



SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
MÉRNÖKI KAR

***SZORTÍROZÓ FELADAT MEGVALÓSÍTÁSA IPARI
ROBOTKARRAL ÉS KAMERÁVAL***

TDK DOLGOZAT

Kormány Tibor

mechatronikai mérnök BSc

hallgató

Témavezetők:

Csík Dominik Miklós

egyetemi tanársegéd

Pesti Richárd

egyetemi tanársegéd

SZEGED

2024.

TARTALOMJEGYZÉK

1. BEVEZETÉS.....	2
2. IRODALMI ÁTTEKINTÉS	3
2.1. Iparban használt robotkar típusok felépítésük szerint.....	3
2.1.1. Csuklókaros Robotkar (RRR)	4
2.1.2. Gömbi koordinátás Robotkar (RRT)	4
2.1.3. Scara Robotkar (RRT).....	5
2.1.4. Henger koordinátás Robotkar (RTT).....	7
2.1.5. Derékszögű koordinátás Robotkar (TTT)	8
2.1.6. Delta Robotkar	9
2.2. A hatsuklós robotkarról.....	9
2.2.1. A hatsuklós robotkar tulajdonságai.....	9
2.2.2. Hatsuklós robotkar ipari felhasználása	11
2.3. Képfeldolgozó eszközök működési elve.....	12
2.3.1. Kép digitalizálása	12
2.3.2. Képfeldolgozási műveletek.....	13
3. MEGVALÓSÍTÁS	14
3.1. Színérzékelő TCS34725	16
3.1.1. TCS34725-höz tartozó Arduino könyvtár	19
3.2. Elkészített Arduino Program bemutatása.....	19
3.2.1. Létrehozott függvények leírása	21
3.3. Arduino-Fanuc Összekötése.....	28
3.4. Fanuc oktatócella	29
3.4.1. iRVision.....	30
3.4.2. Fanuc program leírása	35
4. EREDMÉNYEK	39
5. ÖSSZEFOGLALÁS.....	40
5.1. Fejlesztési lehetőségek	41
IRODALOMJEGYZÉK	42
MELLÉKLETEK.....	44

1. BEVEZETÉS

Egyetemi tanulmányaim alatt lehetőségem nyílt részt venni a Fanuc Olimpia Robot versenyen. Mivel a két egymást követő évben is indultam ezen a megmérettetésen csapattársammal, így evidens választásnak tűnt a TDK dolgozat témáját is a Fanuc robotkarhoz kapcsolni.

A mai ipari világban már nehezen elképzelhető, hogy egy gyártósoron ne legyen implementálva valamilyen robotkar. Széleskörű feladatokra lehet alkalmazni ezeket a robotkarokat és ebből fakadóan sok típusa is jött létre. Dolgozatom során utánajártam ezeknek a robotkar típusoknak, alapkonfigurációjuk szerint és említést teszek azoknak sajátos előnyeikről. Külön kitérek a hatsuklós robotkarra és ismertetem tulajdonságait és ipari felhasználását. A feladatomként beprogramozom a Fanuc ER-4iA hatsuklós robotkarját.

Emellett szükség szokott lenni a termék vizsgálatára is, amelyhez valamilyen szenzoros rendszert alkalmaznak. Jelen TDK dolgozatom során utánajártam a kép alapján történő alakzat felismerésnek. Megismerkedtem az alapvető képfeldolgozásnál használt műveletekkel. Kitanultam a Fanuc oktatócellához tartozó iRVision szoftver kezelését.

Információt gyűjtöttem a dolgozatomban használt TCS34725-ös érzékelőről, annak működési elvéről és használatáról.

Dolgozatom célja egy szín szerinti szortírozó rendszer tervezése és kiépítése. A projekt megvalósítása Fanuc robotkar oktatási célú cellájának felhasználásával történt meg, amely Fanuc ER-4iA típusú robotkart tartalmaz. A projekten belül, létre kell hoznom a robotmozgást megvalósító Fanuc robotkar programot, alkalmaznom kell az oktatási cellában található szürkeárnyaltos kamerát az iRVision-ös Fanuc szoftver segítségével, valamint egy színérzékelő implementálása a Fanuc robotkarhoz, amely képes felismerni az érzékelőjéhez tartott munkadarabok színtkomponenseit, valamint megoldható legyen a kommunikációja a Fanuc robotkarral.

Ez a megvalósítás gyorsabbá és egyszerűbbé tudja tenni a színszerinti válogatást, mintha emberi munkaerőre hagyatkoznánk. Sokkal kisebb a hibázás lehetősége és precízebb munkát tud biztosítani. Egyik felhasználási területe az élelmiszeriparban található, ahol a kézi hibás gyümölcsök kiválogatását hatékonyabbá tudja tenni egy ilyen robotkaros színszortírozó rendszer.

2. IRODALMI ÁTTEKINTÉS

Ebben a fejezetben kifejtem a robotkarok felépítését, valamint részletezem a különböző konfigurációjú robotkarokat, kinézetre és tulajdonságaikra. A dolgozatom során használt hatsuklós típusú robotkart részletesebben taglalom. Írok a tulajdonságairól és bemutatom a felhasználási területeit az iparban, hogy miért használják széles körben ezeken a területeken.

A dolgozatom során felhasználásra kerül a Fanuc cella kamerás rendszere és ennek okán utána jártam, hogyan is történik egy ilyen rendszer esetén a képfeldolgozási művelet és leírom a képszegmentálás alapötletét.

2.1. Iparban használt robotkar típusok felépítésük szerint

Az iparban használt robotkarokat csoportosítani lehet a robotkar rögzített alapzatától kezdődő első 3 csukló típusával. Ezt a robotkarok alapkonzfigurációjának nevezzük. A csuklóknak két típusa van, rotációs és transzlációs. A rotációs csukló a szegmens forgó mozgását biztosítja a másik szegmens körül. A transzlációs csukló a szegmens haladó mozgását biztosítja a másik szegmenshez viszonyítva. A robotkar alapkonzfigurációja meghatározza a munkaterének formáját, azaz azt a ponthalmazt, amelyet képes elérni a robotkar. További csuklók alkalmazása lehetővé teszi a komplexebb mozgások elvégzését. Az utolsó szegmensen található az endeffektor (legtöbbször valamilyen megfogó), mely kapcsolatba kerül a munkadarabbal. Az effektor kiválasztott szerszámközéppontját TCP-nek rövidítjük, amely az angol Tool Center Point szóból ered (Mester, 2011).

A következő fejezetben a fontosabb alapkonzfigurációjú robotkarokat mutatom be, melyek: csuklókaros robotkar (RRR), gömbi koordinátás robotkar (RRT), Scara robotkar (RRT), henger koordinátás robotkar (RTT), derékszögű koordinátás robotkar (TTT), delta robotkar.

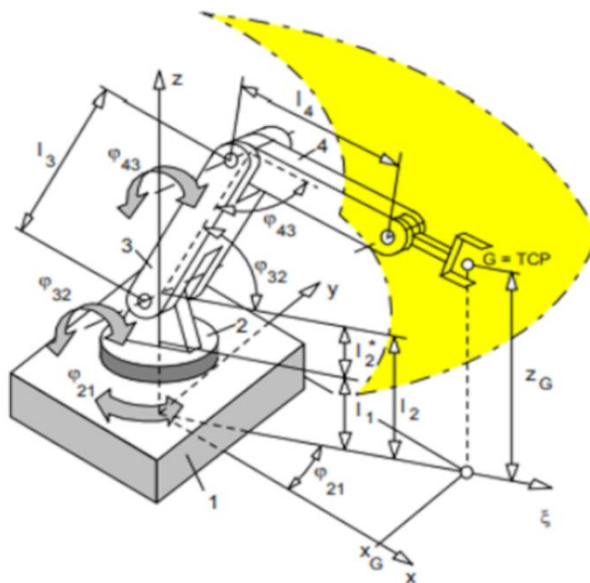
1. táblázat Az alkalmazott képletek betűjelöléseinek magyarázata:

(x,y,z)	Derékszögű koordináta rendszer koordinátáját jelöli.
l	A robotkar két csuklója közötti távolságot jelöli, vagyis a robotszegmens hosszát.
q	A robotkar csuklóihoz tartozó csuklókoordináták vektorát jelöli. A csuklókoordináta rotációs csukló esetén elforgatási szöget jelöl, transzlációs csukló esetén pedig a csukló tengelye menti elmozdulást jelöl.
s	A robotkar világkoordinátája. Meghatározza az effektor pozícióját és orientációját a robotkar alapjához kötött koordináta rendszerben. 6 értékből áll. Az (x,y,z) a pozíciót jelöli az (ψ - yaw, θ - pitch, ϕ - roll) pedig az orientációt.

2.1.1. Csuklókaros Robotkar (RRR)

Az alapkonfigurációját 3db rotációs csuklóból álló kinematikai lánc teszi ki. Karjaikat összekötő kényszerek közül a két egymással párhuzamos tengelyű irányba szerint kettő felé lehet sorolni őket: függőleges vagy vízszintes síkú csuklókaros robotkar.

A függőleges síkú robotkar esetén az egyik kényszernek a tengelye függőleges, míg a másik kettőnek a tengelye vízszintes, ezáltal hozva létre a függőleges mozgást, lásd: (1. ábra) (Dr. Pintér, 2017).

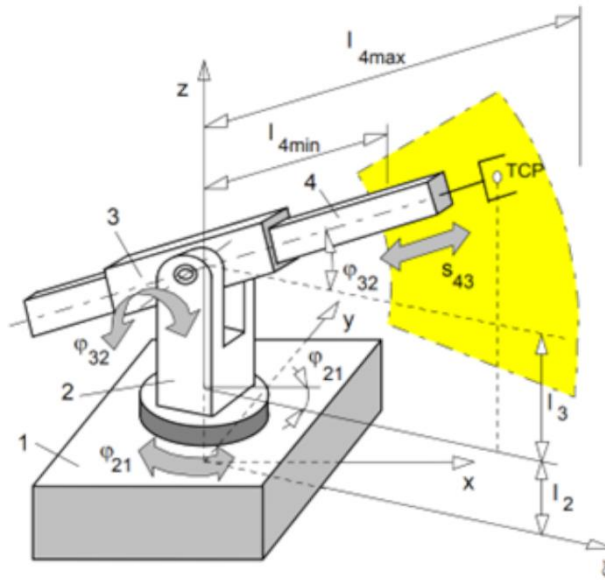


1. ábra Csuklós robotkar felépítése forrás: (Béla, 2012).

A robotkarok esetén mátrixszámításokat kell végezni, hogy megállapítsuk a robot TCP koordinátáját. Vannak robotkar konfigurációk, ahol kevesebb számítást igényel ennek kiszámítása ilyen például a derékszögű koordinátás robotkar (2.1.5), viszont a hatsuklós robotkar esetén sokkal nagyobb számítást igényel, amelyet bővebben a (2.2.1) fejezetben taglalok (Béla, 2012).

2.1.2. Gömbi koordinátás Robotkar (RRT)

A gömbi koordinátás robotkarok belül üreges gömb munkatérrel rendelkeznek. Alapkonfigurációjuk kettő rotációs és egy translációs csuklóból épül fel. Ezen robotkarok munkatérének geometriája hasonlít az előzőleg taglalt csuklókaros RRR alapkonfigurációs robotkar munkatérével, lásd (2. ábra) (Dr. Pintér, 2017).

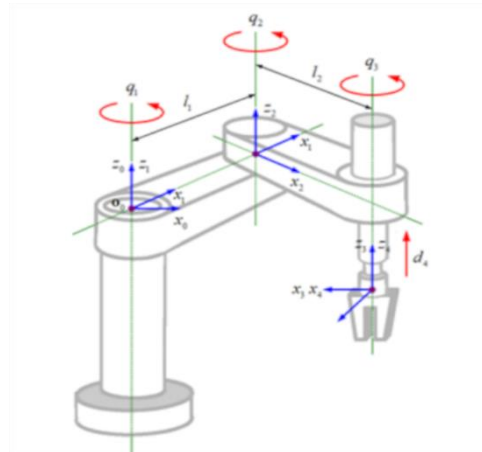


2. ábra Gömbi koordinátarendszerű robot felépítése forrás: (Béla, 2012).

Az ábrán látható módon ez a felépítés a robotkar mozgását lehetővé teszi mind horizontális, mind vertikális irányban, míg a munkatér mélységét egy a sugár irányával megegyező irányú transzlációs mozgás biztosítja.

2.1.3. Scara Robotkar (RRT)

Ugyanazzal az alapkonfigurációval rendelkeznek, mint az előbb említett robotkar viszont itt más orientációban vannak elhelyezve a csuklók. Itt nem gömbölyű munkatérrel veszik fel a robotkar, hanem hengeres formáját, lásd (3. ábra).

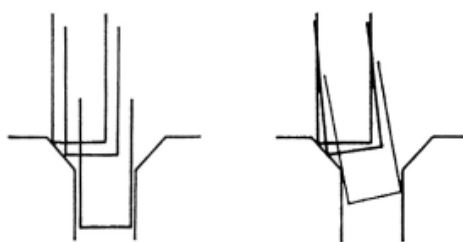


3. ábra Scara robotkar modell (Lavin-Delgado és mtsai., 2020)

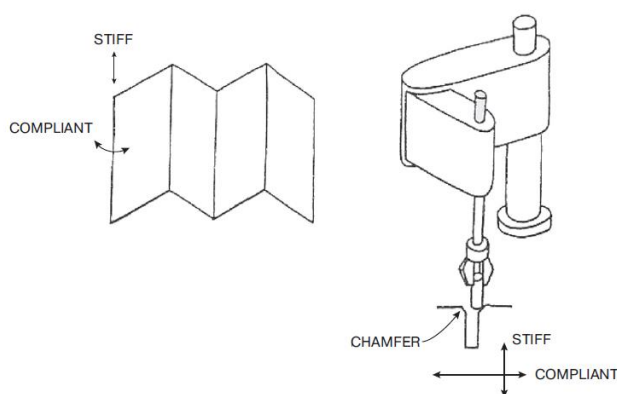
A (3. ábra) látható, hogy 4 darab csuklóval rendelkeznek, de az alapkonfigurációba a q_1 -es, q_2 -es, és a d_4 -es csukló tartozik bele. Az első két rotációs csuklóval alakítjuk ki a hengeres

tér alapját, míg a translációs csuklóval adunk mélységet a hengeres térnek. A q_3 -as csukló felel a vég-effektor z tengely körüli forgatásáért.

Maga a SCARA megnevezés a Selective Compliance Assembly Robot Arm rövidítéséből ered. A megnevezésben szereplő Selective Compliance a szelektív engedékenységre utal. Vertikális irányban merev, míg a horizontális irányban engedékeny. Ezt a byobu, Japánban használt összecusukhatós paravánszerű szerkezettel éri el, lásd (5. ábra). Ezzel a típusú szerkezettel megoldást találtak a „peg-and-hole problem”, vagy magyarul a csap és lyuk problémájára. Ennek a problémának az az alapja, hogyha vertikálisan el tud billeni a csap, akkor beleszorulhat már az elején a lyukba. Mivel ez a szerkezet vertikálisan merev és horizontálisan engedékeny így, ha nem is pontosan talál bele a lyukba, akkor is könnyedén bele tud illeszkedni, lásd (4. ábra). Ezt a típusú robotkart széleskörben használják különböző összeszerelési, de legfőképpen Pick & Place feladatoknál (Hiroshi, 2014).



