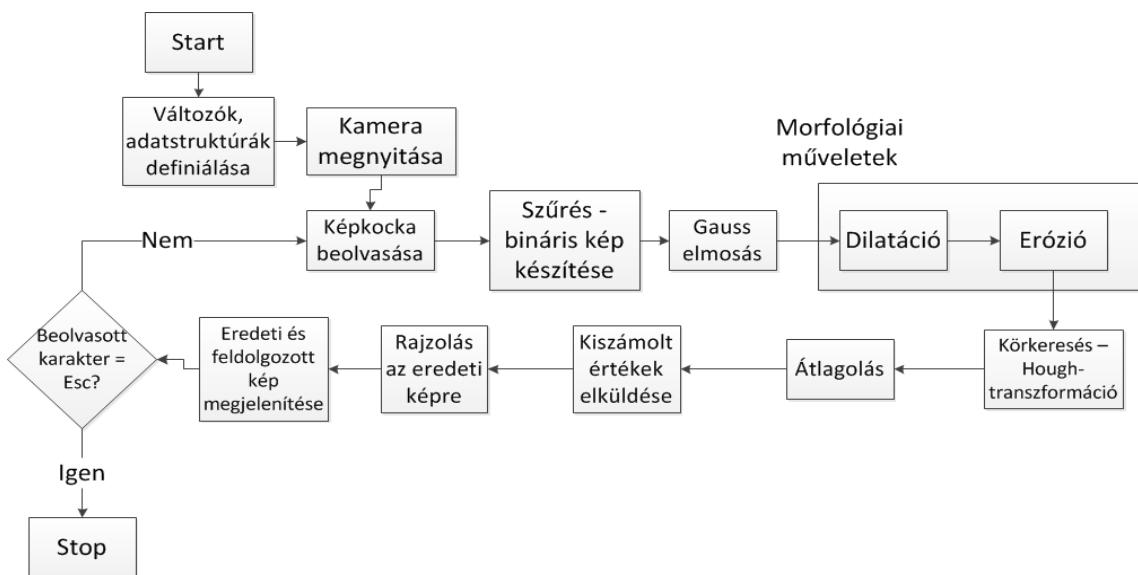
Következő lépésben a program kiolvassa a PLC memóriájának megfelelő regiszte-rét, és ha érkezett kérés a robotkar oldaláról, akkor elküldi a továbbiakban fontos

4.3. A KÉPFELDOLGOZÓ PROGRAM



4.4. ábra. Képkocka feldolgozásának folyamata piros színre való szűrés esetén.

adatokat. Ezekről az adatokról később lesz szó a következő 4.3.2. alfejezetben. A *for* ciklus után a felhasználó számára külön ablakban megjeleníti a beolvasott, valamint feldolgozott képkockákat, és grafikus indikátorként kört/köröket rajzol¹ az eredeti képre. A kiértékelt adatok átlagolása révén a megjelenített körök tökéletesen simulnak a célobjektumra, valamint pontosabbak lesznek a továbbított információk is. A *while* ciklusból meghatározott billentyű (Escape) lenyomása után lép ki a program. Bezárás előtt törli a kommunikáció során használt regiszterek értékét, majd bontja a Modbus kapcsolatot.



4.5. ábra. A képfeldolgozó program folyamatábrája.

¹Technikailag a *for* cikluson belül valósul meg a rajzoló rutin

Kisegítő lehetőségek

A program elején definiált csúszkák (angol nevén trackbars) segítségével a szín szerinti szűrés paramétereit (minimum és maximum korlátok) tudjuk futás közben manipulálni. Így újraindítás nélkül képesek vagyunk más színű korongok megtalálására.

Az egér kezelését végző függvény egyetlen bemenete a rutin által kezelt ablak. Előre beállított kombinációra (kattintás és billentyűzet ctrl gomb) kiértékeli az egér, mint marker pixelben mért pozícióját a kiválasztott képernyőn, valamint BGR színtér esetén a függvény segítségével beállíthatjuk a megadott pixel színértéke szerinti szűrési feltételeket.

