Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Méréstechnika és Információs Rendszerek Tanszék

Videórögzítő robotkar vezérlésének és képfeldolgozásának megvalósítása

Szakdolgozat

Készítette
Törjék Noel

Konzulens
dr. Orosz György

2019. december 12.

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott *Törjék Noel*, 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, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), 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 eltelte után válik hozzáférhetővé.

Budapest, 2019. december 12.

Törjék Noel
hallgató

Kivonat

Film- és reklámforgatásoknál előfordul, hogy nem elegendő a kamerát állandóan egy helyben tartani. Ilyenkor használhatnak a kamera mozgatására síneket vagy kézzel is mozgathatják a kamerát, azonban manapság egyre gyakoribb kellék egy úgynevezett „cinebot". Ez egy robotkar a végére szerelve egy kamerával. Egy ilyen rendszer nem csak gyorsabban és pontosabban képes mozgatni a kamerát, de akár többször is reprodukálható vele ugyanaz a mozdulatsor.

Egy ilyen videórögzítő robotkar kisebb változatának elkészítése volt a dolgozat célja. A rendszer feladatai, hogy a kamera mozgatásához tudjon útvonalat tervezni, majd a tervezett útvonalon a mozgatást végrehajtani. Egy másik funkciója, hogy képes egy előre megtanított képet felismerni és a kamera orientációját úgy változtatni, hogy a kép ne térjen ki a kamera látószögéből.

Először bemutatom, hogyan lehet a robotkart, mozgást és koordináta-rendszereket mátrixok segítségével reprezentálni. Szó lesz az útvonaltervezéshez, inverz kinematikai feladat numerikus megoldásához és SURF módszer szerinti képfelismeréshez használt algoritmusokról, amiket MATLAB segítségével valósítottam meg. Leírom továbbá a szükséges hardverelemek kiválasztásának menetét, a hajtómotorok vezérlését végző elektronika és az erre írt szoftver működését.

Abstract

At film or commercial shoots sometimes it is not enough to keep the camera in one place. In this case cameras are usually moved on rails or by hand, however nowadays it is more and more common to use a so called 'cinebot'. This is a robotic arm with a camera attached to its end. With this system you can not only move the camera faster and more precisely, but also replicate the same movement multiple times.

The goal of this thesis is to make a smaller version of this video recorder robotic arm. The task of this system is design a path for the movement of the camera and move the camera on that path afterwards. Another feature is to be able to recognize an image and change the orientation of the camera so that the image does not move out of frame.

Firstly, I will explain how we can represent the robotic arm, movement and coordinate systems with matrices. Topics include the algorithms of path planning, solving inverse kinematics with a numerical method and image recognition using the SURF method which were used in MATLAB. I also write about selecting the hardware components, the control electronics of the servomotors and the software I wrote.

1. fejezet

Bevezetés

1.1. Motiváció

Film- és reklámforgatásoknál előfordul, hogy nem elegendő a kamerát állandóan egy pozícióban tartani, mert a felvétel tárgya mozgásban van vagy túlságosan egyhangú lenne a felvétel. Ilyenkor gyakran vagy egy operatőr mozgatja a kamerát vagy sínrendszert használnak. Ezen módszerek legtöbbször megfelelőek, de vannak hátrányaik. Emberi pontatlanságból adódóan az operatőr nem képes milliméteres pontossággal mozdulatsorokat végrehajtani vagy pontosan ugyanazt a mozdulatsort többször reprodukálni. Sínes mozgatás esetén ezek kevésbé vannak jelen, viszont a kamera csak egy fix pályán tud mozogni. Ha az előbb említett módszerek nem jöhetnek szóba, akkor általában egy úgynevezett „cinebot" alkalmazása a megoldás, ami egy robotkar a végén egy kamerával, ami a 1.1 ábrán is látható. Egy ilyen robotkar nem csak pontosabban, de gyorsabban is végre tudja hajtani a kívánt mozdulatsort, miközben a kamera beállításait (fókusz, írisz, zoom) is képes állítani. Az ilyen célokra használt robotok általában módosított ipari robotkarok és áruk több százezer dolláros nagyságrendben mozog, ezért kisebb költségvetésű projektekhez ritkán alkalmazzák.

Egy hasonló funkciójú, de kisebb robotkar megvalósítása volt a cél, ami egy asztalon is elfér. Léptetőmotorok helyett egyenáramú szervomotorokat alkalmazva, valamint alumínium lemezből kivágott vázat használva csökkenthetők a költségek. Egy ilyen robotkar hasznos lehet például kis költségvetésű filmekhez vagy reklámfilmekhez. A rendszer egyik felhasználási módja, hogy megadott szempontok alapján, megadott pozíciók között egy útvonalat képes megtervezni és ezen végigmenni. Mialatt a robot végrehajtja a mozdulatsort, a végére szerelt kamera felvételt rögzít. Egy másik használási módja, hogy egy előre megtanított objektumot képes felismerni és látótérben tartani. A robotkar végberendezésének pozícióját megadjuk, viszont az orientációját változtathatja úgy, hogy a mozgó objektum ne térjen ki a képből.

1.1. ábra. A Motorized Precision által forgalmazott KIRA videórögzítő robotkar [6]

1.2. A szakdolgozat felépítése

A 2. fejezet egy átfogó leírást ad a rendszerről.

A 3. fejezet a robotkarral kapcsolatos elméleti ismereteket tartalmazza.

A 4. fejezet a robotkarról szól, különbontva hardver és szoftver szerint. A hardverrel foglalkozó rész a hardver kiválasztásával foglalkozik és kitér a vázra, a hajtásra és a mikrovezérlőt tartalmazó áramkörre. A szoftverrel foglalkozó rész magában foglalja az útvonaltervezés módját és a grafikus felhasználói felület elkészítését is.

A 5. fejezet a képfelismerésről és -követésről szól.

A 6. fejezet pedig a végeredmény értékelését és a továbbfejlesztési lehetőségeket tartalmazza.

2. fejezet

Rendszerterv

A rendszer - a 2.1. ábrán látható módon - a következő fő komponensekből áll:

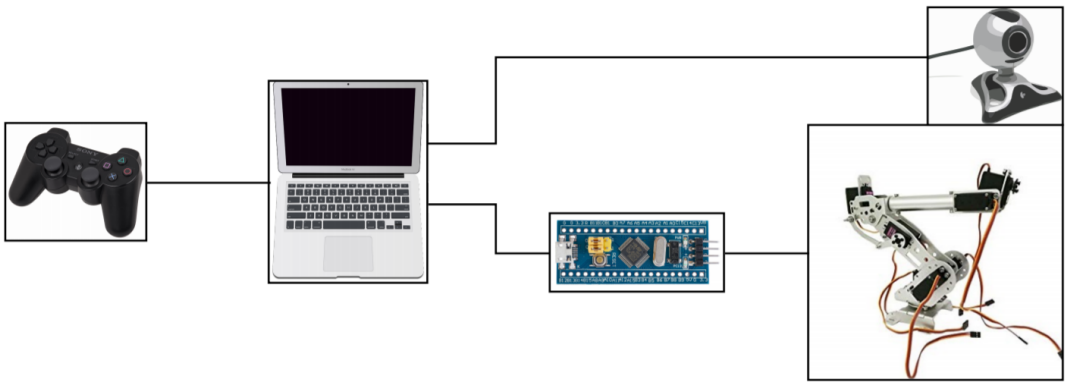
- kontroller
- számítógép
- mikrovezérlő
- robotkar
- kamera

A kontroller egy közönséges, általában számítógépes játékokhoz használt eszköz, amin gombok és két darab két tengelyes kar található.

A számítógép lehet egy PC vagy laptop, amin a MATLAB képes megfelelő gyorsasággal futni. Minden számítás és feldolgozás ezen a számítógépen történik.