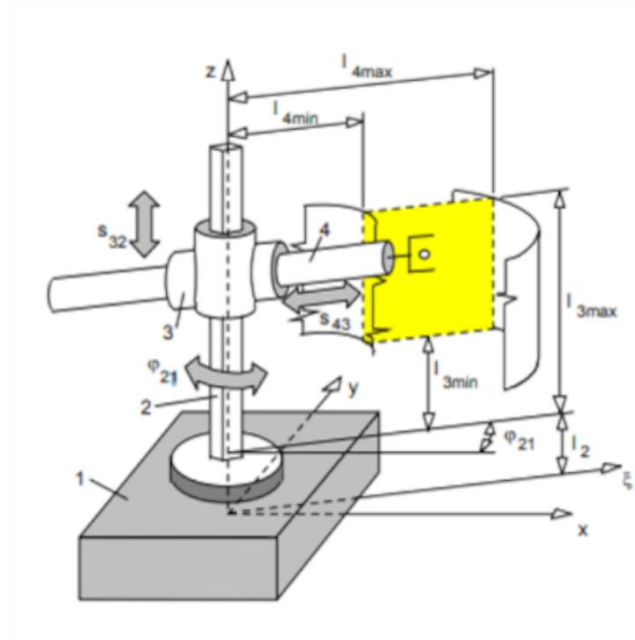
4. ábra csap-és-lyuk probléma (Hiroshi, 2014).



5. ábra Byobu-szerű struktúra (Hiroshi, 2014).

2.1.4. Henger koordinátás Robotkar (RTT)

Az ilyen felépítésű robotkarok két transzlációs csuklóval és egy, az alapnál elhelyezkedő rotációs csuklóval, rendelkeznek, ezáltal elérve a hengeres munkatért, lásd (6. ábra) (Zhang & Wei, 2017).



6. ábra Hengeres robotkar felépítése forrás: (Béla, 2012).

A transzlációs csuklók a vízszintes (X) és a függőleges (Z) síkban biztosítanak mozgást, míg a rotációs csukló a bázispont körül forog (θ). A rotációs tengely egybeesik az egyik transzlációs csukló tengelyével, így a robotkar kiválóan alkalmas szalagok közötti objektumok mozgatására. Jó emelési kapacitása van, és könnyen programozható (Poole, 1989) (Béla, 2012).

A felépítéséből látszódik, hogy sokkal kompaktabb a kialakítása ezáltal szűkebb helyeken is alkalmazható, valamint kisebb az esélye annak, a többi robotkonfigurációhoz képest, hogy akadályba ütközzön.

Ebben az esetben az alábbi képletek felhasználhatóak a robot TCP koordinátájának kiszámításához (Béla, 2012).

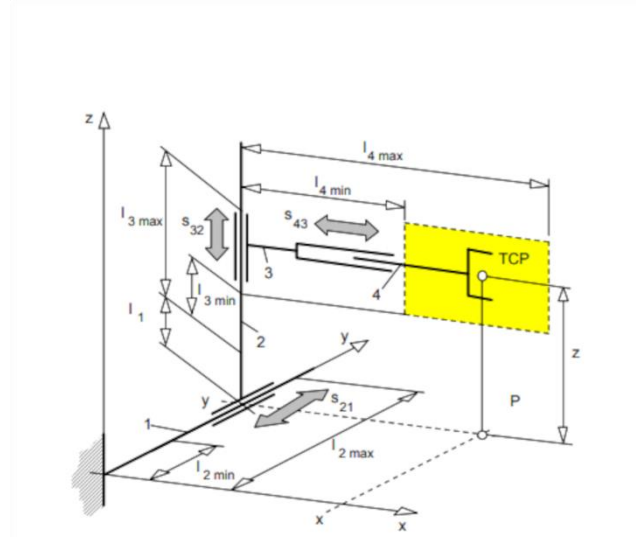
$$x = (l_{4min} + d_{43})\cos q_{21} \quad (1)$$

$$y = (l_{4min} + d_{43})\sin q_{21} \quad (2)$$

$$z = l_2 + l_{3min} + d_{32} \quad (3)$$

2.1.5. Derékszögű koordinátás Robotkar (TTT)

A derékszögű koordinátás robotkar alapkonfigurációja 3 transzlációs csuklóból áll, tehát 3 lineáris tengely mozgására alkalmas, amelyek egymásra merőleges elhelyezkedésűek, lásd (7. ábra) (Zhang & Wei, 2017).



7. ábra Karteziánus robot vázlatos felépítése forrás: (Béla, 2012).

Két típust különböztetünk meg ennél a felépítésnél. Vannak a portálos és vannak az álló változatok.

Álló változatnál egy állványon helyezkedik el a robotkar és a bázis koordinátája. A legtöbb robotkar típus álló változatú. Az ilyen robotkarok csak a csuklók által meghatározott munkatérben belüli pontokat tudják elérni (Béla, 2012) (Poole, 1989).

A portálos (vagy angol nevén *gantry*) változat felülről van felszerelve, és sínrendszeren mozog a bázisa, ezáltal képes akadálymentesen keresztül menni zsúfolt helyeken. A mozgó bázisáról képes leereszteni a robotkart és ezáltal mozgatni az objektumokat, főleg a nehezebbeket (Poole, 1989).

Ennél a felépítésnél a robotkar tengelyei egyértelműen vannak hozzárendelve a világkoordináta tengelyeihez, így az alábbi képletekkel meghatározható a robot TCP pontja (Béla, 2012).

$$x = l_{4min} + d_{43} \quad (4)$$

$$y = l_{2min} + d_{21} \quad (5)$$

$$z = l_1 + l_{3min} + d_{43} \quad (6)$$

2.1.6. Delta Robotkar

Maga az alapötlet a delta robotkarok kialakításra a paralelogramma adta. Ezzel a geometriai formával egy síkba lehet tenni a végszerszám mozgását. Három ilyen sík alkalmazásával tudjuk mozgatni a térben 3 szabadság fokkal (Bonev, 2001).

A delta robotkarok a párhuzamos robotkarok közé tartozik. Maga a párhuzamos robotoknak a kinematikai lánc zárt, azaz a végszerszám a robot bázisához több egymástól független kinematikai láncsal van csatlakoztatva (Wu & Shen, 2021).

Mivel nem sorosan csatlakoznak a csuklók, így ezáltal nem adódnak össze a csuklókat mozgó mechanizmusuk hibái, ezáltal nagyobb pontosságot és precízséget tudnak biztosítani. lásd (8. ábra) Jobban meg tud oszlan a terhelés a karok között, valamint a párhuzamos robotkar szimmetrikus kialakítottságának köszönhetően képes ugyanazt a sebességet és teljesítményt nyújtani bármely irányba történő mozgatasakor. Az aktuátorokat a robotkar bázisához közel helyezik el, ezáltal kisebb szögelfordulással nagyobb elmozdulást lehet elérni. Másik nagy előnye, hogy az aktuátoroknak sem kell a térben elmozdulniuk, amely által kevesebb tömeget kell mozgatni, szemben a soros robotkarokkal (Guanglei, 2013).



8. ábra ABB FlexPicker forrás: (Guanglei, 2013).

2.2. A hatsuklós robotkarról

Korábbiakban részletezve volt felépítése szerint ez a típusú robotkar. Ebben a fejezetben megvizsgáljuk jobban a tulajdonságai szerint és hogy az iparban milyen felhasználási módjai vannak.

2.2.1. A hatsuklós robotkar tulajdonságai

Maga ezt a típust szokták még csuklókaros robotkarnak is nevezni, mivel a csuklók elhelyezkedésével az emberi kart próbálták meg modellezni.

Ez a típusú robotkar a hat darab rotációs csuklójával éri el a vég-effektor hat szabadsági fokát. Az első csukló a bázis pontját határozza meg, valamint ez végzi a z tengely körüli forgatást. A második és harmadik felel a függőleges síkban való mozgásért. A maradék három csukló felel azért, hogy megfelelő orientációba mozgassa a vég-effektort.

A vég effektor pozíciójának a meghatározásához a Denavit Hartenberg féle transzformációs mátrix alkalmazása szükséges, amely felhasználja a homogén koordináta transzformációs mátrixot. Ezzel lehet a csuklókoordinátákat átalakítani világkoordinátákká. A módszer alapja az, hogy két haladó és két forgómozgással átvihető egy koordináta rendszer egy másikba. Ehhez a mátrixhoz ebből kifolyólag 2-2 paraméter tartozik. A d és az a a távolságra vonatkozik, az α és a q meg a szögelfordulásra vonatkoznak. A homogén koordináta-transzformációs mátrixszal leírhatók az egyes paraméterekhez tartozó mátrix, és ha mind a négy mátrixot összeszorozzuk, akkor megkapjuk a Denavit-Hartenberg féle transzformációs mátrixot (8) képlet.

$${}^{i-1}D_i = D(q_i)D(d_i)D(a_i)D(\alpha_i) \quad (7)$$

$${}^{i-1}D_i = \begin{bmatrix} \cos q_i & -\sin q_i \cos \alpha_i & \sin q_i \sin \alpha_i & a_i \cos q_i \\ \sin q_i & \cos q_i \cos \alpha_i & -\cos q_i \sin \alpha_i & a_i \sin q_i \\ 0 & \sin \alpha_i & \cos \alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (8)$$

A DH féle transzformációs mátrixszal képesek vagyunk a csuklóra rögzített koordináta rendszerből áttérni az azt követő másik rotációs csuklóra rögzített koordináta rendszerbe. Ha az összes csuklón végig megyünk így a bázisponttól a vég-effektorig és ezek a mátrixokat ebben a sorrendben összeszorozzuk, akkor megkapjuk azt a 4x4-es mátrixot, amely megadja nekünk a robotkarunk TCP koordinátáját.

$${}^0T_n = {}^0D_1 {}^1D_2 {}^2D_3 \dots {}^{n-2}D_{n-1} {}^{n-1}D_n \quad (9)$$

A bal felső index határozza meg a kiinduló pontot, míg a jobb alsó pedig a végpontot. Látható, hogy önmagában is sok számítást igényel a TCP koordinátájának meghatározása, így az ebből a koordinátából való csuklókoordináták visszafejtése még nagyobb számítást igényel (Mester, 2011).

Ezzel a részével az inverz kinematika foglalkozik. Az inverz kinematikai számítások során felmerülnek problémák, amelyek abból fakadóak, hogy az ahhoz használt egyenletek nem lineárisak és ebből kifolyólag nem lehet véges számú művelettel elvégezni. Ezen felül a megoldás rendelkezhet több vagy végtelen számú eredménnyel. Az ilyen esetekben a robotkar

nem tud egyértelmű mozgásműveletet rendelni az adott pozíció elérésére. Ezt szingularitásoknak szoktuk hívni. Ezekre a pozíciókra a robotkarunk programozása során figyelni kell (Siciliano és mtsai., 2009).

A robotkarokra tervezett inverz kinematikai megoldások még nem hibátlanok, így maga a számítás során is léphetnek fel hibák, amelyek abban mutatkoznak meg, hogy ugyanarra a pozícióra ismételt ráállása során eltérő eredményeket kaptunk. Ezen felül a hatcsuklós kialakítás nem tartozik a legprecízebb alapkonfigurációk közé. A csuklók sorosan vannak összekötve, ezáltal az egyes csuklók hibái összeadódnak a vég-effektornál. Ennek ellenére az iparban használt hatcsuklós robotkarok képesek mikrométeres pontossággal működni (Nayak és mtsai., 2019).

2.2.2. Hatcsuklós robotkar ipari felhasználása

Előnyét jelenti a többi robotkar típushoz képest a sokoldalúsága. Hat szabadságfokának, valamint az emberi karnak leképezésének köszönhetően, képes helyettesíteni azon feladatokat, amely precíziós karmozgásokat igényel.

Pakolás: Alapvetően, ha gyors anyagmozgatásra, vagy csak egyszerű egyik szalagról a másikra való mozgatásról van szó, akkor delta vagy scara robotkart alkalmaznak leggyakrabban. Ha fontos az orientációja a megfogandó munkadarabnak, vagy csak bizonyos oldaláról vagy pontjáról fogható meg, akkor már indokolt a hatcsuklós robotkar alkalmazása.

Hegesztés: A hegesztés alapvetően nagy közügyességet igényel, mivel akár egyes esetekben egy kis hiba is nagymértékben csökkentheti a hegesztés minőségét. Ezen felül a hegesztés során sok veszélynek vannak kitéve a dolgozók. Ebből kifolyólag jó megoldást tud nyújtani a hatcsuklós robotkar, mivel ez a típus képes leképezni azokat az íveket, pályákat, amelyet a hegesztés igényel nagy pontossággal, ráadásul ezzel kíméljük a munkásokat a veszélyes hatásoktól.

Festés: Hasonlóan a hegesztéshez ez is megadott pályákat igényel, valamint egyenletes adagolását a festéknek, amelyet a hatcsuklós robotkar könnyedén tud kezelni. Itt is veszélyhatásoknak vannak kitéve a munkások, mivel az emberre veszélyes kemikáliák jutnak ki a levegőbe, ezért itt is érdemes a robotkar használatát.

Összeszerelés: Ez a folyamat a gyártmány gyártási költségeinek akár 80%-át is kiteheti. Ezáltal előnyössé tud válni, ha ezt a folyamatot optimalizálni tudjuk. Maga a kézi munkaerő igénybevétele nem elég effektív egy robotkarhoz képest, amely nem csak gyorsabban, hanem képes nagyobb erővel is dolgozni és megemelni olyan munkadarabokat, amely egy ember számára kivitelezhetetlen. Ezen felül szükséges az összeszerelés során a munkadarab megfelelő

orientáltsága, amelyet pontosan tud biztosítani egy hatsuklós robotkar, szemben a többi típussal.

Munkadarab megmunkálás: A csiszoló, sorjázó, maró és fűrész megmunkálásokra, egyre elterjedtebb a soros kinematikájú robotkarok használata. Ezeknek a robotkaroknak a sokoldalúságának és költséghatékonyságának köszönhetően kiválóan alkalmasak ezen feladatok elvégzésére. Az ilyen munkafolyamatok során alkalmazott szerszámok passzív rugalmassággal, vagy aktív erőszabályzással vannak kombinálva a munkadarabok felületi eltéréseire vagy anyagi tulajdonságaira való tekintettel. (Hägele és mtsai., 2016).

Alkalmazva van az orvosi területeken is, ahol fontos a konzisztens erő kifejtés. Artériák feltárásához használt szondák mozgásánál fontos szempont volt, hogy meghatározott erőt fejtsen ki, amelyre az emberi kéz nem alkalmas és ezért merült fel az ipari robotkar alkalmazása. Ehhez szükséges volt megkövetelni a robotkar magas fokú biztonságos működését (Pierrot és mtsai., 1999).

2.3. Képfeldolgozó eszközök működési elve

A képfeldolgozás első szükséges lépése a kép készítése egy digitális kamera segítségével. Ezt követően a képfeldolgozás során különböző műveleteket végzünk a digitális képünkön, hogy ezáltal vagy számunkra jobban tudjuk alakítani, vagy elemzéseket tudjunk rajta elvégezni. Átalakítás során új képet kapunk, míg elemzés során információt nyerünk ki a képünkből (Czap, 2007).

Ahhoz, hogy biztosítani tudjuk a képfeldolgozást, eszközökre van szükségünk. Legfontosabbak közé tartozik a kamera, amely lehetővé teszi a fizikális feldolgozni kívánt kép digitalizálását, valamint a mérést, adatgyűjtést és feldolgozást végző szoftverek, amelyekhez szükség van számítások végzésére alkalmas gépre. Ezen felül kellhet a kamerához lencse, állványzat és legtöbb esetben megvilágítás (Czúni & Tanács, 2011).

2.3.1. Kép digitalizálása

Digitális kép készítésekor mintavételezzük a fizikai képet és ebből egy digitálisan értelmezhető számtömböt hozunk létre. Ez a számtömb tartalmazza a diszkrét értékeit a fizikai képnek.

Egy digitális kép két-vagy több dimenziós tömbből áll. Ezen tömbök első két dimenziójának mérete határozza meg a kép méretét, valamint a pozícióját az egyes képpontoknak. A harmadik dimenzió mérete a képben használt különböző színek számát határozza meg. Egy szürkeárnyaltos kép esetén a harmadik dimenzió mérete 1, míg RGB-s (Red, Green, Blue azaz piros, zöld, kék színekből álló) kép esetén 3.