A 4.1. képlet segítségével, és az egérkezelő rutin felhasználásával, valamint a kör sugarának ismeretében meghatározható, hogy a képen korong felszínére kattintottunk-e.

$$\text{dis} = \sqrt{((b_x - a_x) * (b_x - a_x) + (b_y - a_y) * (b_y - a_y))} \quad (4.1)$$

Ahol:

- dis - két pont távolsága,
- b_x - egérrel kattintott x pixel értéke,
- b_y - egérrel kattintott y pixel értéke,
- a_x - kiértékelt kör középpontjának x pixel értéke,
- a_y - kiértékelt kör középpontjának y pixel értéke.

Ezeket a lehetőségeket működés közben teszteltem, további információk megtalálhatóak az 5. fejezetben.

4.3.2. A képfeldolgozó alkalmazás által küldött adat

A program kiértékelt változói tehát az adott körök középpontjainak koordinátái, valamint a körök sugara pixelben kifejezve.

A kamera felbontását a programon belül manipulálhatjuk az OpenCV könyvtár megfelelő algoritmusának segítségével (1.3. alfejezet). Az általam választott 640x480 pixel a képanalízis szempontjából megfelelő eredményre vezet, valamint gyengébb számítógépeken se okoz feldolgozási idővel járó problémát.

Az alábbi két lehetőség tűnt megvalósíthatónak:

Először is a roboton lehetséges egy palettát definiálni az 4.1.1. alfejezetben bemutatott módon.

A paletta előnye, hogy nem kell a pixel értékeket átszámolni mm értékekbe, a három sarokpont pozíciójának betanítása után a robot csak egy paletta számot vár bemeneti értékként és abból automatikusan kiszámítja a megfelelő pozíciót. Ez esetben a paletta számát a képfeldolgozó program értékelné ki, ami nem lenne más, mint a korong középpont koordinátából képzett egész szám.

$$z = (x + 1) + yd \quad (4.2)$$

Ahol:

- z - paletta száma
- x - x koordináta értéke
- y - y koordináta értéke
- d - maximális sorok száma

A palettázás határa egyben ennek a megvalósításnak a hátrányát is jelenti. A robot maximum 32767 darab palettát tud kezelni. Ez körülbelül 181 pozíció soronként és oszloponként. Tehát a pixel palettával való megfeleltetés aránya nem lehet 1:1, mivel a 640x480 felbontás 307200 palettát feltételez, melyre a robotvezérlő program hibát jelezne.

A tároló rekeszek maximális száma (arányosan a 640x480 pixelfelbontáshoz) 207x155. Ez 32085 pozíciót eredményez, amely a maximális határon belül van. Ez alapján láthatjuk a lehetőség korlátait, mivel a programunkban keletkező hiba (akár a képfeldolgozás eredménye téved pár pixelt, akár a számolások utáni kerekítés miatt keletkező eltérés) felszorzódik.

Bár alapvetően egyszerűbb megoldásnak tűnt minden össze egy adatot elküldeni a robotnak, a precizitás nélkülözhetetlen a projektben, mivel a megfogáshoz milliméter pontosság szükséges. Így a soron következő lehetőség mellett döntöttem.

4.3.3. Az arány mérése

A biztosabb és ezzel együtt pontosabb megoldás, hogy a korong középpontjának x, y koordinátáit mm -be átskálázva küldjük el a robotnak.