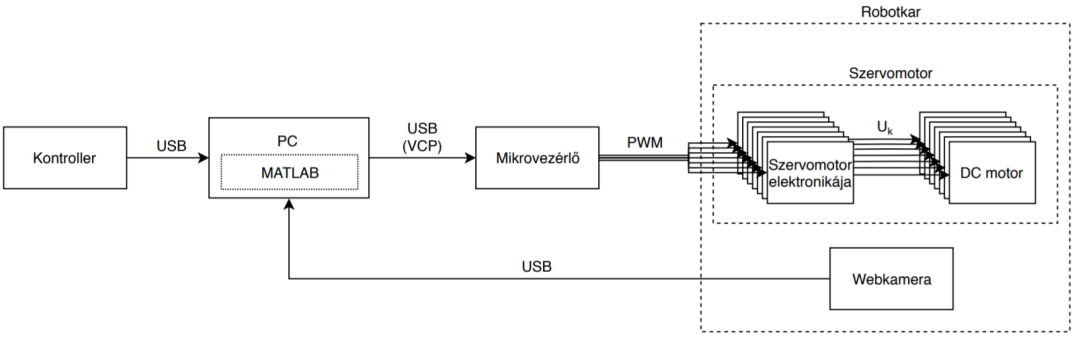
A robotkar magába foglalja a vázat és a hajtást, valamint a kamera a végére van rögzítve. A kamera egy USB webkamera. Az hajtást jelen esetben szervomotor biztosítja. A szervomotor belsejében található egy egyenáramú motor fogaskerékrendszerrel, valamint bele van építve egy elektronika, ami impulzusszélesség-modulációval (PWM) irányítható. Egy adott PWM jelhez tartozik egy szög, amire a vezérlő elektronika beállítja a tengelyt a belül visszacsatolt pozíció alapján. A szervok vagy 0-180° vagy 0-270° tartományban képesek változtatni a tengelyük szögét.

A mikrovezérlő feladata, hogy kapcsolatot teremtsen a számítógép és a szervomotorok beépített elektronikája között.



2.1. ábra. Egyszerűsített rendszerterv

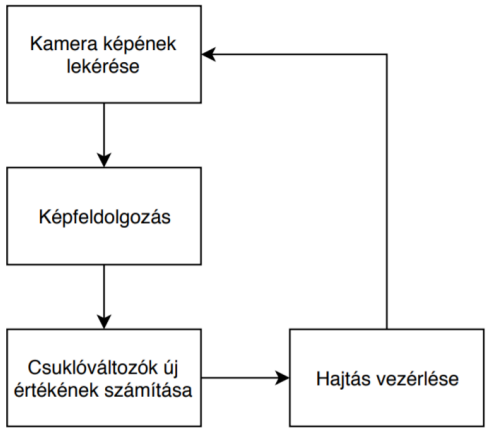
Ahogy a 2.2. ábra is mutatja, a számítógéphez csatlakozik a kontroller, webkamera és mikrovezérlő USB-n keresztül. A mikrovezérlőhöz kapcsolódnak a robotkart mozgató egyenáramú szervomotorok. A kontroller egy HID (Human Interface Device) osztályú USB eszköz, amelytől a MATLAB megkapja a gombok állását. Ha a MATLAB mozgatni kívánja a robotkart, akkor számításokat végez és a kiszámolt értékeket (szögeket) elküldi a mikrovezérlőnek. A mikrovezérlő amint megkapta az értékeket, előállítja a megfelelő szélességű impulzust és a szervok elektronikáját impulzusszélesség-modulációval vezérli. A szervok beépített elektronikája a PWM jelek alapján a megadott pozícióba vezérlik a szervokat. A webkamera szintén USB-n kereszült csatlakozik a számítógéphez. Paramétereit (pl. felbontás, fókusztávolság) képes a MATLAB beállítani, valamint az aktuális képet lekérni, amiket elment és/vagy feldolgoz.



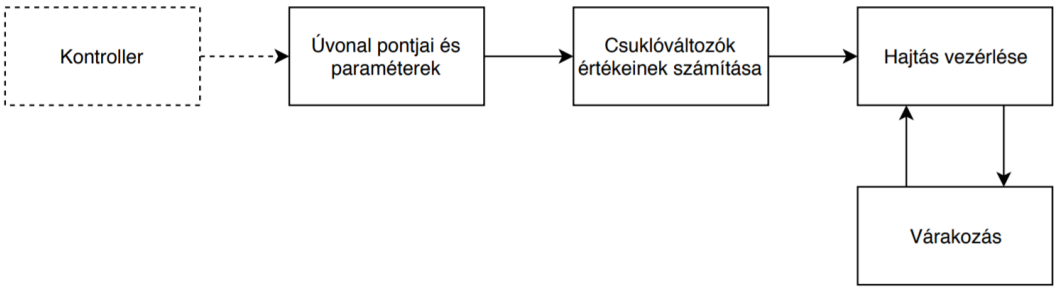
2.2. ábra. Részletes rendszerterv

Ideális esetben a számítások ideje és kommunikációból adódó késleltetés kisebb, mint a szervokat vezérlő PWM jel periódusideje, ilyen esetben beszélhetünk valós idejű rendszerről. A robotkar egyik felhasználási módjának vázlata a 2.3. ábrán látható. Ebben az

esetben a robotkar végén található kamera pozícióját a kontroller segítségével előre beállíthatjuk, az orientációját pedig maga állítja oly módon, hogy a felismert tárgy a kamera látószögében maradjon.



2.3. ábra. 1. felhasználási mód



2.4. ábra. 2. felhasználási mód

A másik felhasználási módjának vázlata a 2.4. ábrán látható, amikor előre megadunk pontokat és orientációkat a térben és a programnak van ideje, hogy bizonyos paramétereknek megfelelően megtervezze az útvonalat. A kontrollert is használhatjuk a pontok megadására, de számokkal is megadhatjuk a kezelőfelületen keresztül. Az útvonalat elég diszkrét pontokra kiszámolni. Ha a számításokat elvégezte a gép, akkor a szervo számára az új szögeket PWM periódusonként egyszer küldjük el.

3. fejezet

Robottechnika

3.1. Bevezető

Ez a fejezet tartalmazza a robotokkal és robotirányítással kapcsolatos fő fogalmakat és problémákat. A magyar nyelvű irodalomban különböző elnevezései vannak az egyes fogalmaknak, ezért az új fogalmak bevezetésénél zárójelben szerepelnek a további nevei is. A feladatkiírásban szereplő robotkar kifejezés egy emberi karhoz hasonló felépítésű robotra utal, azonban van olyan irodalom, ami a robot szegmenseit hívja robotkarnak, ezért csak robotként fogok hivatkozni rá. A fejezet alapvetően a [4] [5] [9] [12] források felhasználásával készült.

3.2. Alapfogalmak

3.2.1. Robotcsuklók

A robotcsuklók [12] alapvetően 1-szabadságfogúak és két fajta mozgásuk lehetséges: haladó- vagy forgómozgás. Ezek az úgynevezett:

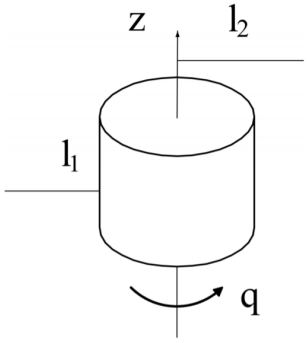
- transzlációs csuklók
- rotációs csuklók

A rotációs csuklót R szimbólummal jelöljük és vázlata a 3.1. ábrán látható. A rotációs csukló lehetővé teszi a az ábrán q-val jelölt tengely körüli elfordulást. A transzlációs csuklót T szimbólummal jelöljük és a 3.2 ábrán látható a vázlata. Lehetővé teszi az ábrán q-val jelölt tengely menti elmozdulást.