A számtömbünk elemeinek mérete meghatározza az árnyalatok számát. Általában 8 bit-en tárolják az adott szín intenzitását, így bontva fel 256 különböző árnyalatra. Egy RGB-s kép esetén ez azt jelenti, hogy egy képpont lehetséges intenzitásának száma $2^{3 \cdot 8} = 16\,777\,216$, amely sok helyet igényel. Hogy csökkentsük ennek méretét anélkül, hogy észrevehető módon romoljon a kép minősége lehetőség van a kvantálási szintek csökkentésére, azaz 8-nál kevesebb bit felhasználására (Palágyi, 2011) (Pratt, 2001).

2.3.2. Képfeldolgozási műveletek

A képfeldolgozó rendszerünk bemenetén érkező kép feldolgozását csoportosíthatjuk az szerint, hogy a feldolgozás alatt álló képpontot mennyire befolyásolja a körülötte lévő pontok értéke. Ezek alapján van pont-pont, lokális és globális műveletek. (Czap, 2007).

Pont-pont műveleteknél csak a feldolgozandó képpont értékét vesszük figyelembe. Ilyen művelet lehet a küszöbölés, ahol állítunk egy küszöb értéket, és ha a képpontunk világossága meghaladja ezt az értéket, akkor egyes, azaz igaz értéket vesz fel, ha pedig nem, akkor meg nullás, azaz hamis értéket vesz fel. Ezáltal készül egy úgynevezett bit kép vagy ismertebb nevén fekete-fehér kép. Másik ebbe a kategóriába tartozó művelet a hisztogram készítés. Ennél összeszámoljuk, hogy a képünkön a világossági szintekhez hány képpont tartozik (Czap, 2007).

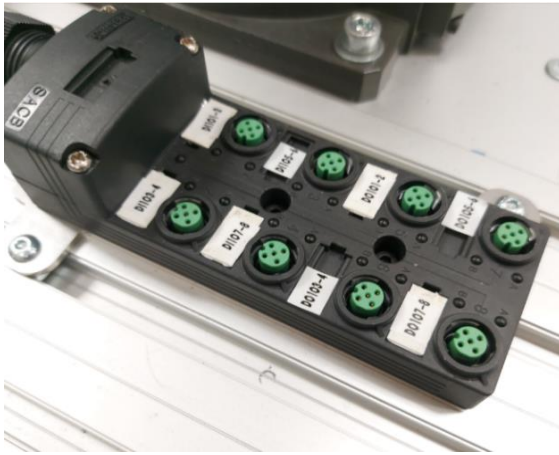
Lokális műveleteknél meg kell határoznunk a feldolgozandó képpont körüli terület nagyságát, amely hatással lesz a képpont újértékének kialakításában. Zajszűrésre lehet alkalmazni medián szűrőt, vagy Outlier szűrőt. Mindkettő lokális műveletnek minősül mivel a képpont körüli értékeket is számításba veszik. A medián szűrő a képpont és az az körüli világosságsszint értékeket veszi figyelembe, melyeket sorba rendez. A következő lépésben kiválasztja a középső, azaz medián értéket és azt írja be a képpont helyére. Az outlier szűrő a képpontkörüli értékeket átlagolja és ezzel az átlagolt értéket írja be a képpont helyért (Czúni & Tanács, 2011).

Globális műveleteknél a feldolgozandó képpont újértékét az összes képpont értéke befolyásolja (Czap, 2007).

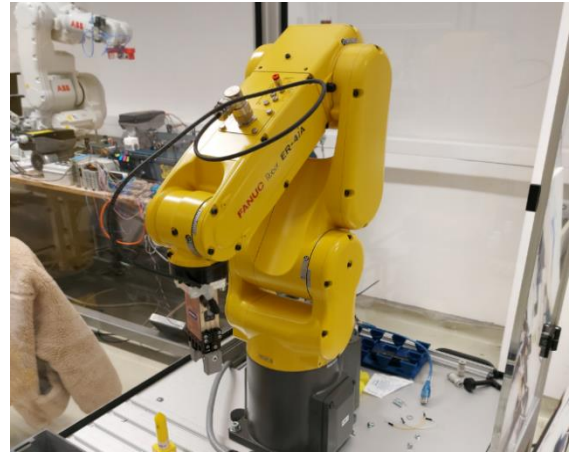
A küszöbölés és a hisztogram készítés ötvözésével egy kezdetleges szegmentálást tudunk csinálni. A hisztogram segítségével kiálló csúcserkéket, azaz, magas előfordulási számmal rendelkező világossági szinteket tudunk keresni, amelyről feltételezhetők, hogy képpontjaik egységes alakokat reprezentálnak a képen. Ezek után küszöböléssel külön tudjuk választani ezeket a magas előfordulási számmal rendelkező részeket (Haralick & Shapiro, 1984).

3. MEGVALÓSÍTÁS

Egy hatsuklós Fanuc robotkar, a beépített színérzékelő, valamint az iRVision-höz tartozó kamera segítségével szín szerint szétszortírozza a hasábokat. A színérzékeléshez a TCS34725-ös színérzékelőt használtam. Ezt egy Arduino UNO-val működtettem, amely a pinein keresztül küldte el a Fanuc robotkar IO-jára lásd (9. a) árba). Meg kellett valósítani az illesztést, amely az Arduino 5V-os jelét átalakítja 24V-os jellé, amelyet tranzisztorok segítségével értem el. A hasábok mozgatását a Fanuc ER-4iA típusú robotkar által végeztem lásd (9. b) árba). Ennek programozását a robotkarhoz tartozó TeachPendant-on vagy röviden TP-n végeztem el lásd (9. c) árba) A cellában beépített kamera segítségével tudja a robotkar meghatározni a behelyezett színes hasábok pozícióját lásd (9. d) árba) ezáltal tudva, hogy hova kell nyúlnia.



a)



b)



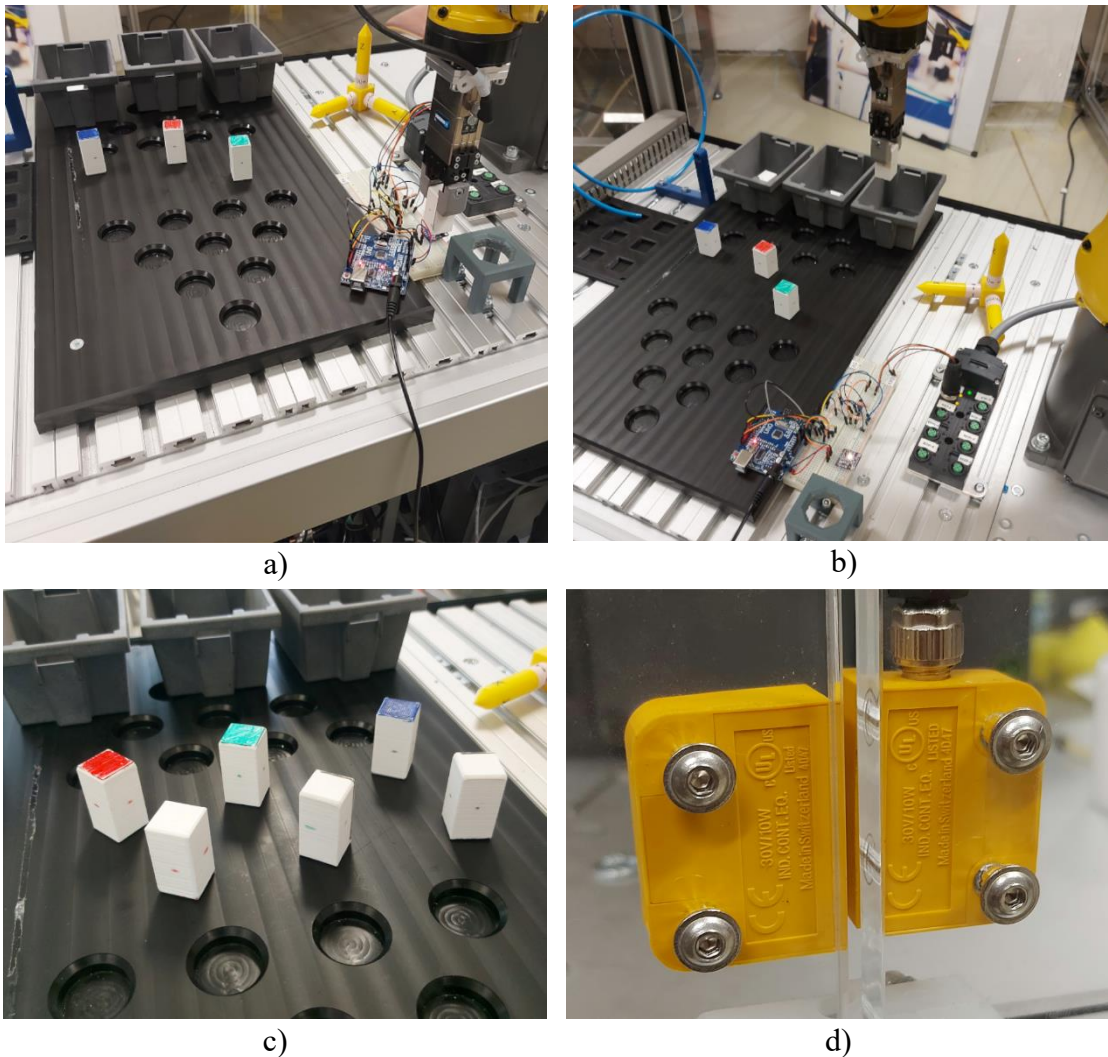
c)



d)

9. ábra a) IO modul b) Fanuc robotkar c) Teach Pendant d) Fanuc B/W kamera forrás:
(saját ábra).

Ezután a színérzékelő fölé viszi, lásd (10. a) árba) a hasábot és ott visszajelzést kap arról, hogy a 3 tároló közül melyikbe kell tennie, lásd (10. b) árba). Ezt mindaddig ismétli, amíg el nem fogynak a hasábok a felvételi mezőből lásd (10. c) árba). A dolgozatom során használt hasábok ez alábbi ábrán láthatók. A programok tesztelésére kiválóan alkalmasak voltak ezek a 3D nyomtatott hasábok. A megfogó gond nélkül meg tudta fogni. Tábla filctollak segítségével színeztem az aljukat, hogy későbbiekben le lehessen törölni róluk, ha más felhasználásra kellenek. AUTO módba álláshoz először be kell csukni a cella ablakát, hogy a biztonsági érzékelők nyugtázó jelet adjanak, lásd (10. d) árba). A következő lépés a program kiválasztása.



10. ábra a) Fanuc robotkar színérzékelő fölött b) Fanuc robotkar a tárolók fölött
 c) Hasábok felvételi mezője d) Fanuc cella fence érzékelője forrás: (saját ábra).

Ezt a TP-n található SELECT gomb megnyomásával lehet megtenni. A program futtatásához a TP bal felső sarkában található ENABLE kapcsolót OFF állásba kell kapcsolni.

Ezt követően a Fanuc vezérlő dobozán található módváltó gombbal lásd (11. ábra) át kell váltani AUTO-ba, majd egy SHIFT + RESET gomb együttes megnyomásával ki kell venni



11. ábra Fanuc módváltó, ciklus start, vészstop gombjai forrás: (saját ábra).

az ERROR-t és ezután nyomjuk meg a módváltó mellett található START gombot. Ekkor elindul a kiválasztott programmal a robotkar ciklus. AUTO módban a robotkar maximális sebességen működik.

3.1. Színérzékelő TCS34725

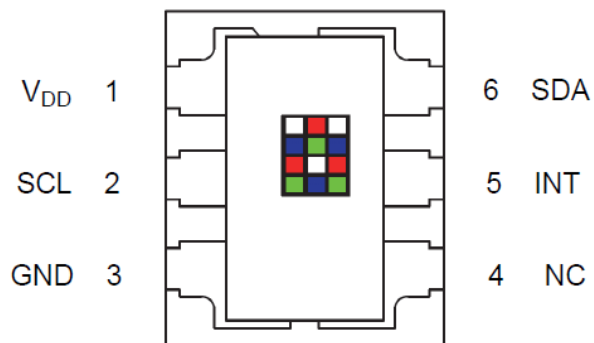
A dolgozatom során egy Arduino UNO által működtetett TCS34725 színérzékelőt használtam. Az érzékelőnek hat kivezetése van, amelyet a következő táblázat mutat be (2. táblázat). Az érzékelő I2C soros kommunikációt használ adatátvitelre, amelyhez az SCL és SDA pin-eket használja. 4 féle fotodióda van beleintegrálva, amelyek egy infravörös fény blokkoló szűrővel vannak lefedve. A diódák egy 3x4-es tömbben vannak, amelyek elhelyezkedése a (12. ábra) mutat be. Ezek a fényre érzékeny fotodiódák egy fényszűrővel vannak ellátva, amelyek csak a fotodiódák által mérni kívánt fényt engedik át. A 4 féle fotodióda a következőkből tevődik össze: R a vörös fénytartományra érzékeny, G a zöld fénytartományra, míg B a kék színtartományra. A negyedik, a C - clear value (tisztá érték) nincs ellátva külön szűrővel. Mindegyik fotodióda egy ADC analóg -konverterrel van ellátva, amely számszerűsíti az érzékelt fényintenzitásokat egy 16 bites számértékre. Tesztelések során azt tapasztaltam, hogyha nagy intenzitású (telefon vakuja) fényt bocsájtunk ki az érzékelőre, akkor mind a 4 érzékelő túlcsoordul és a 16 biten elérhető legmagasabb értéket adja ki, amely a 65535. Ebből következtethető, hogy a 16bitre való konvertálás lekorlátozza a mérés eredményét, viszont ez

nincs nagyon kihatással a mérésre, hiszen nincs szükség nagy intenzitású fények mérésére, hanem csak felületek színének mérésére. (TCS3472 adatlap).

2. táblázat TCS34725-ös színérzékelő pin kiosztása forrás: (TCS3472 adatlap).

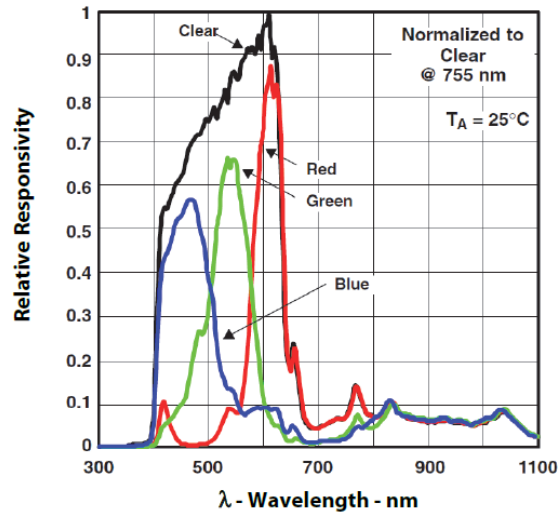
Pin száma	Pin neve	Pin típusa	Leírás
1	V _{DD}		Tápfeszültség (max 3,8V)
2	SCL	bemenet	I ² C soros kommunikáció órajele
3	GND		Tápfeszültség földje
4	NC	kimenet	Ne kössük be*
5	INT	kimenet	Interrupt - megszakítás
6	SDA	be- és kimenet	I ² C soros kommunikáció I/O terminálja

* A gyártó megköötése



12. ábra TCS34725-ös érzékelő forrás: (TCS3472 adatlap).

Az érzékelő használata során észrevehető volt, hogy az adott színre érzékeny fotodiódák intenzitás értékebe valamilyen mennyiségben beleszámítódik a másik kettő színtkomponens is. Ezt jól szemlélteti az adatlapjában található fotodiódák karakterisztikája lásd (13. ábra). A gyártó végzett méréseket ezzel kapcsolatosan, ahol vörös, zöld és kék színhez tartozó elektromágneses hullámhosszt kibocsájtó LED-el világítottak az érzékelőre és megállapítottak egy minimum és maximum %-ot arra, hogy mekkora arányban mutatott értéket az R a G és a B fotodiódák a clear value-hoz arányosítva.