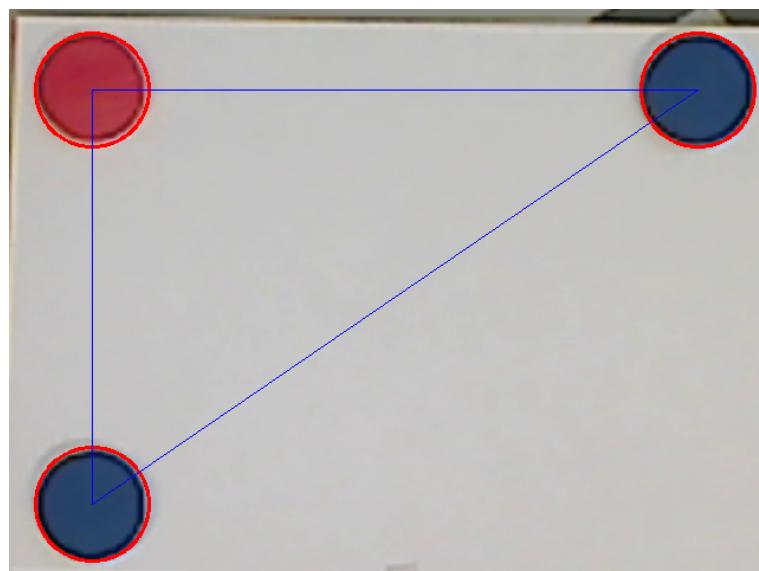
A programban számolt pixel, valamint a robotmozgató utasítás számára szükséges milliméter értékek között találom kellett egy pontos átváltást. A feladat megvalósításakor a következő kézi módszert használtam.

Elhelyeztem az előre meghatározott mérési panel három sarkába a szelektálni kívánt korongok közül három darabot.

Az OpenCV könyvtár függvényeinek segítségével köröket rajzoltam a képernyőn megjelenő korongokra, melyek közül 2-2 középpontja a koordinátarendszer megfelelő tengelyére illeszkedett (4.6. ábra). Ezen középpontok kivonásával megkaptam a körök között pixelben mért távolságokat.

A robotkarral a betanító pult segítségével felvettek a korongok megfogási pozícióját, majd elmentettem ezeket.

Az elmentett értékek alapján egyszerű kivonások segítségével megkaptam X valamint Y irányú, milliméterben mért távolságokat. A két kiszámolt adat hányadosa adja a számunkra szükséges milliméter/pixel arányt.



4.6. ábra. Körillesztés OpenCV segítségével.