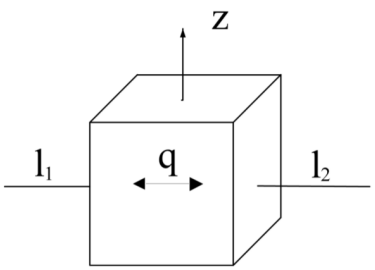
3.2.2. Robotszegmensek

A robotszegmensek (karok/tagok) merev testek, melyeket a csuklók kapcsolnak össze. Ezek kinematikai paraméterei:

- a szegmens hossza
- a robotcsukló-tengelyek egymással bezárt szöge



3.1. ábra. Rotációs robotcsukló vázlata [12]



3.2. ábra. Transzlációs robotcsukló vázlata [12]

3.2.3. Kinematikai lánc

A kinematikai lánc egymást követő szegmensekből és az ezeket összekötő csuklókból (kényszerekből) áll. Két szegmens és az ezeket összekötő csukló esetén kinematikai párról beszélünk. A robotstruktúra szerint feloszthatő:

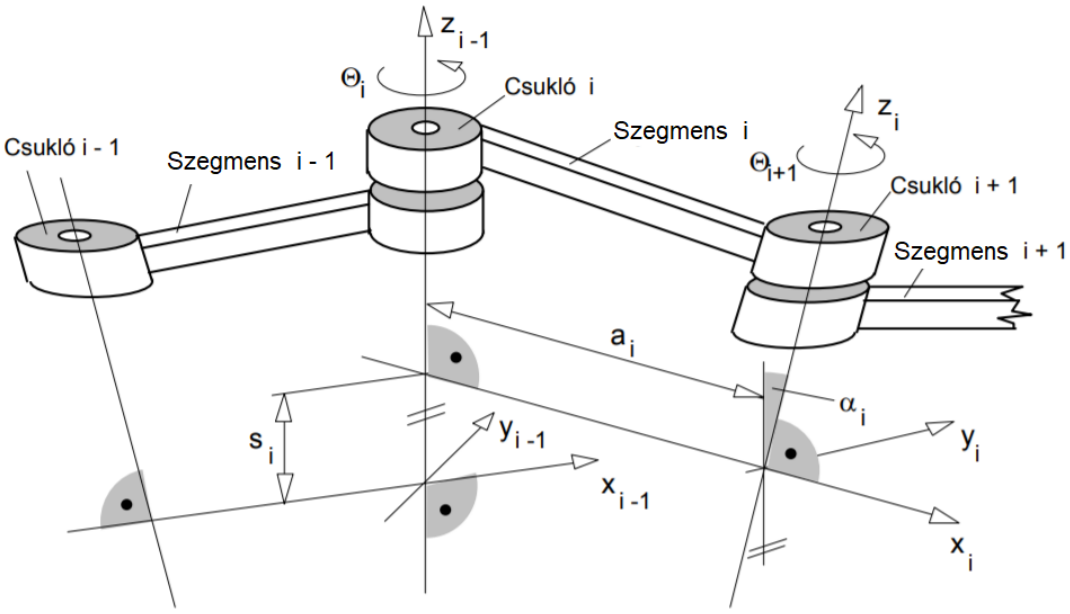
- egyszerű
- összetett

illetve kényszerek szempontjából:

- nyitott
- zárt

A kinematikai lánc akkor egyszerű, ha egyik szegmens sem kapcsolódik több, mint két kinematikai párhoz. Amennyiben van legalább egy olyan szegmense, ami több mint két kinematikai bárhoz tartozik, akkor összetett kinematikai láncról beszélünk.

Nyitott kinematikai láncnál nem minden szegmens tartozik két kinematikai párhoz, zárt esetén viszont mindegyik.



3.3. ábra. Kinematikai lánc [5]

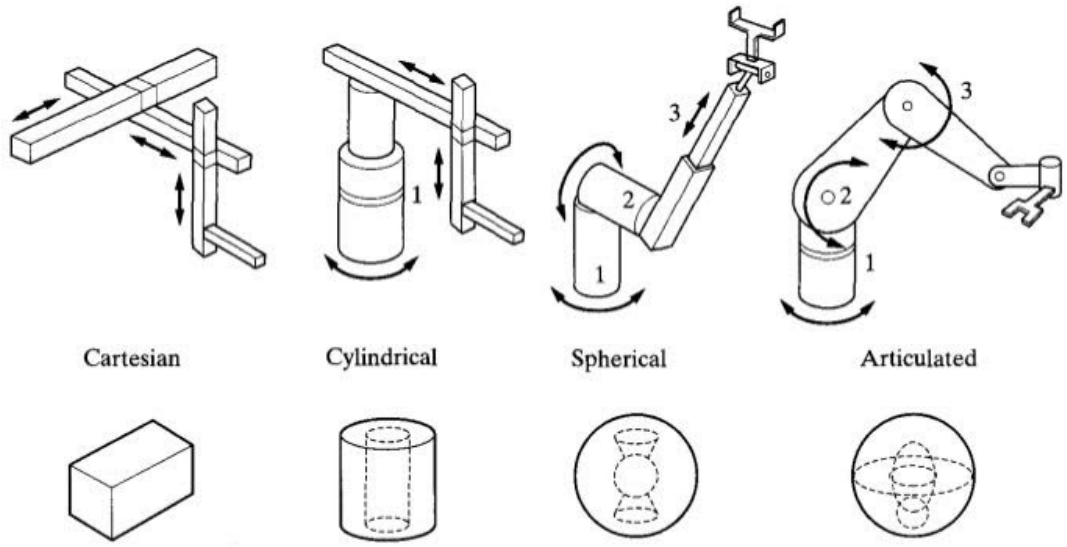
3.3. Robotok csoportosítása

Az alapkonfiguráció alatt a rögzített alapzattól kiindulva három csuklós, 3-szabadságfogú kinematikai láncot értünk. [4] Az alapkonfiguráció feladata, a TCP (Tool Center Point / szerszámközéppont) pozicionálása a munkatérben. Mivel a csuklók rotációsak vagy transzlációsak lehetnek, kinematikailag $2^3 = 8$ összekapcsolás lehetséges:

- R R R
- R R T
- R T R
- R T T
- T R R
- T R T
- T T R
- T T T

Ezek megszabják, hogy milyen munkatérrel rendelkezik a robot, ahogy a 3.4. ábrán látható. Az alábbiak terjedtek el az iparban:

- derékszögű koordináta-rendszerű T T T,



3.4. ábra. A derékszögű koordináta-rendszerű, henger koordináta-rendszerű, gömbi koordináta-rendszerű és függőleges síkú csuklókaros robotot és munkaterük [18]

- henger koordináta-rendszerű R T T
- gömbi koordináta-rendszerű R R T
- csuklós rendszerű R R R

Ha azonos paramétereket feltételezünk, azaz az elmozdulások maximális hossza és szegmensek hossza állandó, valamint a maximális rotáció ±180°, akkor az RRR típusú robotok munkatere a legnagyobb. Hátránya viszont, hogy rotációs csuklóknál a pozicionálási hibák összeadódnak, ezért a pozicionálási hiba is itt lehet a legnagyobb.