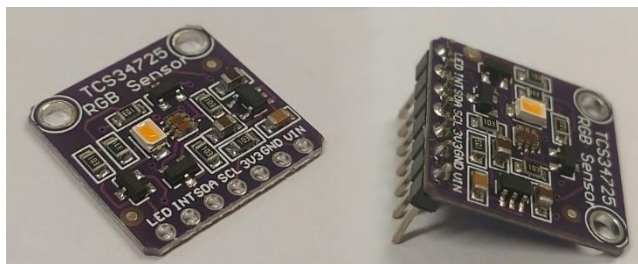


13. ábra Az R-G-B-C fotodiódák elektromágneses spektrumra való érzékenységeinek karakterisztikája forrás (TCS3472 adatlap).

3. táblázat fotodiódákon végzett mérések forrás: (TCS3472 adatlap).

Paraméter	Teszt kondíciók	Vörös csatorna		Zöld csatorna		Kék csatorna		Tiszta érték			Mértékegység				
		Min	Max	Min	Max	Min	Max	Min	Typ	Max					
R_e besugárzási érzékenység		$\lambda_D = 465 \text{ nm}$				0%	15%	10%	42%	65%	88%	11,0	13,8	16,6	érték száma/ μW /cm ²
		$\lambda_D = 525 \text{ nm}$				4%	25%	60%	85%	10%	45%	13,2	16,6	20,0	
		$\lambda_D = 615 \text{ nm}$				80%	110%	0%	14%	5%	24%	15,6	19,5	23,4	

Az érzékelő bele van építve egy nyák lapba lásd (14. ábra), amelynek kapcsolási rajza az (1. mellékletben) látható. A nyáklap VIN bemenetére rákapcsolható az Arduino UNO 5V-os kimeneti feszültsége, mivel a nyák tartalmaz egy RT9193 feszültség szabályzót, amely a működtetéséhez szükséges 3,3V-ra szabályoz. A nyák bemenetei tartalmazzák azokat a lábakat, amelyek össze vannak kötve az érzékelő pin-jeivel. Ezen felül van egy LED is beépítve a nyáklapba, amely megvilágítást tud biztosítani a színérzékeléséhez. A LED lábat a kapcsolás alapján földre kell kötni, ha le akarjuk kapcsolni. A TCS34725 érzékelő nyáklapjához tartozik egy 7 pin-es tűsoros, amelyet beforrasztottam a nyáklapba és ezáltal a breadboardba be tudtam csatlakoztatni lásd (14. ábra)



14. ábra TCS34725 érzékelő nyáklapban bal, és tűkesorral jobb forrás: (saját ábra).

3.1.1. TCS34725-höz tartozó Arduino könyvtár

A TCS34725 vezérléséhez az Adafruit TCS34725 nevű könyvtár volt használva. A sorszámok a (2. melléklet)-ben található programsorokra vonatkoznak. Ennek a könyvtárnak a segítségével meg tudjuk kapni a fotodiódák értékeit, valamint be tudjuk állítani az érzékelő integrációs idejét, valamint az erősítési szorzót. Ezek előre definiált értékek, így a könyvtárból kell kinézni milyen értékek írhatók be. A programban 50ms-os integrálási idő és 4-szeres erősítő tényező van beállítva. A 6. sorban adjuk meg ezen értékeket valamint, hogy hogyan hivatkozunk a programon belül erre az érzékelőre (a programban tcs-ként fogunk hivatkozni rá).

A `tcs.begin()` parancs indítja el az adatátvitelt az Arduino és az érzékelő között. Ez a függvény egy `TRUE` értéket ad vissza, ha felállt a kapcsolat. A `setup-on` belül inicializáljuk az érzékelőt, és ha a `tcs.begin()` `FALSE` értéket ad vissza, akkor értesít minket a program, hogy nem sikerült felállítani a kapcsolatot és végtelen `while` ciklusba teszi a programot, tehát megállítja. Ha ezt az üzenetet kapjuk, akkor érdemes leellenőrizni a bekötést.

A `tcs.getRawData()` parancs írhatja ki az érzékelő R, G, B és C értékeit. Ezekről az értékekről bővebben (3.1)-es fejezetben. Ennek a parancsnak 4 argumentuma van, amelyet meg kell adni. Erre láthatunk példát a 112. sorban. Az első argumentum az R-re, a második a G-re, a harmadik a B-re és a negyedik a C értékre vonatkozik. Azok a változók elé, amelyekbe be akarjuk írni az értéket egy `&` jelet kell rakni. Ennek miértjéről bővebben a (3.2.1)-es fejezet `getRGB255` részénél olvashatunk. A parancs argumentuma `uint16_t` típusú változót fogad csak el.

3.2. Elkészített Arduino Program bemutatása

Az Arduino feladata, hogy az érzékelőt működtesse, az ebből kapott értékeket feldolgozza és a feldolgozásból kapott eredményeket kiküldje a pin-jeire. Mivel az Arduino 5 Voltos DC jelet ad ki a pin-jein a Fanuc robotkar IO modulja pedig 24 Voltos DC feszültségen

működik, így szükség volt az 5V-os jelet 24V-os jelre illeszteni. Erről bővebben a (3.3)-as fejezetben. A teljes program a (2. mellékletben) található.

Három könyvtárat kell behívni az `#include <könyvtárneve.h>` paranccsal. A `Wire.h`-t, amely az Arduino kivezetéseit kezeli, az `Adafruit_TCS34725.h`-t, amely a TCS34725 érzékelőt kezeli és a `MatrixMath.h`-t, amellyel mátrix műveleteket lehet végezni. Ezt követően inicializáljuk az érzékelőt két alapértékkel, amelyről bővebben a (3.1.1)-es fejezet taglal. A programban a 8. sornál a `#define` parancsot használtam, hogy a későbbiekben egyszerűbb legyen a pin váltás. A `#define` az azt csinálja, hogy lecseréli az ezt követő szót az azutáni szóra. A szövegben `BIT_0` és `BIT_1` hivatkoztam a pinekre tehát a programomban, az összes helyen, ahol `BIT_0` használtam azt lecseréli a 13-as számra. Változók résznél 12. sor `uint16_t` típusban adtam meg azokat a változókat, amelyekbe az érzékelő értékeit írom bele. A számításokhoz használt RGB változókat a pontosabb számítás érdekében `float` típusba tettem, amely képes tizedes pontossággal kiírni az értékeket 32 biten, szemben a többi változóval, amely egész számokat írnak ki.

A keresendő színértékeknek létrehoztam egy struktúrált adattípust `PredefColor` néven lásd: (15. ábra), amely tartalmazza azt a számtömböt, amelyben a keresendő szín RGB értékét tárolja, a színnek a nevét, valamint a színértékhez tartozó min-max színtartományt, amelyről a későbbiekben részletesebben lesz kifejtve. A `CNUM` határozza meg, hogy hány színt tud ilyen formában eltárolni. Fontos, hogy ennek a `CNUM` értéknek mindig nagyobb vagy egyenlőnek kell lennie, mint ahány színt keresni akarunk. Mivel a struktúrált adattípus méretének meghatározásakor fix értéket tud csak elfogadni, ezért alkalmaztam a `#define`-os megoldást. A

```

18 // Keresendő színek
19 //Előre meghatározott színek száma
20 #define CNUM 5
21 //Struktúrált datatípus
22 struct PredefColor {
23     float colorvalues[3]; // 1.R - piros; 2.G - zöld; 3.B - kék
24     const char* colorname;
25     float colorrange[2][3]; //[0][rgb] min / [1][rgb] max
26 };
27 //Színmegadás !!MINDIG ELLENŐRIZD LE HOGY A CNUM NAGYOBB VAGY EGYENLŐ LEGYEN
28 //AZ ITT MEGHATÁROZOTT SZÍNEK SZÁMÁVAL!!
29 // A nevek különbözzenek!
30 // Piros | Zöld | Kék | színnév
31 PredefColor Pcolor[CNUM] = {
32     { 33.0, 171.0, 57.0, "Zöld" },
33     { 140.0, 115.0, 60.0, "Fekete" },
34     { 198.0, 64.0, 14.0, "Piros1" },
35     { 235.0, 44.0, 5.0, "Piros2" },
36     { 64.0, 85.0, 122.0, "Kék" }
37 };
38 //

```

15. ábra `PredefColor` struktúrált adattípus definiálása forrás: (saját ábra).

PredefColor az adattípus nevét jelöli, ha használni akarjuk, akkor adni kell a változónak is egy nevet, amely ezt az adattípust használja fel. Esetemben én ezt a változót Pcolor-nak neveztem el és ebbe lettek beleírva az értékek, ahogyan a (15. ábra) is mutatja.

3.2.1. Létrehozott függvények leírása

ColorRangeMaker(float minV, float maxV) (16. ábra). Ez a függvény a Pcolor-ban tárolt összes színhez tartozó színtkomponens értékekhez készít egy minimum értéket és egy maximum értéket, amely meghatározza azt a tartományt, amelybe a talált színérték komponensének bele kell esnie, ahhoz hogy igazat adjon vissza az összehasonlítás során. Két argumentuma van. Az első a minimum értéket, azaz mennyivel legyen kevesebb az adott értékhez képest. A második a maximum értéket, azaz mennyivel legyen nagyobb az adott értékhez képest. Ezek float adattípusban vannak megadva, hogy tizedes pontossággal lehessen dolgozni. A függvény két for ciklussal nézi át a Pcolor-t. Van egy bool változó a végén, amely visszaadja, hogy le lett-e futtatva ez a függvény vagy sem. Ez az érték a prtPcolor() függvényben használtam fel.

```

40 void ColorRangeMaker(float minV, float maxV) {
41     // Elkészíti a színhez tartozó RGB értékek min-max tartományát
42     for (int j = 0; j < CNUM; j++) {
43         for (int i = 0; i < 3; i++) {
44             Pcolor[j].colorrange[0][i] = Pcolor[j].colorvalues[i] - minV;
45             Pcolor[j].colorrange[1][i] = Pcolor[j].colorvalues[i] + maxV;
46         }
47     }
48     rangesmade = true;
49 }

```

16. ábra ColorRangeMaker függvény forrás: (saját ábra).

FindColor(String colorname) (17. ábra). Egy argumentuma van, amelynél szövegesen "szín neve" kell megadni, hogy az általunk meghatározott színek közül melyiket keresse meg. A függvény az általunk megadott színnév alapján kikeresi a Pcolor-ból, hogy melyik index értéken van a színünk, majd ezután megnézi az érzékelő által kapott 3 színtkomponens értékét, hogy beleesik-e a hozzátartozó színtartományokba. Amennyiben az R, G és B színtkomponens tartományon belül esik, akkor a függvény IGAZ értéket ad vissza, ezzel jelezve, hogy azt az általunk megadott színt érzékeli éppen a színérzékelő.

```

51 bool FindColor(String colname) {
52     int index;
53     for (int i = 0; i < CNUM; i++) {
54         if (colname == Pcolor[i].colname) {
55             index = i;
56         }
57     }
58
59     bool r = (Pcolor[index].colorrange[0][0] < Red) && (Red < Pcolor[index].colorrange[1][0]);
60     bool g = (Pcolor[index].colorrange[0][1] < Green) && (Green < Pcolor[index].colorrange[1][1]);
61     bool b = (Pcolor[index].colorrange[0][2] < Blue) && (Blue < Pcolor[index].colorrange[1][2]);
62
63     if (r && g && b) {
64         return true;
65     } else {
66         return false;
67     }
68 }

```

17. ábra FindColor függvény forrás: (saját ábra).

`prtColor()` (18. ábra). Ez a függvény lényegében csak ellenőrzés céljából készült, hogy le tudjam ellenőrizni, hogy megtörtént-e a tartományok elkészítése. Egy for ciklusban kiírja az összes általunk meghatározott szín nevét RGB értékeit, valamint azokhoz az értékekhez tartozó min-max értéket is. Másik feladata még, hogyha valamilyen oknál fogva nem futott le, vagy nem lett meghívva, akkor jelezze, hogy a `ColorRangeMaker()` függvénnyel gond történt.

```

70 void prtPcolor() {
71     // Kiíratja a színnek értékét, nevét, tartományát tartalmazó tömb értékeit
72     if (rangesmade) {
73         for (int j = 0; j < CNUM; j++) { ...
74     }
75     } else {
76         Serial.print("Nem lett elkészítve a színtartományok.
77         | | | | | Kérlek ellenőrizd le a programot, hogy meg van-e hívva a ColorRangeMaker()");
78         while (1)
79             ;
80     }
81 }
82 }

```

18. ábra prtColor függvény forrás: (saját ábra).

getRGB255(float* R, float* G, float* B, int print, int adv) (19. ábra). Itt történik az érzékelő értékeinek kiolvasása és azok feldolgozása. A tcs.getRawData() parancsról már szó esett a (3.1.1) fejezetben. Ha a változó elé & jelet teszünk, akkor nem a változó értékét, hanem a memória címét kapjuk meg, ezáltal közvetlenebb módon tudunk memória címbe írni, ahogy láthatjuk a (19. ábra). Ez szükséges ahhoz, hogy egy függvény argumentumait kimenetként tudjuk használni, amikor meghívjuk. A függvény megadásánál meg csillagozni kell. Ha csillagot teszünk a változó elé azzal a változó memória címén lévő értéket olvassuk ki. Lényegében nincs különbség, ha egy változónál simán vagy a memória címén lévő értéket olvassuk ki mivel a kettő megegyezik, hanem a különbség a program lefutásakor jelenik meg mivel a kettő megközelítést másképp kezeli. Összegezve, ha kimenetet akarunk csinálni az argumentumból, akkor függvény megadáskor csillagot kell a változó elé tenni, meghíváskor &-et.

```

106 void getRGB255(float* R, float* G, float* B, int print, int adv) {
107     // R, G, B - beírandó értékek; print - 0 ne írjon ki, 1 RGB értékek,
108     // 2 Mátrix értékek advance módban;
109     // adv - Számítási módok kiválasztása 0 egyszerű 255, 1 mátrix számítás.
110     // Ez a függvény kinyeri az RGB értékeket a szenzorból és normalizálja RGB255-ös értékekre
111     // RGBC értékek kinyerése a szenzorból
112     tcs.getRawData(&red_raw, &green_raw, &blue_raw, &clear_raw);
113
114 > // A nyers értékek átváltása RGB255-be...
115 // értékek kiírása debug céljából
116 if (print > 0) {
117     Serial.print("RED: ");
118     Serial.print(Red);
119     Serial.print(" GREEN: ");
120     Serial.print(Green);
121     Serial.print(" BLUE: ");
122     Serial.println(Blue);
123 }
124 }

```

19. ábra getRGB255 függvény forrás: (saját ábra).

A függvény print argumentumánál egy számértéket kell megadni. Ha 0-t írunk, akkor nem ír ki semmit a Serial.print-en, ha 1-et akkor kiírja a feldolgozott RGB értéket, ha pedig 2-t akkor még kiírja a számított mátrixokat. Az adv argumentummal azt adjuk meg, hogy a szimpla számítást végezze el lásd (20. ábra), ahol csak arányosítja az RGB értékeket a clear value-hoz és korlátozza a végeredményt 0 és 255 közé, vagy a mátrixos megközelítésű számítást végezze el.

```

181     } else {
182         *R = constrain(((float)red_raw * 255 / (float)clear_raw), 0, 255);
183         *G = constrain(((float)green_raw * 255 / (float)clear_raw), 0, 255);
184         *B = constrain(((float)blue_raw * 255 / (float)clear_raw), 0, 255);
185     }

```

20. ábra getRGB255 függvény adv=0 esetén forrás: (saját ábra).

A mátrixos megközelítésnél figyelembe veszem a (13. ábra) látható átfedéseket. Ez azt jelenti, hogy az érzékelőből kapott R, G, és B értékek valamilyen nagyobb arányban tartalmazzák a velük azonos színtkomponenst, valamint a valamilyen kisebb arányban tartalmazzák a többi színtkomponenst is. Ezért a pontosabb mérés érdekében szükséges az átfedések okán belekerült színtkomponensek eltávolítását. Ez alapján felírtam egy egyenletrendszert.

$$R = x_r r + x_g g + x_b b \quad (10)$$

$$G = y_r r + y_g g + y_b b \quad (11)$$

$$B = z_r r + z_g g + z_b b \quad (12)$$

Tehát mindegyik nagybetűs RGB érték felírható a valós kisbetűs rgb értékek szorzótényezővel ellátott összegével. Ahhoz, hogy megkapjuk a valós rgb értékeket át kell rendeznünk az egyenletrendszerünket úgy, hogy a bal oldalán szerepeljenek az rgb értékek. Ehhez átírtam mátrixos formában az egyenletrendszert, hogy könnyebben lehessen vele dolgozni.