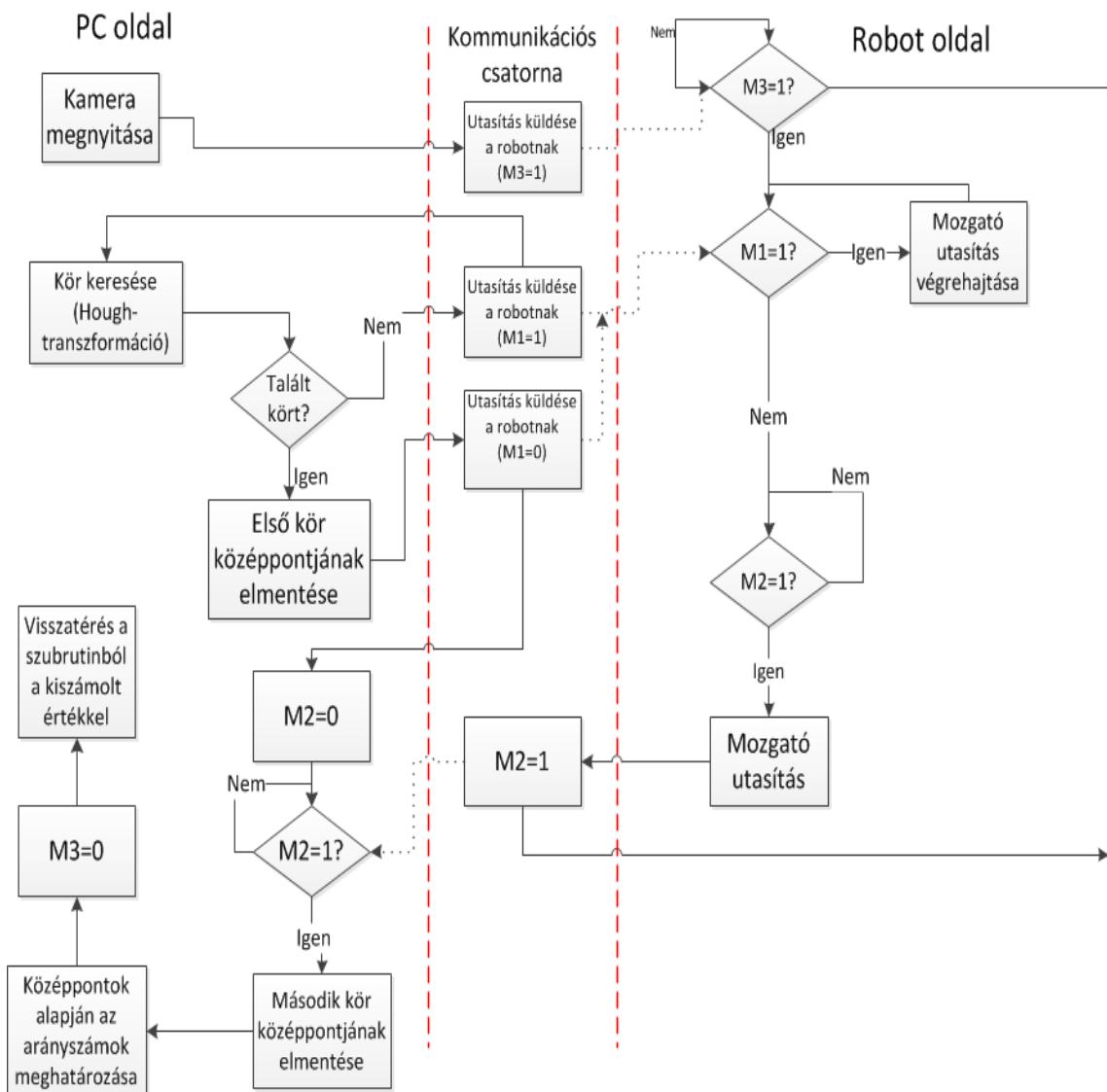
4.3.4. Kalibrációs program

A kommunikációs rendszer kialakítása után lehetőségem volt az előbb említett mérést automatizálni. Ennek jegyében elkészítettem egy programrészletet, amely lefutása után visszaadja a szükséges arányszámokat. A működéshez fel kell tennünk, hogy a robot kezében egy a szűrésre alkalmas korong található, vagy pedig a megfogó végberendezésre, mint hiányos körre tekinthetünk.

A megvalósításhoz mind a számítógép, mint a robot számára írnom kellett egy programrészletet. Az alkalmazás működése az alábbi 4.7. folyamatábra alapján nyomon

4.3. A KÉPFELDOLGOZÓ PROGRAM

követhető. A számítógép a Hough-transzformáció segítségével (OpenCV beépített függvény) minden egy koordinátatengely mentén eltolt (tehát a térben másik két irányban fix) körök középpontjainak pixeles különbségét számolja ki. Ebből az adatból valamint az előre definiált robotelmozdulásból osztással meghatározza a $mm/pixel$ arányt. Természetesen X , valamint Y irányban is külön-külön el kell végeznünk a kalibrációt.