3.4. Direkt kinematikai feladat

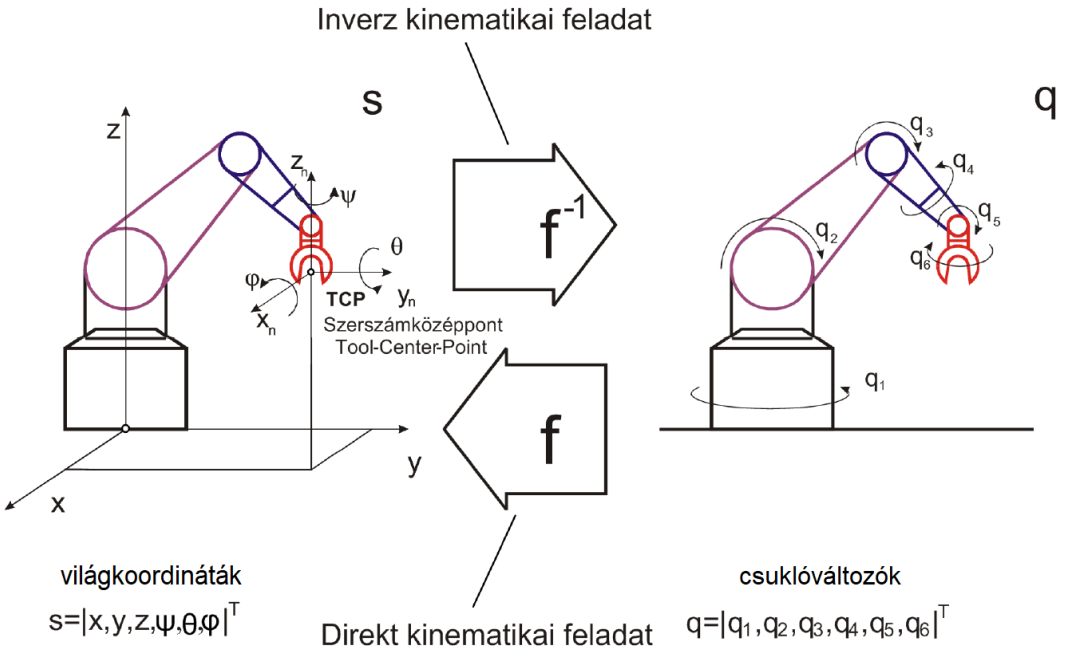
Direkt kinematikai feladatról akkor beszélünk, ha a csuklóváltozók értékei ismertek és ezek alapján szeretnénk megtudni a TCP helyzetét világkoordinátákkal megadva. [12]

$$f(q) = s \tag{3.1}$$

ahol a 3.5. ábrának megfelelően:

- q: a csuklóváltozók vektora
- s: a világkoordináták vektora

A direkt kinematikai feladatnak mindig egy megoldása van.



3.5. ábra. Inverz és direkt kinematikai feladat [12]

3.4.1. Rotációs mátrix

Két koordináta-rendszer orientációjának viszonya megadható egy rotációs mátrix segítségével [5]:

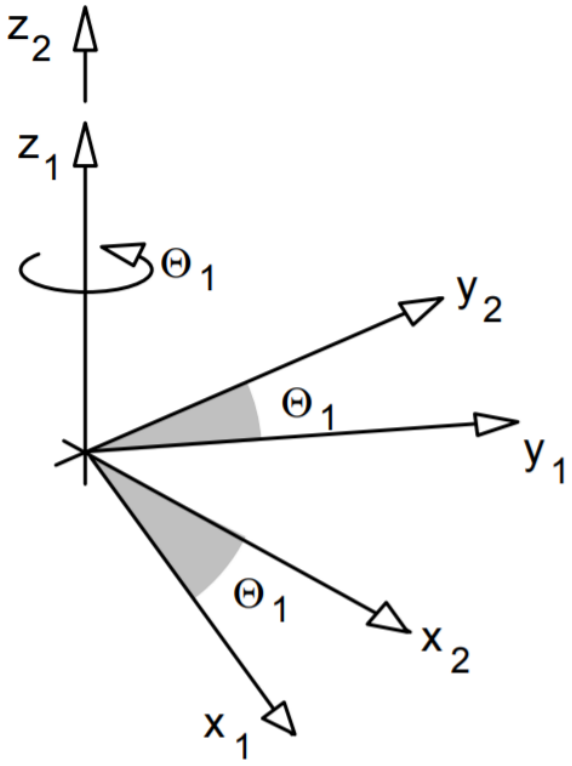
$$R = \begin{bmatrix} e_{1x} & e_{2x} & e_{3x} \\ e_{1y} & e_{2y} & e_{3y} \\ e_{1z} & e_{2z} & e_{3z} \end{bmatrix} \tag{3.2}$$

Ez nem más, mint három egységvektor (e_1, e_2, e_3), amely az új koordináta-rendszer x, y és z tengelyének irányába mutat.

A z tengely körüli Θ_1 szöggel való forgatás, amelyet a 3.6. ábra mutat a következőképpen írható le:

$$R_z = Rot(z) \begin{bmatrix} \cos\Theta_1 & -\sin\Theta_1 & 0 \\ \sin\Theta_1 & \cos\Theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

Hasonló módon felírható rotációs mátrixok az x és y tengelyek körül forgatásra α_1 és β_1 szöggel:



3.6. ábra. z tengely körüli forgatás [5]

$$R_x = Rot(x) \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha_1 & -\sin\alpha_1 \\ 0 & \sin\alpha_1 & \cos\alpha_1 \end{bmatrix} \tag{3.4}$$

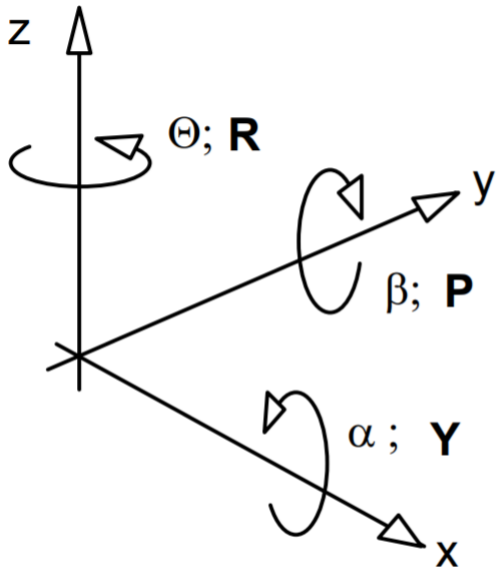
$$R_y = Rot(y) \begin{bmatrix} \cos\beta_1 & 0 & \sin\beta_1 & 0 \\ 0 & 1 & 0 & \\ -\sin\beta_1 & 0 & \cos\beta_1 & 0 \end{bmatrix} \tag{3.5}$$

A három mátrixot összeszorozhatjuk és megkapjuk a három tengely körüli egyidejű forgatás mátrixát:

$$
\begin{aligned}
&R_z \cdot R_x \cdot R_y = Rot(z) \cdot Rot(x) \cdot Rot(y) = \\
&= \begin{bmatrix} \cos\Theta_1 & -\sin\Theta_1 & 0 \\ \sin\Theta_1 & \cos\Theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha_1 & -\sin\alpha_1 \\ 0 & \sin\alpha_1 & \cos\alpha_1 \end{bmatrix} \begin{bmatrix} \cos\beta_1 & 0 & \sin\beta_1 & 0 \\ 0 & 1 & 0 & \\ -\sin\beta_1 & 0 & \cos\beta_1 & 0 \end{bmatrix} = \\
&= \begin{bmatrix} \cos\Theta_1\cos\beta_1 - \sin\Theta_1\sin\alpha_1\sin\beta_1 & -\sin\Theta_1\cos\alpha_1 & \cos\Theta_1\sin\beta_1 + \sin\Theta_1\sin\alpha_1\cos\beta_1 \\ \sin\Theta_1\cos\beta_1 - \cos\Theta_1\cos\alpha_1\sin\beta_1 & \cos\Theta_1\cos\alpha_1 & \sin\Theta_1\sin\beta_1 - \cos\Theta_1\sin\alpha_1\cos\beta_1 \\ -\cos\alpha_1\sin\beta_1 & \sin\alpha_1 & \cos\alpha_1\cos\beta_1 \end{bmatrix}
\end{aligned}
\tag{3.6}
$$

Az orientáció megadása három független szög megadásával többféleképpen is lehetséges attól függően, hogy melyik tengelyek körül és milyen sorrendben forgatunk.