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} x_r & x_g & x_b \\ y_r & y_g & y_b \\ z_r & z_g & z_b \end{bmatrix} \begin{bmatrix} r \\ g \\ b \end{bmatrix} \quad (13)$$

Ezután át kell rendeznünk a mátrixos képletünket $\begin{bmatrix} r \\ g \\ b \end{bmatrix}$ -re. Ehhez a következő azonosságot használjuk: A, B és C mátrixokat jelölnek

$$A = B \cdot C \quad (14)$$

$$B^{-1} \cdot A = B^{-1} \cdot B \cdot C \quad (15)$$

$$B^{-1} \cdot B = I \quad (16)$$

$$B^{-1} \cdot A = I \cdot C = C \quad (17)$$

$$C = B^{-1} \cdot A \quad (18)$$

Ez alapján átrendezve megkapjuk ezt a képletet.

$$\begin{bmatrix} r \\ g \\ b \end{bmatrix} = \begin{bmatrix} x_r & x_g & x_b \\ y_r & y_g & y_b \\ z_r & z_g & z_b \end{bmatrix}^{-1} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad (19)$$

A valós értékek kinyerésének érdekében az érzékelőből érkező RGB adatokat be kell szorozni a szorzótényezős mátrix inverzével. A számítások elvégzésére az Arduinóban a MatrixMath könyvtárat fogom használni. Mátrix megadásánál a neve elé mindenesetben oda kell írni a `mtx_type` szöveget, valamint az argumentum beírásnál (`mtx_type*`)-ot. A könyvtárból használt parancsok argumentumait a következőképpen kell megadni: `Matrix.Copy` esetén meg kell adni a másolandó mátrixot, annak sorának és oszlopának értékét, majd azt a mátrixot, amibe átmásoljuk az értékeket. `Matrix.Invert` esetén az invertálandó mátrixot kell, valamint sorainak számát, aminek meg kell egyeznie az oszlopainak számával, hogy invertálható legyen. A `Matrix.Multiply` esetén meg kell adni a szorzó mátrixot, a szorzandó mátrixot, a szorzó mátrix sorainak számát, a szorzó mátrix oszlopainak számát, amely egyenlő a szorzandó mátrix sorainak számával, a szorzandó mátrix oszlopainak számát és végül azt a mátrixot, amelybe az eredmény kerül.

Létrehoztam egy mátrixot az érzékelőből kapott RGB értékeknek, a szorzótényező értékeinek, az ehhez tartozó inverz értékeknek és egyet a valós rgb értékeknek. A `Matrix.Copy` paranccsal átírtam a szorzótényezős mátrix értékeit az inverzének kijelölt mátrixba, majd ezután azt bevitettem a `Matrix.Invert` parancsba, ahol megkaptam az inverz mátrixot és ezt összesoroztam az RGB mátrixszal, így megkapva a valós értékű rgb mátrixot. Ezt láthatjuk a (21. ábra). A szorzótényezők értékeit a (3. táblázat) határoztam meg. Mivel nem rendelkezem a gyártó által végzett mérések adataival, csak a végeredménnyel, ezért átlagot vontam a min-max értékekből és ezeket írtam be a szorzó tényezőknek. Mivel egy színtkomponenshez tartozó szorzótényezők összege meghaladja a 100%-ot így ezeket még normalizálnom kellett, hogy az

összérték egyszeres szorzót kapjon. Ezt történik a 122-131 sorban. Mátrixos számítás során pontosabb értékeket kaptam, mintha csak szimplán arányosítottam volna a tiszta értékhez az érzékelőből kapott RGB értékeket.

```

115 // Az RGB fotódiódák RGB komponensek arányai
116 float x1 = 80, x2 = 14, x3 = 7,
117       xr, xg, xb,
118       y1 = 7, y2 = 72, y3 = 26,
119       yr, yg, yb,
120       z1 = 14, z2 = 27, z3 = 82,
121       zr, zg, zb;
122 // Ezen komponensek arányosítása
123 xr = x1 / (x1 + x2 + x3);
124 yr = y1 / (y1 + y2 + y3);
125 zr = z1 / (z1 + z2 + z3);
126 xg = x2 / (x1 + x2 + x3);
127 yg = y2 / (y1 + y2 + y3);
128 zg = z2 / (z1 + z2 + z3);
129 xb = x3 / (x1 + x2 + x3);
130 yb = y3 / (y1 + y2 + y3);
131 zb = z3 / (z1 + z2 + z3);
132 // A számolt RGB érték vektor egyenlő a valós rgb vektor és az xyz szorzótényezős mátrix
133 // mátrix szorzatával. Ha ebből ki akarjuk fejezni az rgb vektort, akkor az xyz mátrix
134 // inverzével be kell szorozni az RGB vektort
135 // Mátrixok létrehozása
136 mtx_type xyz[3][3] = {
137     { xr, xg, xb },
138     { yr, yg, yb },
139     { zr, zg, zb }
140 };
141 mtx_type RGB[3] = { red_raw, green_raw, blue_raw };
142 mtx_type rgb[3];
143 // Mátrix műveletek
144 // inverz mátrix helye
145 mtx_type xyz_inv[3][3];
146 // inverzálandó mátrix értékeinek behelyezése az inv mátrix helyére
147 Matrix.Copy((mtx_type*)xyz, 3, 3, (mtx_type*)xyz_inv);
148 // inv mátrix inverzálása
149 Matrix.Invert((mtx_type*)xyz_inv, 3);
150 // inv és RGB mátrix szorzása
151 Matrix.Multiply((mtx_type*)xyz_inv, (mtx_type*)RGB, 3, 3, 1, (mtx_type*)rgb);

```

21. ábra getRGB255 mátrix számításai forrás: (saját ábra).

pinColor2bit() (22. ábra). Ebben a függvényben a FindColor függvény segítségével összehasonlítjuk az érzékelt színértékeket a beírt színekkel. Az értékek kiírása 2 pinen történik. A (4. táblázat) alapján beállítjuk a pinek értékeit, valamint kiíratjuk, hogy melyik színt találta meg. Alapvetően a 3 felé szortírozás elegendő, de ha a későbbiekben bővítésre van szükség, akkor az könnyen megoldható.

4. táblázat pinColor2bit igazságtábla forrás: (saját táblázat)t

b0	b1	eredmény
0	0	nem talált
0	1	piros
1	0	zöld
1	1	kék

```

199 void pinColor2bit() {
200     // 2 biten kerül át 3db szín kiválasztásának eredménye.
201     // A pinek egy tranzisztort nyitnak negált módban.
202     if (FindColor("Zöld")) {
203         b0 = 0;
204         b1 = 1;
205         Serial.print("Zöld szín találva");
206     } else if (FindColor("Piros1") || FindColor("Piros2")) {
207         b0 = 1;
208         b1 = 0;
209         Serial.print("Piros szín találva");
210     } else if (FindColor("Kék")) {
211         b0 = 1;
212         b1 = 1;
213         Serial.print("Kék szín találva");
214     } else {
215         b0 = 0;
216         b1 = 0;
217         Serial.println("Nem találtam színt");
218     }
219 }

```

22. ábra pinColor2bit függvény forrás: (saját ábra).

A setup rész csak egyszer fut le (23. ábra). Itt indítom el a Serial kommunikációt, amely az Arduino és a gép között kommunikál és beállítom annak sebességét a Serial.begin() paranccsal. Megadom, hogy mely pinek legyenek a kimenetek. Esetemben a 12 és 13-as. Lefuttatom a tcs.begin() parancsot és kiíratom, hogy sikeres volt-e, avagy sem a színszenzor inicializálása. Amennyiben sikertelen, akkor egy végtelen while ciklusba teszi magát a program, azaz megáll. Ilyen esetben érdemes leellenőrizni a vezetékelést. Itt történik meg a ColorRangeMaker és a prtColor függvények meghívása.

```

221 void setup() {
222   // beállítások
223   Serial.begin(9600);
224   pinMode(BIT_0, OUTPUT);
225   pinMode(BIT_1, OUTPUT);
226   // Szenzor inicializálás
227   if (tcs.begin()) {
228     Serial.println("TCS34725 csatlakoztatva és sikeresen inicializálva");
229   } else {
230     Serial.println("TCS34725-t nem sikerült megtalálni. Vezetékek ellenőrzése szükséges");
231     while (1)
232       ; // Program leállítása, ha nem talált szenzort
233   }
234   // Színtartományok megadása
235   ColorRangeMaker(15.0, 15.0);
236   prtPcolor();
237 }
238

```

23. ábra setup rész forrás: (saját ábra).

A loop rész (24. ábra) folyamatosan fut. Itt futnak a fő folyamatok. delay() helyett millis()-es megoldást használok, mivel a delay() esetén csak azután kezd el hogy az leszámolt, így több idő telik el két ismétlés között, mintha a millis() módszert használnám. Az utóbbi módszernél nem állítódik meg a program a számolás közben, hanem fut tovább és amikor a millis() nagyobb a megadott értékkel a CurrentMillis-hez képest, akkor futtatja le az IF-en belüli részt. Ezen belül írom ki a pinekre az értékeiket, olvasom ki az érzékelő értékeit és futtatom a pinColor2bit függvényt.

```

239 void loop() {
240   if ((millis() - CurrentMillis) > 100) {
241     CurrentMillis = millis();
242     // Pinekre kiírás. Tranzisztoros kapcsolás miatt negálva vannak az értékek
243     digitalWrite(BIT_0, b0);
244     digitalWrite(BIT_1, b1);
245     // Színértékek kinyerése
246     getRGB255(&Red, &Green, &Blue, 0, 1);
247     // Pinre való kiírás
248     pinColor2bit();
249   }
250 }

```

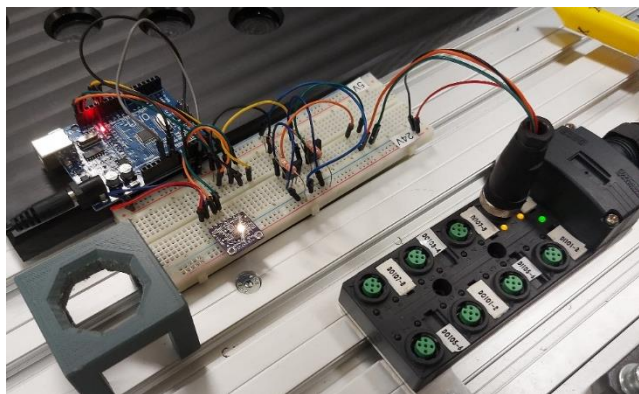
24. ábra loop rész forrás: (saját ábra).

3.3. Arduino-Fanuc Összekötése

A (3. mellékletben) található az ehhez készített kapcsolási rajz. A TCS VIN lábát az Arduino UNO 5 Voltos kimenetére kötöttem. A GND lábát a közös földre kötöttem. Ezután összekötöttem a TCS SDA, SCL lábát összekapcsolom az azonos elnevezésű Arduino pinjeivel. Ahhoz hogy az Arduino 5 Voltos jelét értelmezni tudja a Fanuc 24 Voltos feszültségen működő IO modulja, ahhoz létre kell hozni egy tranzisztoros kapcsolást, ahol az Arduino 5 Voltos jele nyitja a tranzisztort, hogy azon át tudjon folyni a 24 Voltos áram. Ezért az Arduino D12 és D13 lábát rákötöm a két tranzisztor bázisára úgy, hogy sorba kötök velük egy 180 Ohmos előtét ellenállás. Erre azért van szükség, hogy le korlátozzuk a bázis-emitteren folyó áramot. A tranzisztor lábkiosztását a (BD135 adatlap) alapján néztem ki. A collector lábára csatlakoztatom

a Fanuc 24 Voltos kimenetét úgy, hogy sorba kötök egy 820 Ohmos felhúzó ellenállást. Ennek az ellenállásnak lesz a feladata, hogy stabil jelszintet biztosítson, valamint kiszűrje az esetleges zajokat. Itt fontos volt a megfelelő érték kiválasztása. Ha túl nagyot választunk, akkor túl nagy feszültség fog esni rajta és így nem fogja tudni értelmezni azt a jelszintet a Fanuc IO modulja. Ha túl kicsi, akkor meg nem fog tudni rendes jelszintet biztosítani. A rajta eső feszültség nem haladhatja meg a 12 Voltot, mivel a várt feszültségszinttartomány 12-24V között van. A 820Ohmos ellenállás érték tökéletesen megfelel erre a használatra. Az emitter lábát a közös földre kötöm és a breadboard két oldalán lévő negatív sínjét összekötöm egy vezetékkel, így érve el, hogy minden közös földön legyen. Mivel a LED lámpát használom, így annak a LED kivezetését nem kötöm földre, mivel az a kikapcsolását eredményezné.

A tranzisztoros kapcsolás miatt az Arduino által kiadott értékek negálva lesznek, hiszen a tranzisztor nyitott állapotban fog alacsony jelet adni, míg zárt állapotban magast. A második negálást, a robotkar programjában kell megcsinálni, hogyha leáll az Arduino és nem ad jelet a tranzisztorokra, akkor is a (4. táblázat) alapján a nem talált értéket értelmezze és ezzel elkerülve a nem várt működést. Alapvetően a robotkar programját úgy írtam meg, hogy semmilyen körülmények közt ne okozzon komoly problémát, ha rossz jelet ad ki az Arduino. A kapcsolás bekötése a következő (25. ábra) mutatja be.



25. ábra Arduino-Fanuc összekötés forrás: (saját ábra).

3.4. Fanuc oktatócella

Az oktató cella (26. ábra) tartalmazza a Fanuc ER-4iA hatsuklós robotkart, egy a Fanuc iRVision-hez használható szürkeárnyaltos képet adó kamerát, a robot programozásához szükséges Teachpendant-ot, a vezérlő egységet és a robotkar munkaterét elkerítő üvegpanelos rész. Mielőtt AUTO módba raknák le kell rögzíteni a cellát. Ezt megtehetjük a kerekeken elhelyezett fékek benyomásával, valamint a kerekek mellett található támasztó lábak kitékerésével.



26. ábra Fanuc oktatócella forrás: (saját ábra).

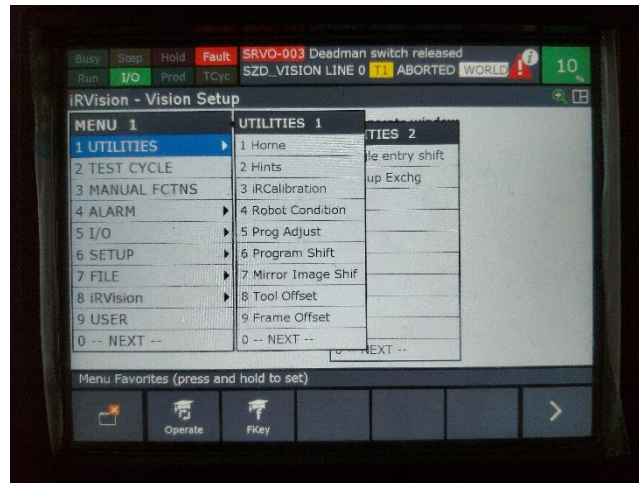
A robotkar lásd (9. b) ábra), ahogyan a nevében is referál rá, 4kg-os munkadarabot tud mozgatni. Pneumatikus vezérlésre is alkalmas, mivel tartalmaz egy pneumatikus bementet és van belül egy beépített vezérlő szelep. A ROBOT IO-kal az endeffektor vezérlését lehet megvalósítani, míg a Digital IO-kal, lásd (9. a) ábra), pedig a munkafolyamatok vezérlését lehet megoldani. A TDK dolgozatom során két Digital Input-ot használtam fel, hogy ezáltal visszajelzést kapjon a munkafolyamatról, hogy el tudja dönteni melyik tárolóra kell mozognia. A robotkar programozása a TP-n történik lásd (9. c) ábra). A TP-vel lehet mozgatni a robotkart, ha a deadman switch be van nyomva, valamint a SHIFT gombot nyomva tartjuk. A deadman switch-nek három állása van, amelyek közül csak a középső állásban lehet mozgatni a robotkart. Ha elengedjük vagy rászorítunk, akkor a robotkar azonnal megáll. Ha finoman akarjuk megállítani a robotkart, akkor azt a HOLD gomb lenyomásával tehetjük meg.