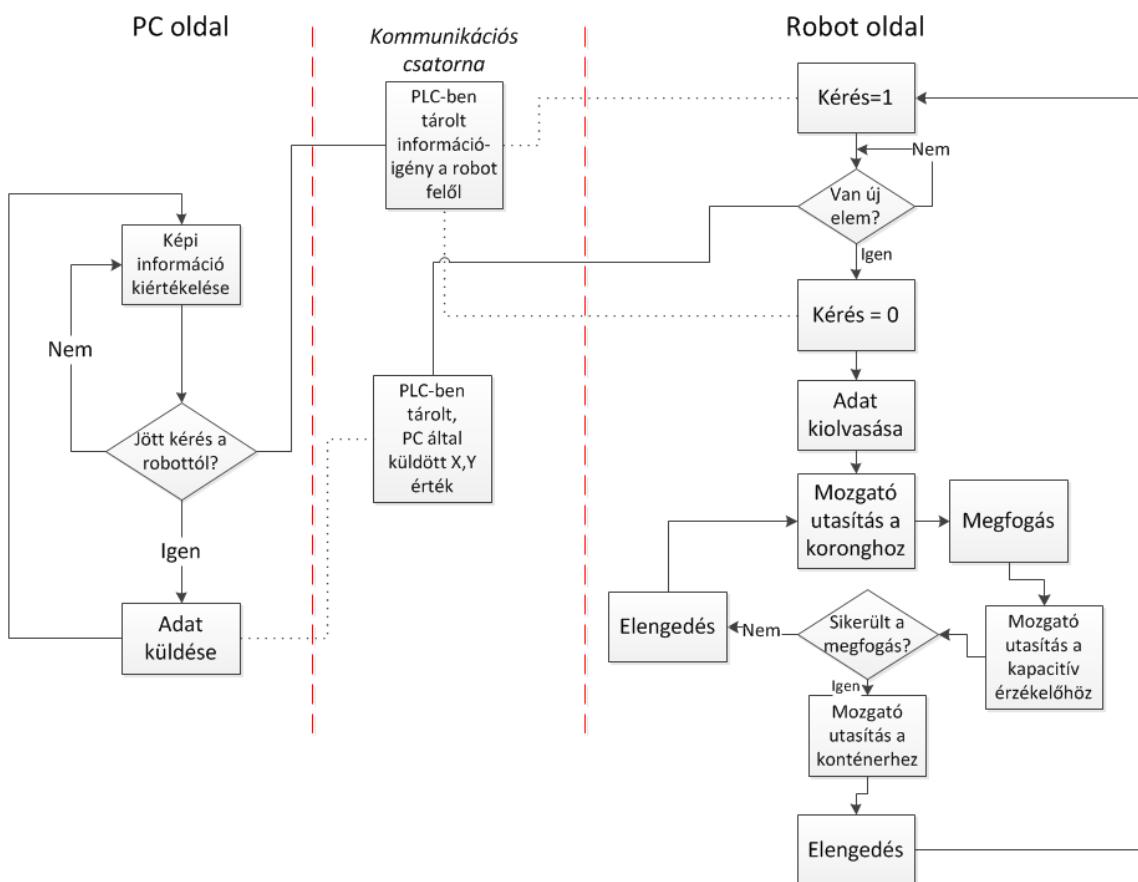
4.7. ábra. Automata kalibráció folyamatábrája egy koordinátatengely mentén.

A megvalósítás során nem ezt a módszert használtam, mivel nagy hátránya, hogy képfeldolgozás eredményére épült volna (amely nem szolgáltatott mindig tökéletes eredményt) a teljes későbbi működés pontossága. Nagy előnye, hogy elláttam megfelelő hibakezeléssel így nem küldhet olyan adatot, amely a robotot és annak környezetét veszélyeztetné, valamint ez a megoldás gyors és rendkívül mobilis.

4.4. Robotprogram

A robotkart vezérlő programot az RT Toolbox 2 keretein belül Melfa Basic 5 nyelven írtam. Az előző alfejezetben bemutatott elvek alapján a robotvezérlő program egy x, y értéket fogad milliméterben, ezt kell hozzáadnia a kiinduló pozícióhoz. Ezt a kezdeti helyzetet² kézi betanítással határoztam meg a betanító pult segítségével (2.1.4. alfejezet). Amennyiben sikerült a korongot a megfelelő helyre elhelyeznie, vagy a megfogás nem volt sikeres (ezt egy kapacitív érzékelőtől kapott digitális információ alapján tudjuk meghatározni), a folyamat újraindul mindenkorábban, amíg az asztalon van a kamera látószögében a szűrésnek megfelelő korong. Valamint az irányító szoftver adatot küld a számítógépnek, ha új pozíciót vár.

Tehát a robotot navigáló program feladatai között szerepel a memória olvasása, írása, előre meghatározott pozíció manipulálása a fogadott adatok alapján, valamint a folyamat ciklusba szervezése.



4.8. ábra. A teljes rendszer folyamatábrája.

²A pozíció tartalmazza azt a z koordinátatengely szerinti értéket, amely szükséges a korongok megfogásához. A működés feltétele, hogy az asztal egyenletes, valamint egy síkban van a robot talpával.

5. fejezet

Összefoglalás, értékelés

A jelenlegi projekt működéséhez elegendő lenne, ha a kamera képet készítene az asztalról és azt elemezné ki. Azonban amíg a számítógép erőforrásai nem számítanak szűk keresztmetszetnek a videó feldolgozását tekintve, addig élhetünk azzal a lehetőséggel, hogy valós idejű videót dolgozunk fel. Ez elősegíti a későbbi újításokat is a témaban.