Gyakran használt mód az orientáció megadására az RPY szögek alkalmazása, amely a csavarás (roll), billentés (pitch) és forgatás (yaw) szögének megadását jelenti, amelyet a 3.7. ábra mutat.

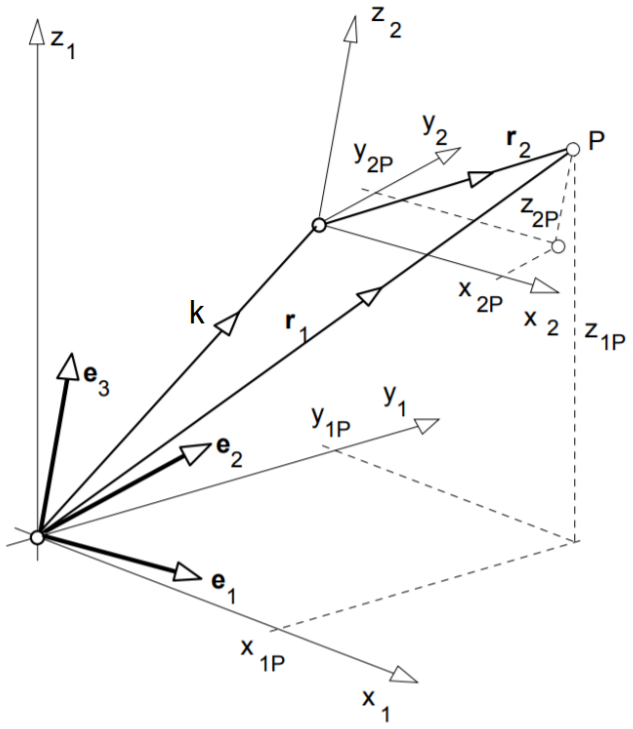


3.7. ábra. RPY szögek [5]

3.4.2. Eltolási vektor

Ha két koordináta-rendszer távolságát akarjuk leírni, akkor ezt egy k vektor segítségével megtehetjük. k nem más, mint egy koordináta-rendszer helyvektora a referencia koordináta-rendszerben, tehát k_x az x tengely mentén történő eltolást, k_y az y tengely mentén történő eltolást, k_z pedig a z tengely mentén történő eltolást írja le.

$$k = \begin{bmatrix} k_x \\ k_y \\ k_z \end{bmatrix} \tag{3.7}$$



3.8. ábra. Nyugvó és mozgó koordináta-rendszer és bennük elhelyezkedő P pont [5]

3.4.3. Homogén koordináta-transzformáció

Homogén koordináta-transzformációt a homogén transzformációs mátrix segítségével tudunk végez. Ez egy olyan 4x4-es mátrix, amely segítségével egy derékszögű koordináta-rendszeren egyszerre tudunk leírni:

- forgatást
- eltolást

Ha az eltolási vektort a rotációs mátrix mellé helyezzük és kiegészítjük a 3.8. relációban látható módon, akkor megkapjuk a homogén transzformációs mátrixot. E mátrix kompakt módon reprezentálhatja két koordináta-rendszer egymáshoz képesti viszonyát vagy valamilyen elmozdulást. A következőképpen írható fel:

$$H = \begin{bmatrix} R & T \\ 0^T & 1 \end{bmatrix} \tag{3.8}$$

Ahol:

- R - 3x3-as rotációs mátrix
- T - 3x1-es transzlációs vektor
- 0^T - 1x3-as nullvektor

Nevezzük el a 3.8. ábrán látható 1-es koordináta-rendszert nyugvó koordináta-rendszernek és rögzítsük a robot alapjához. Legyen a 2-es koordináta-rendszer a mozgó koordináta-rendszer, amely a TCP-hez kötődik.

A homogén transzformációs mátrixot három dologra használható:

- megadja a mozgó koordináta-rendszer orientációját a nyugvó koordináta-rendszerhez képest
- megadja a mozgó koordináta-rendszer origójának helyét a nyugvó koordináta-rendszerhez képest
- ha ismerünk egy adott pontot a mozgó koordináta-rendszerben, akkor fel tudjuk vele írni ugyanezt a pontot a nyugvó koordináta-rendszerben.

3.4.4. Denavit-Hartenberg féle transzformációs mátrix

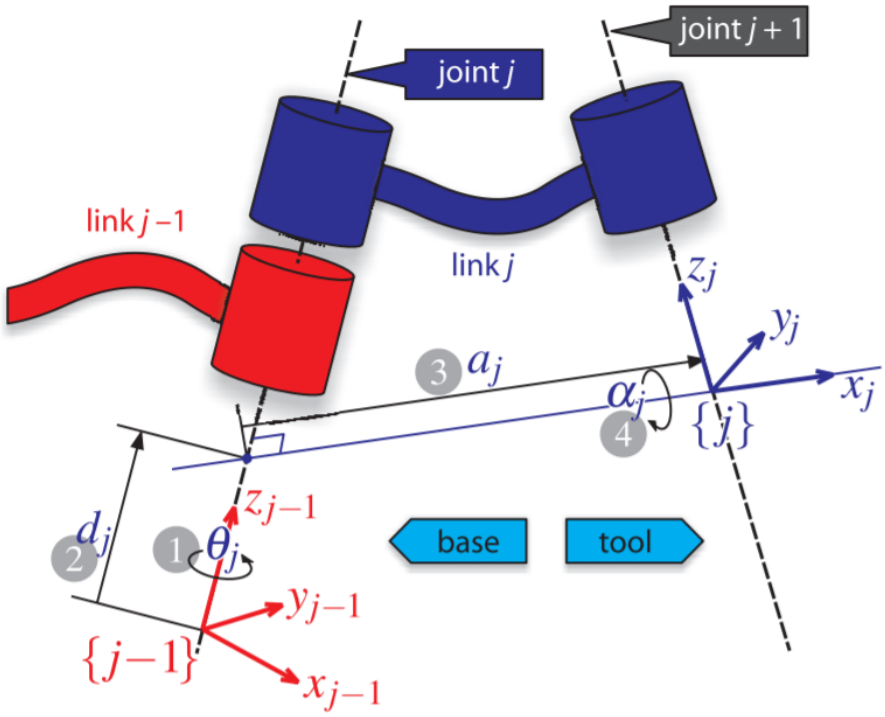
Ha csuklóváltozókból szeretnénk megkapni a világkoordinátákat, akkor azt a Denavit-Hartenberg féle transzformációs mátrix segítségével tehetjük meg. A Denavit-Hartenberg módszer [9] lényege, hogy két haladó (d, a) és két forgó (θ, α) mozgással átvihető egyik koordináta-rendszer a másikba.