A cella tetején található a kamera, amelyet az iRVison használ, lásd (9. d) árba). Ez a kamera szürkeárnyaltos képet tud készíteni. A lencse része tekerésével állítható be a kamera fókusza. A csíknak a (9. d) árba) látható módon a 0.5 értéken kell állnia a megfelelő fókuszáért.

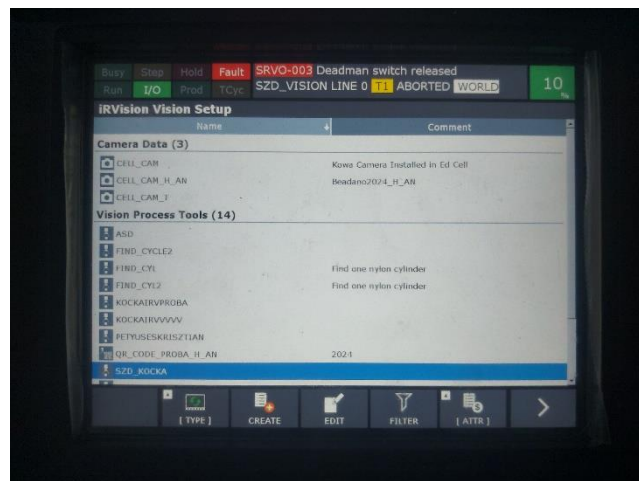
3.4.1. iRVison

A Fanuc iRVison szoftvere felel a cellához tartozó kamera működéséhez, valamint a megkeresendő objektum betanítására és megkereséséhez.

Új objektum tanításakor a következő lépéseket kell végrehajtani (szögletes zárójelben a lenyomandó gombok szerepelnek). [MENU] [8]-al kiválasztom az iRVision-t (27. ábra). Leellenőrzöm a Type [F1]-el, hogy a [1] Vision setup-ban vagyok-e (28. ábra).

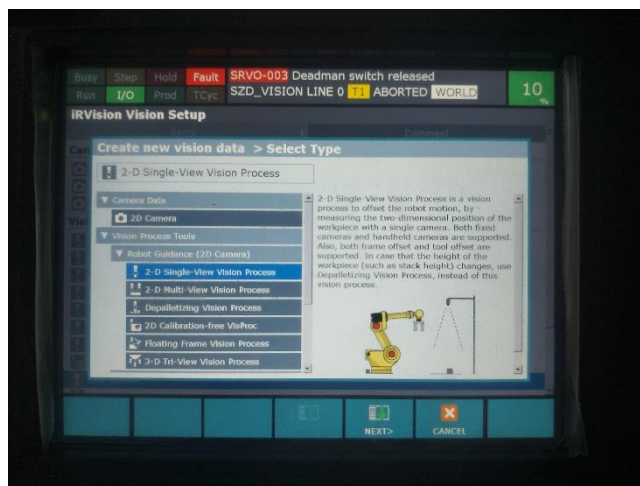


27. ábra iRVision MENU forrás: (saját ábra).

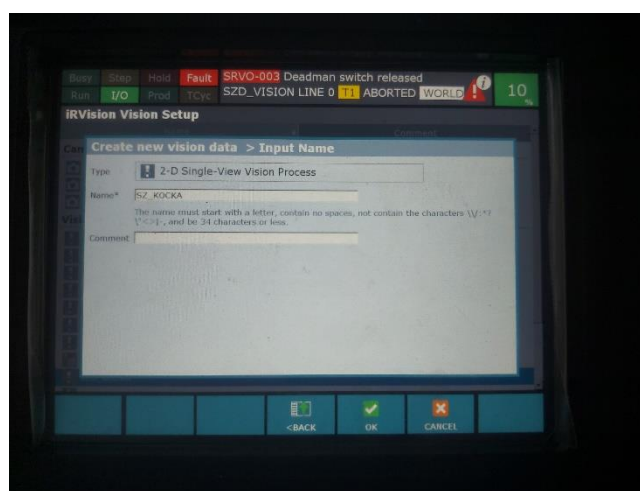


28. ábra iRVision setup forrás: (saját ábra).

Ezután a CREATE [F2] megnyomásával létrehozom a Vision Process Tools-t vagyis képfeldolgozási eszközt. Itt kiválasztom a 2-D Single-View Vision Process-t (29. ábra), mivel csak egy kamerával rendelkezik a cella. (Az egy kamerás rendszerrel lehetőség van a Reader Vision Process kiválasztására is, amelyben QR kódot is lehet beolvasni.) Miután kiválasztottam utána rányomok a NEXT [F4] gombra. Itt elnevezem az eszköz nevét (30. ábra), majd OK-ézzuk le. Dolgozatomban szereplő Vision tool a S2D_KOCKA nevet kapta. Ezt követően bekerült a listába. Itt ki keresem és az EDIT [F3]-re nyomok. Itt megjelenik a balról az Image és Tree fül. A Tree fülre rámenve választom ki a 2-D Single-View Vision Process-t. A jobb oldalt találhatóak a beállítás rész.

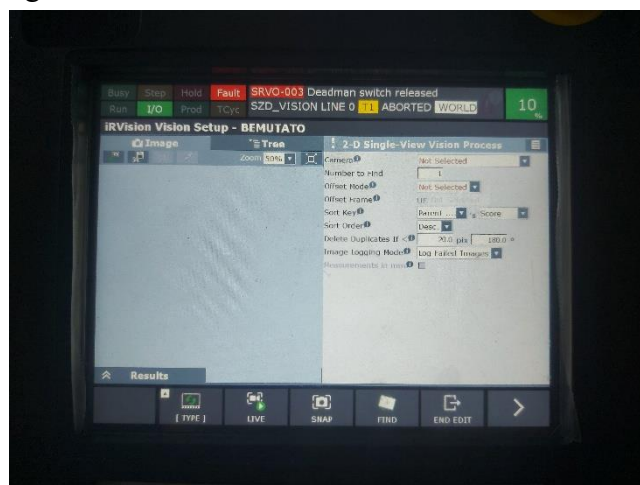


29. ábra Create new vision data forrás: (saját ábra).



30. ábra Vision tool elnevezés forrás: (saját ábra).

A Camera résznél kiválasztom a CELL_CAM opciót (31. ábra). Ezzel már az Image fülön megfog jelenni a kamera képet, ha mégse jelenne meg, akkor a SNAP [F3] gombot megnyomva jelenik meg.



31. ábra 2-D Single-V. beállítás forrás: (saját ábra).

A Number to Find-nál kiválasztom, hogy mennyi találatot keressen. Ha többet talál, akkor a sorba rendezés alapján a legnagyobb pontszámú találatokat helyezi előnybe. A dolgozatom során ezt 10-re állítottam.

Offset Mode-nál a Fixed Frame Offset-et választottam, hogy a kamerának megfelelő módon legyen eltolva a robotkoordináták.

Offset Frame-nél állítom be a kamerához tartozó userframet, amely a 9-es Vision elnevezésű opció lesz. Ez szükséges ahhoz, hogy a robotkar összhangban legyen a kamera által talált pozíciókkal.

A további beállításokon nem kell változtatni, maradhatnak az alapbeállításokon. Ezt követően a Tree fülön kiválasztom a Snap Tool 1-et (32. ábra). Itt lehetőségünk van beállítani a Snap Window-nál a keresendő terület méretét a mellette található Set gomb lenyomásával. Alaphelyzetben ez a teljes képet veszi.

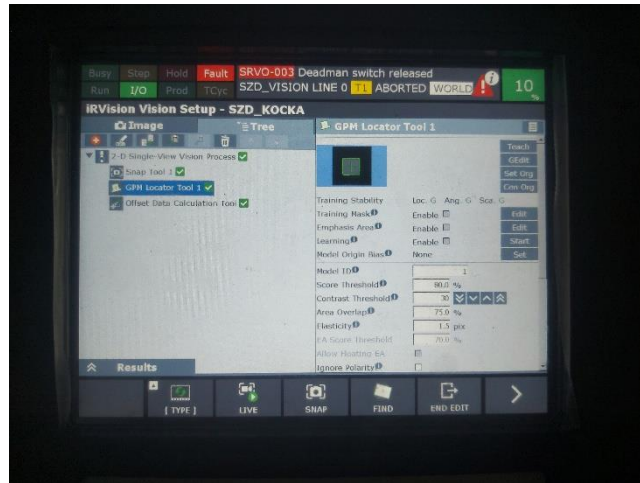


32. ábra Snap Tool 1 forrás: (saját ábra).

Resolution Reduction csökkenti a kép felbontását a gyorsabb feldolgozási időért cserébe. Ezt nem szükséges állítani.

Exposure Mode-nál a Fixed/Auto mód között lehet választani. Auto módban meg kell tanítani neki egy felületet, amihez viszonyítva állítja be fényerőt a képhez.

A többi beállítás maradhat alapbeállításon. Ezután kiválasztom a GPM Locator Tool 1-et (33. ábra). Itt tudom feltanítani az iRVision-nek az munkadarab formáját.



33. ábra GPM Locator Tool 1 forrás: (saját ábra).

Középre helyezem a keresendő munkadarabot a (34. ábra) alapján, majd a Snap [F3] gomb megnyomásával készítsünk egy képet és a Teach gombbal válasszuk ki a képről a munkadarabunkat, amelyet meg akarunk tanítani neki. Ezután automatikusan felismeri a formát és zölddel körbe rajzolja a szélét. Fontos, hogy a munkadarabunk kitűnjön a háttér színéből. A dolgozatomban használt munkadarab fehér volt, így könnyen ki tudott tűnni a fekete háttérből.

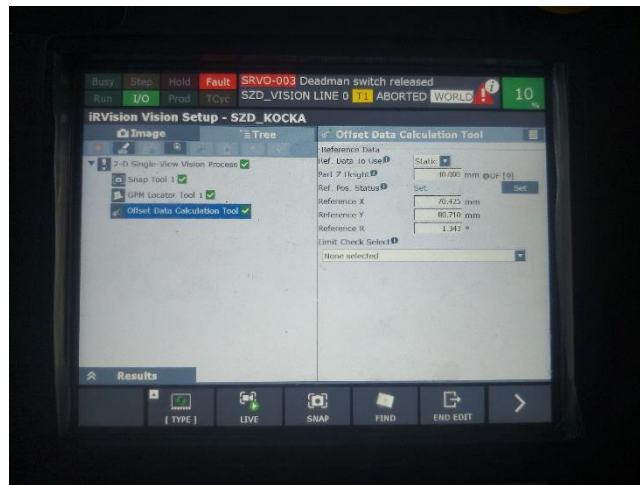


34. ábra Munkadarab kamera középpontjában forrás: (saját ábra).

A Cen Org gombbal automatikusan a betanított felület súlyközpontjára teszi a középpontot.

Ezen felül a Score Threshold és a Contrast Threshold-al kell még, ha nem találja a munkadarabunkat. Ha sötétebb lett a környezeti viszonyok akkor érdemes állítani a kontraszton, ha pedig nagyon eltérőek a keresendő munkadarabok, akkor meg a Score-t kell lejjebb venni.

A Tree fülön az utolsó beállítási lap az Offset Data Calculation Tool 1 (35. ábra), amely kiszámítja a munkadarab koordinátáját a betanított munkadarab alapján.



35. ábra Offset Data Calculation Tool 1 forrás: (saját ábra).

A Ref. Data To Use-nál a Static-ot kell kiválasztani, mivel csak egy referencia értéket fogunk használni.

A Part Z. Height-nál meg kell adni a munkadarabunk magasságát, így a számított koordináta a munkadarab tetején fog meg jelenni a Z tengely mentén.

A Ref. Pos. Status-nál csak rá kell nyomni a Set gombra és, ha nem mozdítottuk el a betett munkadarabunkat, akkor az alapján beméri, hogy hol helyezkedik el a képen és ez által már fog tudni pozíció koordinátákat számolni a munkadarab keresése során. Ha elmozdítottuk, akkor sincs gond, a lényeg, hogy legyen egy Snap fotó, ahol van talált munkadarab a képen. A pontosság érdekében érdemes középre helyezni a munkadarabot, minden ilyen betanítási művelet során.

Ha végig mentünk ezeken a lépéseken, akkor a Vision Process Tool-unk készen áll a használatra. Programozás során fontos beállítani a talált munkadarabra álláskor a 9-es userframe-t, amelyet az Offset Frame-nél beállítottunk.

3.4.2. Fanuc program leírása

A program a (4. melléklet) található. A programkódot kiírtam egy txt fájlba, ezért ez nem a Fanuc programozói környezetet reprezentálja, hanem csak maga a program kódot. Az alaphelyzetre állás résznél egy felugró ablak rákérdez, hogy biztonságosan vissza tud-e térni a robot HOME pozícióba. Amennyiben az operátor úgy látja, hogy igen akkor az OVERRIDE paranccsal felülírom a beállított sebességet és 50%-os sebességen visszamegy HOME pozícióba, ha nem, akkor a program PAUSE-ba megy és nem megy vissza a HOME-ba. Ha folytatni akarjuk a program futását, akkor az FWD gombot kell megnyomni. (36. ábra)

```

1      !A kirakott kockakat szín szerint
2      !szortirozza
3      !keszult:2024.11.11.
4      !keszitette: K.Tibor
5      !verzio:1.0
6
7      !Alaphelyzetre allas
8      Override=50%
9
10     CALL PROMPTYN(3,15)
11     IF R[15:V_yes/no]=1,JMP LBL[2]
12     CALL PROMPTOK(3)
13     PAUSE
14
15     LBL[2]
16     CALL AA_HOME
17
18     !parameter beallitasa
19     UFRAME_NUM=9
20     UTOOL_NUM=1
21     OVERRIDE=100%
22
23     PR[80:beirt poz]=LPOS-LPOS
24     PR[81:talalt poz]=PR[80:beirt poz]
25     PR[82:offset magassag]=PR[80:beirt poz]
26     PR[80,4:beirt poz]=180
27     PR[82,3:offset magassag]=100

```

36. ábra Fanuc program alaphelyzet és paraméterbeállításforrás: (saját ábra).

A paraméterek beállítása résznél beállítom az IRVison-höz használt UFRAME-t és a megfogóhoz tartozó UTOOL-t. Visszaállítom az OVERRIDE-ot 100%. Lenullázom a programban használt pozíció regisztereket. A PR[80]-as pozíció regiszternek a 4. azaz a yaw koordináta értékét beállítom 180°-ra, mivel a későbbi vision-ös pozíció kiolvasásnál erre az értékre 0° ad és ezzel hibára futna ki. A PR[82]-es regiszter lesz a magasság beli eltolásért felelős, ezért neki csak a 3. azaz z koordináta értékét állítom be 100mm-re.

A VISION RUN_FIND paranccsal futtatom a tárgy keresést, majd a VISION GET_NFOUND paranccsal a találatok számát írom ki, itt ki kell választani melyik regiszterbe mentse el az értéket. (37. ábra)

```

29     !IRVISION futtatasa
30     VISION RUN_FIND 'SZD_KOCKA'
31     VISION GET_NFOUND 'SZD_KOCKA' R[16]

```

37. ábra Fanuc program iRVison futtatásforrás: (saját ábra).