Egy a megfogás számára kritikus tulajdonság a rendszer pontossága, amelyeket az alábbi módon lehet tovább fejleszteni:

- nagyobb kamera felbontás,
- a kamera asztallal való párhuzamosságának tökéletesítése,
- $mm/pixel$ arányszám meghatározása pontosabb módszerrel,
- a képfeldolgozó applikáció fejlesztése (a feldolgozott kép további szűrése, más körkereső algoritmusok tesztelése),
- új megfogó végberendezés.

Megállapítható, hogy a rendszer többszöri tesztelés alapján nagy pontossággal működik. Az esetleges hibák oka (megfogás sikertelensége) megfigyeléseim alapján a fizikai környezet és azon belül a megfogó végberendezés paramétereinek korlátaiban rejlik. A jelenlegi mérete, valamint a zárt és nyitott állapot között mérhető mm -es távolság miatt a korongokat X , Y koordináták mentén milliméteresnél kisebb toleranciájú pontosság esetén tudjuk felemelni. Tehát egy új, az alkalmazás számára ideálisabb megfogó eszköz beszerzésével a mért hibás kísérletek száma minimalizálható.

Általam célként kitűzött további fejlesztés egy Windows Grafikus Alkalmazás megvalósítása, ahol a bemutatott funkciók fejlesztése gördülékenyebben megvalósítható, valamint az elkészült program kezelése is felhasználóbarátabb.

A kamera képén közelítés hatására nemlineárisan nő a korongok felülete, mérések alapján megfelelő modellt lehet felállítani, melynek segítségével meghatározott magasságcsökkenés hatására megfigyelt méretváltozásból következtetni tudnánk a korongok *z* koordináta értékére.

A kisegítő lehetőségeknél (4.3.1. alfejezet) bemutatott függvények nem tartoznak szorosan a feladat működéséhez, de a későbbiekben jó alapot szolgáltathatnak további lehetőségek tervezésekor.

A feladat megvalósítása során annak komplexitása miatt nagy gyakorlatot tettem az új eszközök, rendszerelemek, programok és könyvtárak megismerése, valamint implementálása, és összekapcsolása terén. Rendszerszintű szemléletet sikerült kialakítanom magamban. Dolgozatomat szerettem volna abban a szellemiségen megírni, hogy annak elolvasása után a kiírt feladatot az olvasó képes legyen reprodukálni, valamint kellő ismeretet szerezzen esetleges fejlesztése, újragondolása kapcsán.

Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani mindenkinak, aki türelmével, támogatásával, vagy meglátásaival segítséget nyújtott jelen dokumentum megvalósításában.

Ábrák jegyzéke

1.1.	OpenCV logó[15].	8
1.2.	RGB színtér ábrázolása, valamint fontosabb színek értékei.	11
1.3.	HSV színtér ábrázolása, valamint fontosabb színek értékei. [6]	11
1.4.	Gauss elmosás hatása.	12
1.5.	Erózió hatása.	13
1.6.	Dilatáció hatása.	14
1.7.	Zárás művelete egyszer, valamint többször ugyanazon a forráson alkalmazva.	15
1.8.	Kör pontjának leképezése a paraméter-térbe. [22]	16
1.9.	Kör középpontjának meghatározása Hough-transzformáció segítségével. [22]	16
2.1.	Robotkar és irányítórendszer. [4]	18
2.2.	Mozgási irányok, melyek paraméterezésével megadható a robotkar pozíciója, orientációja. [14]	19
2.3.	System Q termékcsaládhoz tartozó PLC.	20
2.4.	Digitális bemenet.	20
2.5.	Megosztott memória címzése.	22
2.6.	Betanító pult.	23
2.7.	Adatküldés megvalósítása két lépésben.	24
3.1.	PLC monitor funkció.	30
3.2.	Modbus példaprogram.	32
3.3.	Virtuális Modbus szerver. [1]	33
3.4.	Virtuális Modbus kliens. [1]	33
4.1.	Palettázó panel elrendezése. [8]	38
4.2.	Paletta minták.	39
4.3.	Logitech C905 webkamera. [3]	40
4.4.	Képkocka feldolgozásának folyamata piros színre való szűrés esetén.	42
4.5.	A képfeldolgozó program folyamatábrája.	42