A mátrix felírásához először a Denavit-Hartenberg paramétereket kell először meghatároznunk. A j-edik és j+1-edik csuklóhoz koordináta-rendszereket rögzítünk a 3.9. ábrán látható módon, a következő szabályokkal:

- x_j tengely metszi z_{j-1}-et
- x_j merőleges z_{j-1}-re

Az ilyen módon rögzített koordináta-rendszerekben a paraméterek:

- θ_j: z_{j-1} tengely körüli forgatás
- d_j: z_{j-1} tengely irányában történő eltolás
- a_j: x_j tengely irányában történő eltolás



3.9. ábra. Denavit-Hartenberg paraméterek helyes felvétele [9]

- α_j: x_j tengely körüli forgatás

Ha ismertek a paraméterek, felírhatjuk belőle a homogén transzformációs mátrixot. Két egymást követő rotációs csuklóra rögzített koordináta-rendszer esetén:

$$ {}^{j-1}D_j = \begin{bmatrix} \cos q_j & -\sin q_j \cos \alpha_j & \sin q_j \sin \alpha_j & a_j \cos q_j \\ \sin q_j & \cos q_j \cos \alpha_j & -\cos q_j \sin \alpha_j & a_j \sin q_j \\ 0 & \sin \alpha_j & \cos \alpha_j & d_j \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.9} $$

Két transzlációs csukló esetén:

$$ {}^{j-1}D_j = \begin{bmatrix} \cos \theta_j & -\sin \theta_j \cos \alpha_j & \sin \theta_j \sin \alpha_j & 0 \\ \sin \theta_j & \cos \theta_j \cos \alpha_j & -\cos \theta_j \sin \alpha_j & 0 \\ 0 & \sin \alpha_j & \cos \alpha_j & q_j \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.10} $$

α és a paraméterek értéke mindig állandó. Rotációs csukló esetén θ változó és d állandó, míg transzlációs csukló esetén θ állandó és d változó.

Ha minden szomszédos koordináta-rendszerre felírjuk a Denavit-Hartenberg féle transzformációs mátrixot, akkor ezen mátrixok szorzatából megkaphatjuk az alaphoz rög-

zített nyugvó koordináta-rendszer és a TCP koordináta-rendszere közötti homogén transzformációs mátrixot.

$$^{0}T_{n} = {}^{0}D_{1}\ {}^{1}D_{2}\ ...\ {}^{n-1}D_{n} \tag{3.11}$$

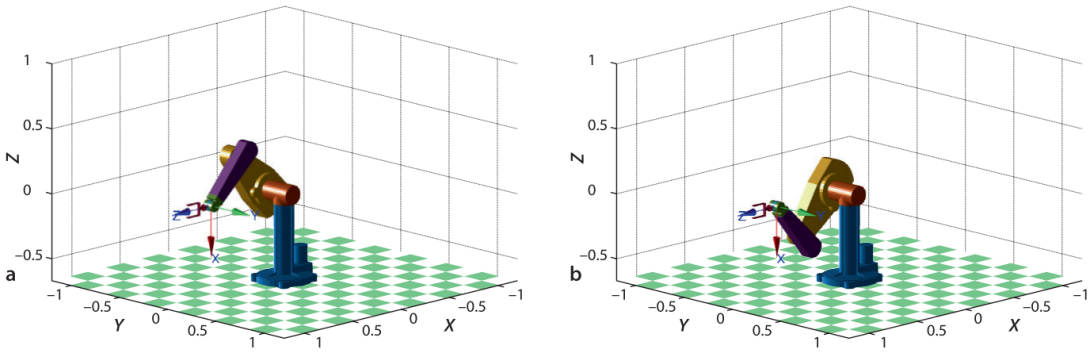
$^{0}T_{n}$ mátrix meghatározása a csuklóváltozók ismertében a direkt kinematikai feladat megoldásának az alapja.

3.5. Inverz kinematikai feladat

Ha világkoordinátákból szeretnék meghatározni a csuklóváltozókat, akkor az inverz kinematikai feladatról beszélünk. Az inverz és direkt kinematika viszonyát jól érzékelteti a 3.5. ábra. Neve is mutatja, hogy a korábban használt függvény inverzére van szükségünk:

$$q = f^{-1}(s) \tag{3.12}$$

Az inverz kinematikai feladatnak van nagyobb gyakorlati haszna: hogyan juttassuk el a TCP-t egy adott pontba. Általában a függvénynek nincs egyértelmű megoldása, több csuklóváltozó vektornak lehet ugyanaz a TCP pozíció az eredménye.



3.10. ábra. Ugyanaz a TCP pozíció elérhető különböző csuklóváltozó értékekkel [9]

Az inverz kinematikai feladat kétféleképpen oldható meg:

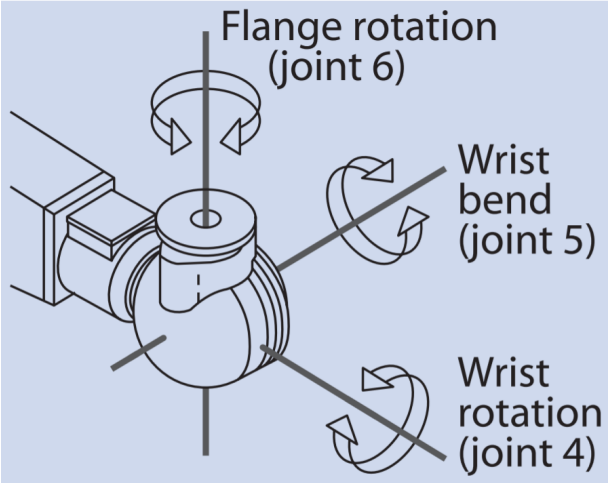
- analitikus módon
- numerikus módon

3.5.1. Analitikus megoldás

Az inverz nemlineáris, folytonosan deriválható f vektorfüggvény, amely leképezi a világkoordinátákat csuklóváltozókká, egy összetett n változós függvény, ezért az inverz

kinematika feladat megoldása összetett. Analitikus megoldása akkor létezhet egy inverz kinematikai feladatnak, ha az alábbi feltételek teljesülnek [9]:

- a robot 6 szabadsági fokkal rendelkezik
- a robot gömbcsuklóval állítja az orientációt



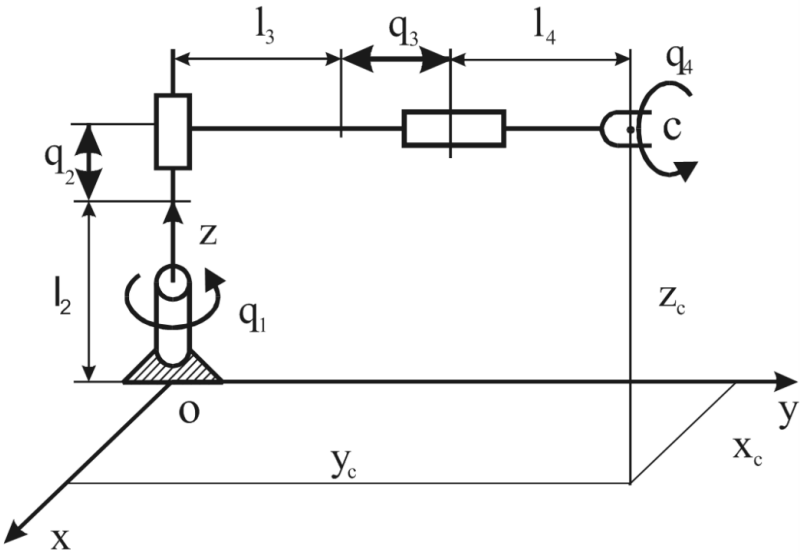
3.11. ábra. Gömbcsukló tengelyei [9]

A gömbcsukló általánosan egy olyan csuklót jelent, ami egynél több tengely körül képesek mozgást végzeni. Ebben az esetben a 3.11. ábrán látható gömbcsuklóról van szó, ami a TCP orientációját úgy képes állítani, hogy közben transzlációs mozgást nem végez. A legtöbb modern ipari robot rendelkezik ilyen gömbcsuklóval. Ha ezen feltételek valamelyike nem teljesül, akkor csak a numerikus módszer használható. Az analitikus megoldásnak viszont számos előnye van:

1. megkaphatjuk az összes megoldást
2. pontos eredményeket kaphatunk
3. kevesebb számítási időt igényel
4. ismerjük a szingularitásokat

Az analitikus módszerrel se kapunk megoldást, ha a robot szegmensei nem elég hosszúak vagy az adott pozíciót fizikailag nem képes megvalósítani. Lehet, hogy egy pozíció nem elérhető, mert több tengely egybe esik, ezzel csökkentve a szabadsági fokot, ezt nevezzük szingularitásnak.