A FOR ciklusban fut le a hasábok felvétele, színérzékelése és a tárolóba pakolása. Az előbb meghatározott találatok száma dönti el a FOR-nak a ciklusszámát. Kiíratom a talált pozíció koordinátáját egy pozíció regiszterbe PR[81], majd ezen értékeket egy másik pozíció regiszterbe PR[80]. Erre azért van szükség, mert a PR[81]-be beírt koordináták konfigurációja eltér, és így nem tud abba a pozícióba menni a robotkar, viszont a megadott értékek pedig pontosak, ezért ezeket a koordináta értékeket átírom egy megfelelő konfigurációval rendelkező PR-be. Minden mozgás gyorsulását 50%-ra állítottam be, hogy elkerüljem a robotkar hirtelen sebesség változását és az ebből fakadó cella rángatását. Két féle mozgási műveletet lehet

megadni. J – joint (csukló) mozgás esetén a csuklók legoptimálisabb mozgásával éri el a pozíciót, amely során íves pályát írhat le. L - linear (egyenes) mozgás esetén a mozgás pályája egy egyenes. Ennél a mozgásnál egyes csuklók akár mozgás irányt is válthatnak miközben a mozgás végrehajtódik. Minden esetben a megfogási vagy érzékelési pozícióra állás előtt az Offset,PR paranccsal 100mm-el a pozíció fölé állok Joint mozgással, majd ezt követően Linear mozgással megyek le a pozícióra. A hasáb megfogásakor mielőtt pozícióra megyek, előtte kinyitom a megfogót, majd amikor leértem a hasáb pozíciójára, akkor összezárom a megfogót és várok egy tized másodpercet, hogy a megfogás megtörténjen, majd ezután ismételten fölé megyek a pozíciónak. Miután a színérzékelő pozícióját elérte vár 4 tized másodpercet, hogy legyen ideje lefutni a színérzékelésnek és kiadni az IO-ra az eredményt. Ezután a kapott jel alapján elviszi a megfelelő tárolóba a hasábot. Ha az érzékelő nem tudta megállapítani a tárgy színét, akkor egy a tárolóktól félre eső pozícióba dobja le. Ha végigment az összes talált hasábon, akkor a FOR ciklus után visszamegy a HOME pozícióba és megáll. (38. ábra)

```

33      !FOR ciklus
34      FOR R[20:V_cikluszam]=1 TO R[16:V_found_item]
35      VISION GET_OFFSET 'SZD_KOCKA' VR[1] JMP LBL[1]
36      !talalt poz beirasa
37      PR[81]=VR[1].FOUND_POS[1]
38      PR[80,1:beirt poz]=PR[81,1:talalt poz]
39      PR[80,2:beirt poz]=PR[81,2:talalt poz]
40      PR[80,3:beirt poz]=0
41      PR[80,6:beirt poz]=PR[81,6:talalt poz]
42      PR[83]=VR[1].OFFSET
43      !ra allas es megfogas
44      J PR[80:beirt poz] 25% CNT100 Offset,PR[82:offset magassag] ACC50
45      CALL HAND_OPEN
46      L PR[80:beirt poz] 100mm/sec FINE ACC50
47      WAIT .10(sec)
48      CALL HAND_CLOSE
49      L PR[80:beirt poz] 250mm/sec CNT25 Offset,PR[82:offset magassag] ACC50
50      !szinerzekeles
51      J P[1:szinerzekelo_poz] 25% CNT25 Offset,PR[82:offset magassag] ACC50
52      L P[1:szinerzekelo_poz] 100mm/sec FINE ACC50
53      WAIT .40(sec)
54      !tarolo_1
55      IF (DI[104:ON ] AND !DI[103:ON ]) THEN
56      L P[1:szinerzekelo_poz] 250mm/sec CNT75 ofs ACC50
57      J P[2:piros_tarolo] 25% FINE ACC50
58      CALL HAND_OPEN
59      JMP LBL[3]
60      ENDIF
61
62      !tarolo_2
63      IF (!DI[104:ON ] AND DI[103:ON ]) THEN
64      L P[1:szinerzekelo_poz] 250mm/sec CNT75 ofs ACC50
65      J P[3:zold_tarolo] 25% FINE ACC50
66      CALL HAND_OPEN
67      JMP LBL[3]
68      ENDIF
69
70      !tarolo_3
71      IF (!DI[104:ON ] AND !DI[103:ON ]) THEN
72      L P[1:szinerzekelo_poz] 250mm/sec CNT75 ofs ACC50
73      J P[4:fekete_tarolo] 25% FINE ACC50
74      CALL HAND_OPEN
75      JMP LBL[3]
76      ENDIF
77
78      !nem ertelmezett
79      IF (DI[104:ON ] AND DI[103:ON ]) THEN
80      L P[1:szinerzekelo_poz] 250mm/sec CNT75 ofs ACC50
81      J P[5:nem_ertelmezett] 25% FINE ACC50
82      CALL HAND_OPEN
83      JMP LBL[3]
84      ENDIF
85
86      LBL[3]
87
88      ENDFOR

```

38. ábra Fanuc program FOR ciklusforrás: (saját ábra).

4. EREDMÉNYEK

Ebben a fejezetben részletezni fogom a dolgozatommal kapcsolatosan elért eredményeimet. Az Arduinohoz talált könyvtár segítségével sikeresen rá tudtam csatlakozni a TCS34725-ös színérzékelő modulra és értékeket olvastam ki belőle. A modulba beépített LED által kizártam a környezeti fények mérést befolyásoló hatását. Az Arduino programban használt mátrix számítások a valósághoz közeli értékeket adnak vissza, ezáltal sikerült minimalizálni az átfedésekből fakadó mérési pontatlanságokat. Az előre definiált színtonkomponens értékeket $\pm 5\%$ -os hibatűrési határral láttam el, hogy kiszűrjem a mérést befolyásoló külső hatásokat. A dolgozatom során Piros, Zöld és Kék felfestésű hasábokat használtam, amelyeket a program megfelelően azonosított. Azonban a filctollas megoldás könnyedén tudott kopni, így időközönként újra kellett színezni, viszont jól demonstrálta a nem értelmezhető munkadarab esetén lefutó feladatát a robotkarnak. Tranzisztoros kapcsolással valósítottam meg a jelillesztést az Arduino 5V-os kimenete és a Fanuc 24V-os IO modulja között. A tranzisztoros kapcsolásoknál megfelelő értékű felhúzó ellenállások kerültek beszerelésre, amely a rajta eső feszültség érték a megengedett 12V alatt van, így kapcsolni tudja a Fanuc IO modul Digital Input-jait. A tranzisztor negáló kapcsolásként funkcionál, amelyet a Fanuc programban kezeltem. 250mm/sec-es Linear és 25% Joint-os mozgássebesség beállítása mellett a robotkar a felvételi és szortírozó ciklust 40-50 másodperc alatt hajtotta végre. Az iRVision a felvételi területen kellő pontossággal mérte be a hasábok koordinátáját, hogy a megfogó széle nyitott állapotban ne akadjon bele hasádba.

5. ÖSSZEFOGLALÁS

TDK dolgozatomban során kitértem a robotkarok alapkonfigurációira. Írtam a csuklókaros, gömbi koordinátás, Scara, henger koordinátás, derékszögű koordinátás és delta robotkarokról. Részletesebben kitértem a hatcsuklós robotkar tulajdonságaira. Bemutattam a TCP koordináta kiszámításához használt Denavit-Hartenberg féle transzformációs mátrixot. Felsoroltam az ipari felhasználási területeit a hatcsuklós robotkaroknak. Írtam a képfeldolgozó rendszerek működési elvéről, azaz a kép digitalizálásáról és az ahhoz tartozó képfeldolgozási műveletekről. Megvalósítás során megismerkedtem a TCS34725-ös színszenzor működési elvével. Programkódot írtam az Arduino UNO-ra, amely kiolvasta a színérzékelőből a mért értékeket, majd azt mátrix műveletekkel feldolgoztam a pontosabb eredményért és ezt követően kiírtattam az Arduino kimenetére az eredményt. Munkám során azt tapasztaltam, hogy a nyers értékek nem tükrözik pontosan a mérendő objektum színét. Az adatlapjában szereplő fotodiódák karakterisztikája és táblázatba foglalt mérési eredményei alapján látható volt, hogy átfedések vannak az egyes értékek között, amelynek okán létrehoztam egy képletet, aminek a segítségével képes voltam nagymértékben kiszűrni az átfedésből adódó pontatlanságot. Meg kellett oldanom az Arduino és Fanuc közti kommunikációt. A jelet az Arduino kimenetén adtam ki, amelyet a Fanuc IO modulja fogadott. Az Arduino 5 voltos és a Fanuc 24 voltos jelszintje közötti különbséget egy tranzisztoros kapcsolással oldottam meg. Mivel foglalkoztam korábban a Fanuc robotkar programozásával, így nem jelentett gondot beállítani iRVisiont a hasábok felismerésére. A tapasztalatok azt mutatják, hogy az egy kamerás rendszerek esetén minél messzebb kerül a középponttól a keresendő objektum annál inkább deformálódik a kamera képe, amely pontatlanabb pozíció eredményeket szül. A robotkar mozgásának programját 6 részre lehet bontani: alaphelyzetre állás, inicializálás/paraméter beállítás, iRVision, megfogás, színérzékelés, tárolóba rakás. A megfogási pozíció megtalálás volt, ami nehézségeket tudott okozni, de végül sikerült megtalálni a módszert, amellyel kellő pontossággal kaptam meg a hasábok koordinátáját.

Végeredményben sikerült megoldani azt, hogy az Arduino program sikeresen megtalálja az általunk beállított színt. Ehhez el kellett készítenem a keresendők szín színtkomponenseihez tartozó értékek kiszélesítését, hogy számításba kerüljön az esetleges mérési hibahatárok. Az elkészített tranzisztoros kapcsolat a tesztelesek során megfelelően működött, nem okozott rövidzárlatot, vagy égette ki a tranzisztorokat. Az iRVision betanítása során szükséges volt az ehhez tartozó userframe beállítása, mivel ez alapján van a kamera pozíciói is bemérve. A programban elhelyeztem egy időzítőt, amely a pakolásnak mérte a ciklus idejét. Átlagosan 40-50 mp körül végezte el a feladatát a robotkar 6 hasáb esetén.

5.1. Fejlesztési lehetőségek

A dolgozatom készítése közben felmerültek fejlesztési lehetőségek, amelyek optimalizálást és pontosság növelést eredményeznek. A hasáb felvételi pozíció koordinátájának pontatlanságát csökkentené a teteje és oldala közti nagy kontraszt elérésével, viszont így számításba kell venni későbbi fejlesztések során, hogy így ezen oldalain nem lehet megvizsgálni a színértékét. Gyorsabb ciklus idő érhető el a robotkar pakolási fázisához tartozó mozgások sebességének növelésével, viszont ehhez ki kell tesztelni még, milyen gyorsulási beállítás, valamint milyen pálya lekerekítési értékek mellett nem fog rángatózni a robotcella. Az iRVision-ben ki kell fejleszteni azt az Vision Process Tool-t, amely képes számításba venni a robotkar megfogójának méreteit, hogy csak a szabad oldalán fogja meg minden esetben a munkadarabot a robotkar. Ezáltal lehetőség nyílna egy folyamatos válogató ciklus létrehozására. Az Arduino és annak kapcsolására készített burkolattal fixálni lehetne az oktatócellán belül a kapcsolásunkat. A kapcsolásra tervezett nyáklappal és a kapcsoláshoz tartozó komponensek beforrasztásával rögzíthetők lennének a komponenseket.

IRODALOMJEGYZÉK

BD135 adatlap. (dátum nélk.).

Béla, K. (2012). *Robottechnika I.* Typotex kiadó.

Bonev, I. (2001). *Delta parallel robot-the story of success.* Newsletter.

Czap, L. (2007). *Képfeldolgozás.* Miskolci Egyetem.

Czúni, L., & Tanács, A. (2011). *Képi Információ Mérése.* Typotex.

Dr. Pintér, J. (2017). *Gyártócellák Ipari Robotok.* Győr: Széchenyi István Egyetem.

Guanglei, W. (2013). *Error Modeling and Design Optimization of Parallel Manipulators.* Aalborg, Denmark: Aalborg University.

Hägele, M., Nilsson, K., Pires, J. N., & Bischoff, R. (2016). *Industrial Robotics.* Springer.

Haralick, R. M., & Shapiro, L. G. (1984. Június 25). Image Segmentation Techniques .
COMPUTER VISION, GRAPHICS, AND IMAGE PROCESSING, old.: 100-132.

Hiroshi, M. (2014). *Development of the SCARA.* Makino Automation Research Institute.

Lavin-Delgado, J., Chavez-Vazquez, S., Gomez-Aguilar, J., Delgado-Reyes, G., & Ruiz-Jaimes, M. (2020). Fractional-order passivity-based adaptive controller for a robot manipulator type scara. *Fractals*.

Mester, G. (2011). *Robotika.* Szeged: Typotex.

Nayak, J., Balas, V., N. Favorskaya, M., Choudhury, B. B., Rao, S., & Naik, B. (2019).
Applications of Robotics in Industry Using Advanced Mechanisms. Springer.

Palágyi, K. (2011). *Képfeldolgozás Haladóknak.* Typotex.

Pierrot, F., Dombre, E., Dégoulange, E., Urbain, L., Caron, P., Boudet, S., . . . Mégnien, J.-L. (1999). *Hippocrate: a safe robot arm for medical applications with force.* Medical Image Analysis.

Poole, H. H. (1989). *Fundamentals of Robotics Engineering.* 115 Fifth Avenue New York, New York 10003: Van Nostrand Reinhold.

Pratt, W. K. (2001). *Digital Image Processing (vol. 3).* Wiley-Interscience.

Schreiber, W. F., Huang, T. F., & Tretiak, O. J. (1971). Image Processing. *Proceedings of the IEEE*.

Siciliano, B., Sciavicco, L., Villani, L., & Oriolo, G. (2009). *Robotics Modelling, Planning and Control.* Springer.

TCS3472 adatlap. (dátum nélk.).

Wu, G., & Shen, H. (2021). *Parallel PnP Robots, Parametric Modeling, Performance Evaluation and Design Optimization.* Springer Nature Singapore.

- Yanowitz, S. D., & Bruckstein, A. M. (1989. Szeptember 28). A New Method for Image Segmentation. *COMPUTER VISION, GRAPHICS, AND IMAGE PROCESSING*, old.: 82-95.
- Zhang, D., & Wei, B. (2017). *Mechatronics and Robotics Engineering for Advanced and Intelligent Manufacturing*. Svájc: Springer .