ÁBRÁK JEGYZÉKE

4.6. Körillesztés OpenCV segítségével.	45
4.7. Automata kalibráció folyamatábrája egy koordinátatengely mentén. .	46
4.8. A teljes rendszer folyamatábrája.	47
F.2.1 Modbus szerver.	56

Irodalomjegyzék

- [1] *Ananas hivatalos weboldala.* <http://www.tuomio.fi/ananas/index.htm>.
- [2] *Az OpenCV hivatalos oldala.* <http://opencv.org>.
- [3] *Logitech C905 webkamera.* <http://www.engadget.com/products/logitech/webcam/c905/specs/>.
- [4] Horog András. Robotprogramozás mérési segédlet. Technical report, 2015.
- [5] Hajdu András Fazekas Gábor. Képfeldolgozási módszerek. Technical report, 2004.
- [6] Dr. Vajda Ferenc. Számítógépes látás c. tárgy bevezető diapor. Technical report, 2015.
- [7] Az OpenCV hivatalos dokumentációja. *Dilatáció és erózió.* http://docs.opencv.org/doc/tutorials/imgproc/erosion_dilatation/erosion_dilatation.html.
- [8] Horog András, Széplaki Ádám. Ipari robotcella fejlesztése. Technical report, 2014.
- [9] ACROMAG INCORPORATED. Introduction to modbus tcp/ip. Technical report, 2005. http://www.prosoft-technology.com/kb/assets/intro_modbustcp.pdf.
- [10] Nem jellemző alapú alakzatfelismerő algoritmusok. Mesterséges intelligencia almanach. https://mialmanach.mit.bme.hu/erdekessegek/nem_jellemzo_alapu_alakzatfelismero_algoritmusok.
- [11] Robert Laganiere. *OpenCV 2 Computer Vision Application Programming Cookbook*. Packt Publishing Ltd., 2011.
- [12] Bíró Csaba, Pénzesné Kónya Erika, Mika János, Utasi Zoltán. Műholdakról távérzékeltek adatok feldolgozása és hasznosítása. 2011. http://www.tankonyvtar.hu/en/tartalom/tamop425/0038_informatika_Muholdas/ar01s17.html.

- [13] Mitsubishi. *Melfa Basic V help.*
- [14] Mitsubishi. *RV-2F-Q - CR750 Detailed explanation of functions and operations.*
- [15] OpenCV. Wikipédia szócikk. november 2012. <http://en.wikipedia.org/wiki/OpenCV/>.
- [16] Image Processing. Webopedia szócikk. http://www.webopedia.com/TERM/I/image_processing.html/.
- [17] Image Processing. Wikipédia szócikk. http://en.wikipedia.org/wiki/Image_processing/.
- [18] Microsoft Visual Studio. Wikipédia szócikk. http://hu.wikipedia.org/wiki/Microsoft_Visual_Studio.
- [19] Színtér. Wikipédia szócikk. december 2014. <http://hu.wikipedia.org/wiki/Színtér/>.
- [20] HSV színtér. Wikipédia szócikk. http://en.wikipedia.org/wiki/HSL_and_HSV.
- [21] Computer Vision. Wikipédia szócikk. július 2014. http://en.wikipedia.org/wiki/Computer_vision/.
- [22] Kató Zoltán. Alakzatok és minták detektálása. Technical report.
- [23] Monszpart Áron. Képfeldolgozó algoritmusok fejlesztése virtuálisvalóságrendserekhez. Technical report, 2009.
- [24] Gary Bradski és Adrian Kaehler. *Learning OpenCV*. O'Reilly Media, 2008.

Függelék

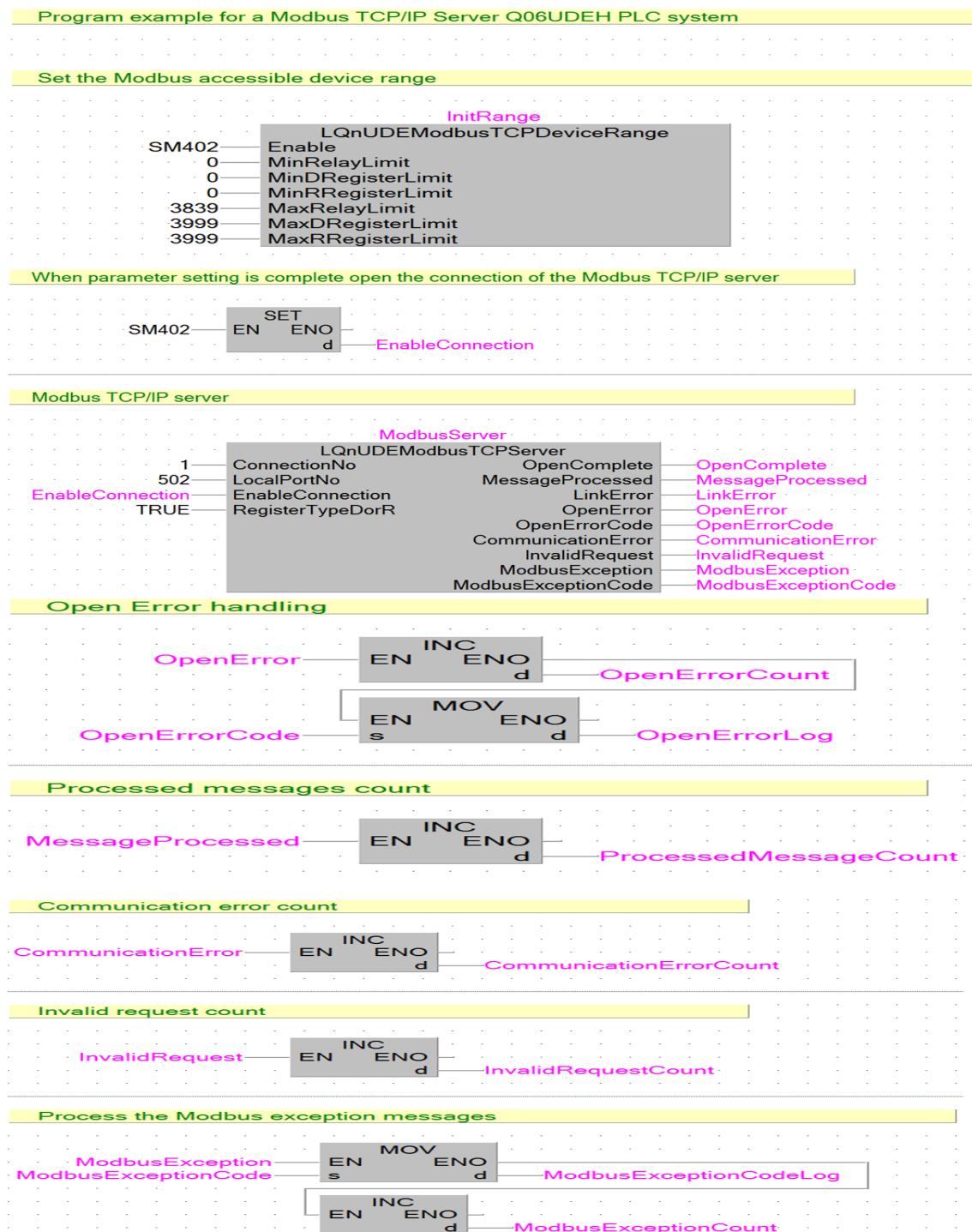
F.1. CD melléklet tartalma

Az adathordozón elérhető fájlok:

- robotprogram kódsor (robotprogram.txt),
- képfeldolgozó program forrásfájl (PC_progam.cpp),
- szakdolgozat pdf formátumban,
- videó az alkalmazás működéséről (robotic_arm.avi).

A szakdolgozathoz csatolt CD-ROM tartalma megtalálható a diplomaterv portálon is mellékletként.

F.2. PLC Modbus szerver létraprogramja



F.2.1. ábra. Modbus szerver.