2. melléklet: Arduino program

Alapértékek megadása (1-39)

```

1  #include <Wire.h>
2  #include <Adafruit_TCS34725.h>
3  #include <MatrixMath.h>
4
5  // TCS34725 inicializálása
6  Adafruit_TCS34725 tcs = Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_50MS, TCS34725_GAIN_4X);
7
8  // Pinek kiválasztása
9  #define BIT_0 13
10 #define BIT_1 12
11
12 // Változók
13 uint16_t red_raw, green_raw, blue_raw, clear_raw;
14 float Red, Green, Blue;
15 bool b0, b1, rangesmade;
16 unsigned long CurrentMillis;
17
18 // Keresendő színek
19 //Előre meghatározott színek száma
20 #define CNUM 5
21 //Struktúrált datatípus
22 struct PredefColor {
23     float colorvalues[3]; // 1.R - piros; 2.G - zöld; 3.B - kék
24     const char* colorname;
25     float colorrange[2][3]; //[0][rgb] min / [1][rgb] max
26 };
27 //Színmegadás !!MINDIG ELLENŐRÍZD LE HOGY A CNUM NAGYOBB VAGY EGYENLŐ LEGYEN
28 //AZ ITT MEGHATÁROZOTT SZÍNEK SZÁMÁVAL!!
29 // A nevek különbözzenek!
30 // Piros | Zöld | Kék | színnév
31 PredefColor Pcolor[CNUM] = {
32     { 33.0, 171.0, 57.0, "Zöld" },
33     { 140.0, 115.0, 60.0, "Fekete" },
34     { 198.0, 64.0, 14.0, "Piros1" },
35     { 235.0, 44.0, 5.0, "Piros2" },
36     { 64.0, 85.0, 122.0, "Kék" }
37 };
38 //
39

```

Tartomány készítő (40-50)

```

40 void ColorRangeMaker(float minV, float maxV) {
41     // Elkészíti a színhez tartozó RGB értékek min-max tartományát
42     for (int j = 0; j < CNUM; j++) {
43         for (int i = 0; i < 3; i++) {
44             Pcolor[j].colorrange[0][i] = Pcolor[j].colorvalues[i] - minV;
45             Pcolor[j].colorrange[1][i] = Pcolor[j].colorvalues[i] + maxV;
46         }
47     }
48     rangesmade = true;
49 }
50

```

Színt kereső függvény (51-69)

```

51 bool FindColor(String colorname) {
52     int index;
53     for (int i = 0; i < CNUM; i++) {
54         if (colorname == Pcolor[i].colorname) {
55             index = i;
56         }
57     }
58
59     bool r = (Pcolor[index].colorrange[0][0] < Red) && (Red < Pcolor[index].colorrange[1][0]);
60     bool g = (Pcolor[index].colorrange[0][1] < Green) && (Green < Pcolor[index].colorrange[1][1]);
61     bool b = (Pcolor[index].colorrange[0][2] < Blue) && (Blue < Pcolor[index].colorrange[1][2]);
62
63     if (r && g && b) {
64         return true;
65     } else {
66         return false;
67     }
68 }
69

```

Színértékek kiírása függvény (70-105)

```

70 void prtPcolor() {
71     // Kiíratja a színnek értékét, nevét, tartományát tartalmazó tömb értékeit
72     if (rangesmade) {
73         for (int j = 0; j < CNUM; j++) {
74             Serial.print(Pcolor[j].colorname);
75             Serial.println(":");
76             Serial.print("RED: ");
77             Serial.print(Pcolor[j].colorrange[0][0]);
78             Serial.print("<");
79             Serial.print(Pcolor[j].colorvalues[0]);
80             Serial.print("<");
81             Serial.println(Pcolor[j].colorrange[1][0]);
82             Serial.println();
83             Serial.print("GREEN: ");
84             Serial.print(Pcolor[j].colorrange[0][1]);
85             Serial.print("<");
86             Serial.print(Pcolor[j].colorvalues[1]);
87             Serial.print("<");
88             Serial.println(Pcolor[j].colorrange[1][1]);
89             Serial.println();
90             Serial.print("BLUE: ");
91             Serial.print(Pcolor[j].colorrange[0][2]);
92             Serial.print("<");
93             Serial.print(Pcolor[j].colorvalues[2]);
94             Serial.print("<");
95             Serial.println(Pcolor[j].colorrange[1][2]);
96             Serial.println();
97         }
98     } else {
99         Serial.print("Nem lett elkészítve a színtartományok.
100         | | | | | Kérlek ellenőrizd le a programot, hogy meg van-e hívva a ColorRangeMaker()");
101         while (1)
102             ;
103     }
104 }
105

```

Színérték kinyerő függvény (106-199)

```

106 void getRGB255(float* R, float* G, float* B, int print, int adv) {
107     // R, G, B - beírandó értékek; print - 0 ne írjon ki, 1 RGB értékek,
108     // 2 Mátrix értékek advance módban;
109     // adv - Számítási módok kiválasztása 0 egyszerű 255, 1 mátrix számítás.
110     // Ez a függvény kinyeri az RGB értékeket a szenzorból és normalizálja RGB255-ös értékekre
111     // RGBC értékek kinyerése a szenzorból
112     tcs.getRawData(&red_raw, &green_raw, &blue_raw, &clear_raw);
113
114     // A nyers értékek átváltása RGB255-be
115     if (adv == 1) {
116         // Az RGB fotódiódák RGB komponensek arányai
117         float x1 = 80, x2 = 14, x3 = 7,
118             xr, xg, xb,
119             y1 = 7, y2 = 72, y3 = 26,
120             yr, yg, yb,
121             z1 = 14, z2 = 27, z3 = 82,
122             zr, zg, zb;
123         // Ezen komponensek arányosítása
124         xr = x1 / (x1 + x2 + x3);
125         yr = y1 / (y1 + y2 + y3);
126         zr = z1 / (z1 + z2 + z3);
127         xg = x2 / (x1 + x2 + x3);
128         yg = y2 / (y1 + y2 + y3);
129         zg = z2 / (z1 + z2 + z3);
130         xb = x3 / (x1 + x2 + x3);
131         yb = y3 / (y1 + y2 + y3);
132         zb = z3 / (z1 + z2 + z3);
133         // A számolt RGB érték vektor egyenlő a valós rgb vektor és az xyz szorzótényezős mátrix
134         // mátrix szorzatával. Ha ebből ki akarjuk fejteni az rgb vektort, akkor az xyz mátrix
135         // inverzével be kell szorozni az RGB vektort
136         // Mátrixok létrehozása
137         mtx_type xyz[3][3] = {
138             { xr, xg, xb },
139             { yr, yg, yb },
140             { zr, zg, zb }
141         };
142         mtx_type RGB[3] = { red_raw, green_raw, blue_raw };
143         mtx_type rgb[3];
144         // Mátrix műveletek
145         // inverz mátrix helye
146         mtx_type xyz_inv[3][3];
147         // inverzálandó mátrix értékeinek behelyezése az inv mátrix helyére
148         Matrix.Copy((mtx_type*)xyz, 3, 3, (mtx_type*)xyz_inv);
149         // inv mátrix inverzálása
150         Matrix.Invert((mtx_type*)xyz_inv, 3);
151         // inv és RGB mátrix szorzása
152         Matrix.Multiply((mtx_type*)xyz_inv, (mtx_type*)RGB, 3, 3, 1, (mtx_type*)rgb);
153     } if (print == 2) {
154         //Számolt mátrixok kiírása
155         Serial.println();
156         Matrix.Print((mtx_type*)xyz, 3, 3, "xyz");
157         Serial.println();
158         Matrix.Print((mtx_type*)xyz_inv, 3, 3, "xyz_inv");
159         Serial.println();
160         Matrix.Print((mtx_type*)RGB, 3, 1, "RGB");
161         Serial.println();
162         Matrix.Print((mtx_type*)rgb, 3, 1, "rgb");
163         Serial.println();
164         Serial.print(red_raw);
165         Serial.print(" ");
166         Serial.print(green_raw);
167         Serial.print(" ");
168         Serial.print(blue_raw);
169         Serial.print(" ");
170         Serial.println(clear_raw);
171     }
172
173
174     float rm = rgb[0];
175     float gm = rgb[1];
176     float bm = rgb[2];

```

```

177
178     *R = constrain((rm * 255 / clear_raw), 0, 255);
179     *G = constrain((gm * 255 / clear_raw), 0, 255);
180     *B = constrain((bm * 255 / clear_raw), 0, 255);
181
182 } else {
183     *R = constrain(((float)red_raw * 255 / (float)clear_raw), 0, 255);
184     *G = constrain(((float)green_raw * 255 / (float)clear_raw), 0, 255);
185     *B = constrain(((float)blue_raw * 255 / (float)clear_raw), 0, 255);
186 }
187
188
189 // értékek kiírása debug céljából
190 if (print > 0) {
191     Serial.print("RED: ");
192     Serial.print(Red);
193     Serial.print(" GREEN: ");
194     Serial.print(Green);
195     Serial.print(" BLUE: ");
196     Serial.println(Blue);
197 }
198 }

```

A

pinekre kiíró függvény (199-219)

```

199 void pinColor2bit() {
200     // 2 biten kerül át 3db szín kiválasztásának eredménye.
201     // A pinek egy tranzisztort nyitnak negált módban.
202     if (FindColor("Zöld")) {
203         b0 = 0;
204         b1 = 1;
205         Serial.print("Zöld szín találva");
206     } else if (FindColor("Piros1") || FindColor("Piros2")) {
207         b0 = 1;
208         b1 = 0;
209         Serial.print("Piros szín találva");
210     } else if (FindColor("Kék")) {
211         b0 = 1;
212         b1 = 1;
213         Serial.print("Kék szín találva");
214     } else {
215         b0 = 0;
216         b1 = 0;
217         Serial.println("Nem találtam színt");
218     }
219 }

```

Setup függvénye az Arduinonak (221-238)

```

221 void setup() {
222   // beállítások
223   Serial.begin(9600);
224   pinMode(BIT_0, OUTPUT);
225   pinMode(BIT_1, OUTPUT);
226   // Szenzor inicializálás
227   if (tcs.begin()) {
228     Serial.println("TCS34725 csatlakoztatva és sikeresen inicializálva");
229   } else {
230     Serial.println("TCS34725-t nem sikerült megtalálni. Vezetékek ellenőrzése szükséges");
231     while (1)
232       ; // Program leállítása, ha nem talált szenzort
233   }
234   // Színtartományok megadása
235   ColorRangeMaker(15.0, 15.0);
236   prtPcolor();
237 }
238

```

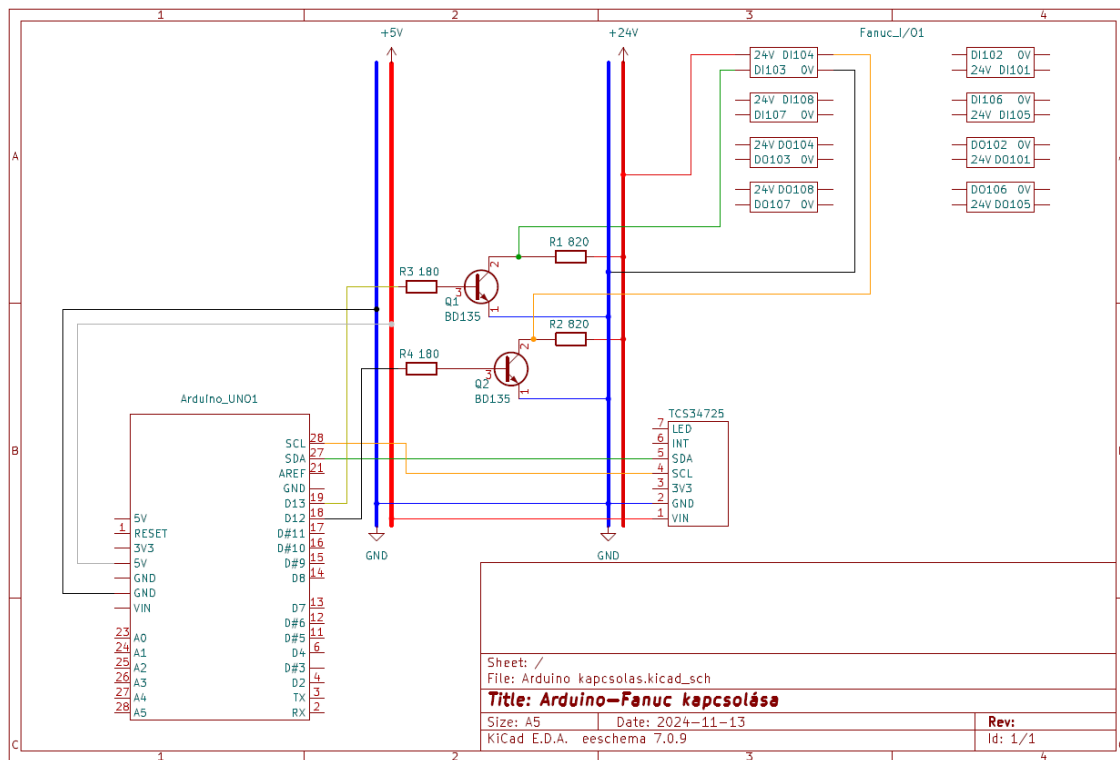
Loop függvénye az Arduinonak (239-250)

```

239 void loop() {
240   if ((millis() - CurrentMillis) > 100) {
241     CurrentMillis = millis();
242     // Pinekre kiírás. Tranzisztoros kapcsolás miatt negálva vannak az értékek
243     digitalWrite(BIT_0, b0);
244     digitalWrite(BIT_1, b1);
245     // Színértékek kinyerése
246     getRGB255(&Red, &Green, &Blue, 0, 1);
247     // Pinre való kiírás
248     pinColor2bit();
249   }
250 }

```

3. melléklet: Arduino-Fanuc összekapcsolása



4. melléklet: Fanuc robotkar program

Fő program

```

1      !A kirakott kockakat szín szerint
2      !szortirozza
3      !keszult:2024.11.11.
4      !keszitette: K.Tibor
5      !verzio:1.0
6
7      !Alaphelyzetre allas
8      Override=50%
9
10     CALL PROMPTYN(3,15)
11     IF R[15:V_yes/no]=1,JMP LBL[2]
12     CALL PROMPTOK(3)
13     PAUSE
14
15     LBL[2]
16     CALL AA_HOME
17
18     !parameter beallitasa
19     UFRAME_NUM=9
20     UTOOL_NUM=1
21     OVERRIDE=100%
22
23     PR[80:beirt poz]=LPOS-LPOS
24     PR[81:talalt poz]=PR[80:beirt poz]
25     PR[82:offset magassag]=PR[80:beirt poz]
26     PR[80,4:beirt poz]=180
27     PR[82,3:offset magassag]=100
28
29     !IRVISION futtatasa
30     VISION RUN_FIND 'SZD_KOCKA'
31     VISION GET_NFOUND 'SZD_KOCKA' R[16]
32
33     !FOR ciklus
34     FOR R[20:V_cikluszam]=1 TO R[16:V_found_item]
35     VISION GET_OFFSET 'SZD_KOCKA' VR[1] JMP LBL[1]
36     !talalt poz beirasa
37     PR[81]=VR[1].FOUND_POS[1]
38     PR[80,1:beirt poz]=PR[81,1:talalt poz]
39     PR[80,2:beirt poz]=PR[81,2:talalt poz]
40     PR[80,3:beirt poz]=0
41     PR[80,6:beirt poz]=PR[81,6:talalt poz]
42     PR[83]=VR[1].OFFSET
43     !ra allas es megfogas
44     J PR[80:beirt poz] 25% CNT100 Offset,PR[82:offset magassag] ACC50
45     CALL HAND_OPEN
46     L PR[80:beirt poz] 100mm/sec FINE ACC50
47     WAIT .10(sec)
48     CALL HAND_CLOSE
49     L PR[80:beirt poz] 250mm/sec CNT25 Offset,PR[82:offset magassag] ACC50
50     !szinerzekeles
51     J P[1:szinerzekelo_poz] 25% CNT25 Offset,PR[82:offset magassag] ACC50
52     L P[1:szinerzekelo_poz] 100mm/sec FINE ACC50
53     WAIT .40(sec)
54     !tarolo_1
55     IF (DI[104:ON ] AND !DI[103:ON ]) THEN
56     L P[1:szinerzekelo_poz] 250mm/sec CNT75 ofs ACC50
57     J P[2:piros_tarolo] 25% FINE ACC50
58     CALL HAND_OPEN
59     JMP LBL[3]
60     ENDIF
61
62     !tarolo_2
63     IF (!DI[104:ON ] AND DI[103:ON ]) THEN
64     L P[1:szinerzekelo_poz] 250mm/sec CNT75 ofs ACC50
65     J P[3:zold_tarolo] 25% FINE ACC50
66     CALL HAND_OPEN
67     JMP LBL[3]
68     ENDIF
69
70     !tarolo_3
71     IF (!DI[104:ON ] AND !DI[103:ON ]) THEN
72     L P[1:szinerzekelo_poz] 250mm/sec CNT75 ofs ACC50
73     J P[4:fekete_tarolo] 25% FINE ACC50
74     CALL HAND_OPEN
75     JMP LBL[3]
76     ENDIF
77

```

```
78      !nem ertelmezett
79      IF (DI[104:ON ] AND DI[103:ON ]) THEN
80 L      P[1:szinerzekelo_poz] 250mm/sec CNT75 ofs ACC50
81 J      P[5:nem_ertelmezett] 25% FINE ACC50
82      CALL HAND_OPEN
83      JMP LBL[3]
84      ENDIF
85
86      LBL[3]
87
88      ENDFOR
89
90      CALL AA_HOME
91
92
93      LBL[1]
94
```