A módszer jól szemléltethető a 3.12. ábrán látható négy szabadságfokú hengeres roboton. [12] Ez a robot ugyan nem rendelkezik gömbcsuklóval, de tekinthetünk rá úgy, mintha a gömbcsukló két tengelye rögzítve lenne. A lényeg, hogy nem végeznek transzlá-



3.12. ábra. Egy 4 szabadságfokú robot [12]

ciós mozgást. A világkoordináták és csuklóváltozók közötti összefüggés:

$$x_c = (l_3 + \ l_4 + \ q_4) \ cos \ q_1, \tag{3.13}$$

$$y_c = (l_3 + \ l_4 + \ q_3) \ sin \ q_1, \tag{3.14}$$

$$z_c = \ q_2 + \ l_2, \tag{3.15}$$

$$\beta = \ q_4, \tag{3.16}$$

ahol

- $s = [x_c, y_c, z_c, \beta]^T$: a világkoordináták vektora,
- $q = [q_1, q_2, q_3, q_4]^T$: a csuklóváltozók vektora,
- l_2, l_3, l_4: a szegmensek hossza.

Az inverz kinematikai feladat analitikus megoldása ebben az esetben:

$$q_1 = \arctan \frac{y_c}{x_c}, \tag{3.17}$$

$$q_2 = z_c - l_2, \tag{3.18}$$

$$q_3 = (x_c^2 + y_c^2)^{\frac{1}{2}}, \tag{3.19}$$

$$q_4 = \beta. \tag{3.20}$$

3.5.2. Numerikus megoldás

Az inverz kinematikai feladat megoldására tekinthetünk úgy is, hogy addig változtatjuk a csuklóváltozókat, ameddig a direkt kinematikájuk megadja a kívánt pozíciót.

Ez gyakorlatilag egy optimalizációs feladat, amelyben minimalizálni kell a hibát a direkt kinematikai eredmény és a kívánt pozíció között.

Ez a megoldási módszer több ideig tart, viszont előnye, hogy:

- nem csak 6 csuklóval rendelkező robotoknál működik
- szingularitásokat is képes kezelni

3.5.2.1. Jacobi-mátrix

A Jacobi-mátrix a világkoordináták sebességvektora és a csuklóváltozók sebességvektora közötti kapcsolatot írja le. Fogalmát a következőképpen vezethetjük be:

A robot direkt kinematikai egyenlete [12]:

$$s = f(q) \tag{3.21}$$

Deriválás után:

$$\dot{s} = \left(\frac{\partial f}{\partial q}\right) \dot{q} \tag{3.22}$$

Ahol a Jacobi-mátrix:

$$J(q) = \left(\frac{\partial f}{\partial q}\right) \tag{3.23}$$

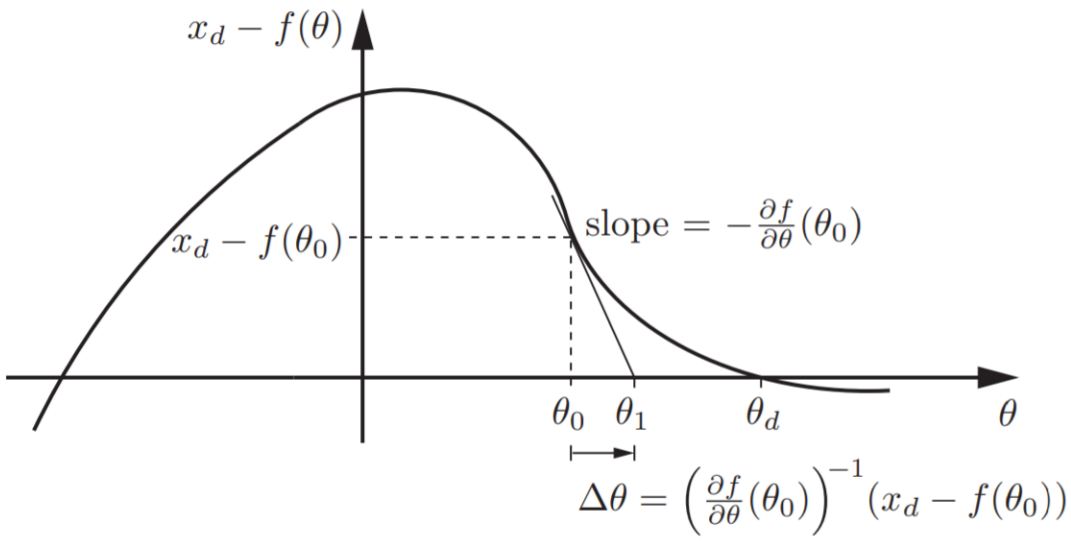
Így a 3.22. reláció felírható a következő módon:

$$\dot{s} = J(q)\dot{q} \tag{3.24}$$

A Jakobi-mátrix segítségével megoldható az inverz kinematikai feladat iteratív módon a Newton-Raphason módszerrel. [9] [24] x_d a kívánt pozíció, q_0 pedig a kezdeti becslés.

1. Meghatározzuk a Jacobi-mátrixot: $J(q_i)$.
2. Előállítjuk az inverzét: $J^{-}1(q_i)$.
3. Kiszámoljuk a hiba mértékét: $e = x_d - f(q_i)$.
4. A csuklóváltozók új értéke: $q_{i+1} = q_i + J^{-1}(q_i)e$.
5. Növeljük i értékét.

Az algoritmust addig futtatjuk, ameddig a egy adott számú iterációt végre nem hajtunk vagy a hiba egy adott ϵ érték alá nem kerül: $|e| < \epsilon$. Ha f lineáris, az algoritmus egy lépésben megoldja a feladatot.



3.13. ábra. Az inverz kinematikai feladat megoldása Newton-Raphason módszerrel. A csuklóváltozók θ-val vannak jelölve. [24]

3.6. Pályatervezés világkoordinátákban

A pályatervezést végezhetjük:

- off-line, azaz a robot tanulási fázisában
- on-line, azaz valós időben

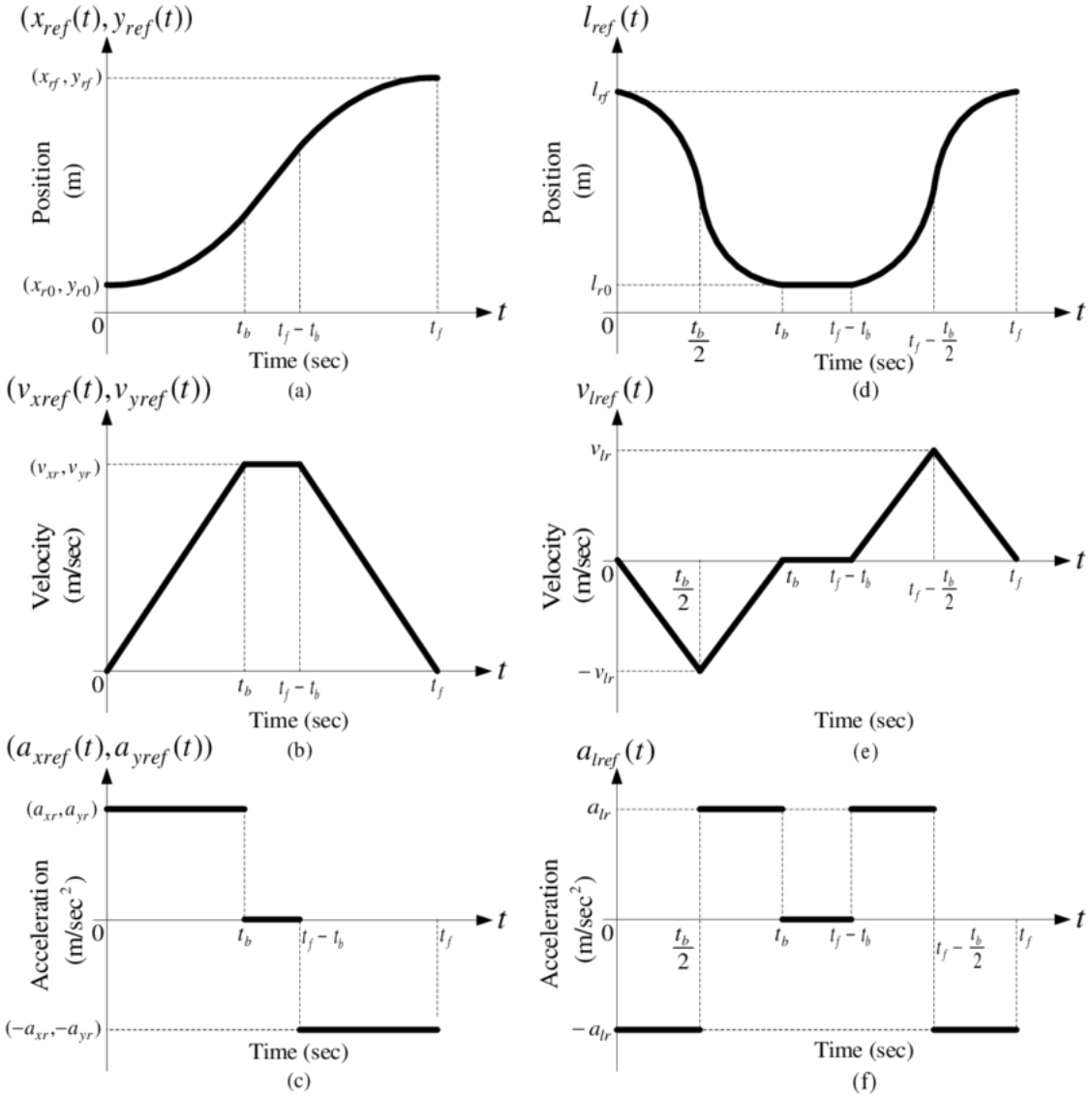
Két adott pont közötti pályatervezés világkoordinátákban τ idő alatt a következő módon végezhető el [12]:

$$s(t) = s^A + \lambda(t)(s^B - s^A), \quad 0 \leq t \leq \tau \tag{3.25}$$

$\lambda(t)$ függvény a TCP sebesség-törvényszerűségét határozza meg, amely különböző típusú lehet. Többek között:

- háromszög
- trapéz
- parabola
- ciklois

alakú lehet. A 3.14. ábra bemutatja trapéz alakú (a, b, c) és háromszög alakú (d, e, f) sebesség-törvényszerűség esetén hogyan változik a pozíció, sebesség és gyorsulás az idő függvényében. A sebesség és gyorsulás kezdő- és végértéke alapesetben 0.



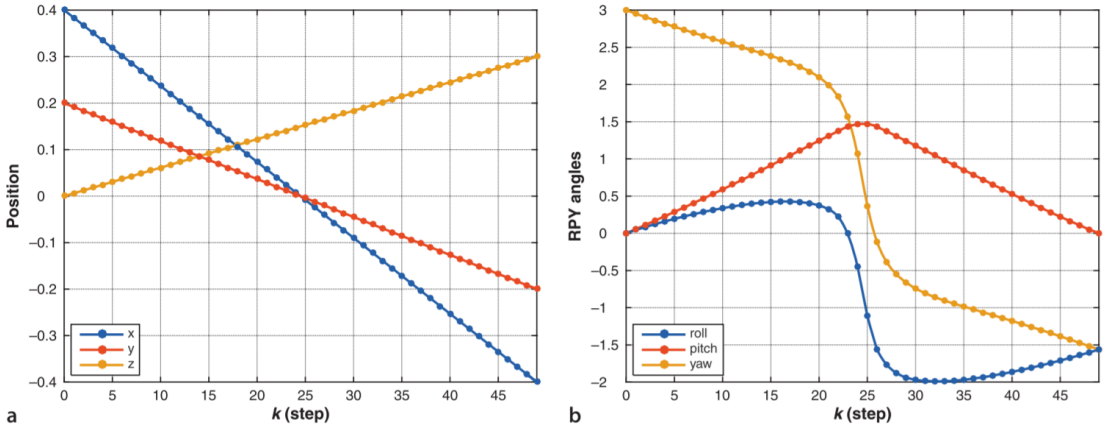
3.14. ábra. Pozíció, sebesség és gyorsulás az idő függvényében ábrázolva trapéz (a, b, c) és háromszög (d, e, f) alakú sebesség-törvényszerűségek esetén

3.6.1. Lineáris pályatervezés

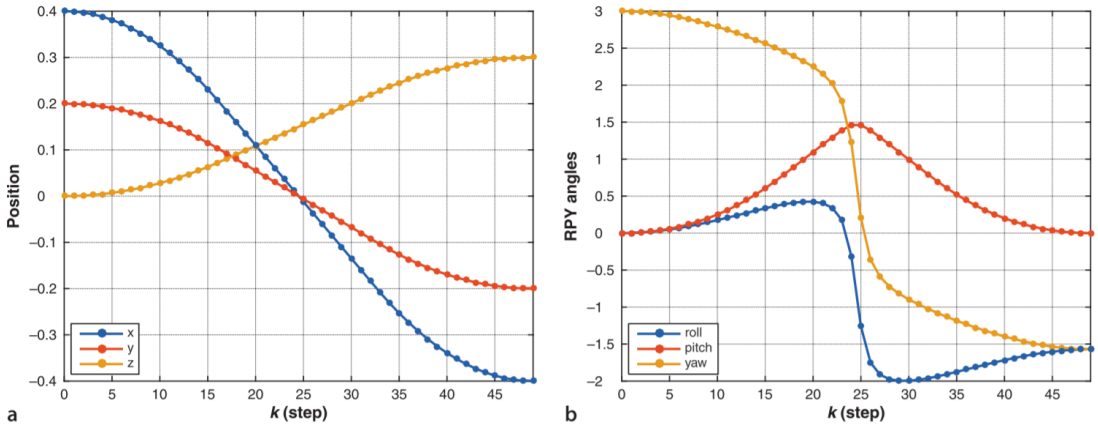
Lineáris pálya esetén a TCP egyenesen vonalon mozog a kezdőponttól a végpontig, miközben az orientációja egyenletesen változik. A számítási módszert Descartes-típusú interpolációnak hívjuk (az angol Cartesian interpolation kifejezésből)[2]:

1. A szakasz megtételéhez szükséges időt apró Δt időintervallumokra bontjuk.
2. Kiszámítjuk TCP új pozícióit minden Δt időegységre.
3. Megoldjuk az inverz kinematikai feladatot minden TCP pozícióra.

A robot mozgatását PTP (Point to Point), azaz pont-pont irányítással végezzük Δt időközzel. Pont-pont irányítás alatt azt értjük, hogy nincs a két pont között definiált



3.15. ábra. Lineáris mozgás esetén a csuklóváltozók értékei [9]



3.16. ábra. Lineáris mozgás és trapéz alakú sebesség-törvényszerűség esetén a csuklóváltozók értékei [9]

pálya. Ha Δt elég kicsit, a mozgás tekinthető folytonosnak. A 3.15. ábrán látható hogyan változnak egy robot csuklóváltozói lineáris pálya esetén. Ha trapéz alakú sebesség-törvényszerűséget alkalmazunk, akkor a pálya kezdeténél sűrűbben helyezkednek el a kiszámolt TCP pozíciók, ami a 3.16. ábrán látható értékeket eredményezi az állandó gyorsulásos szakaszok miatt.

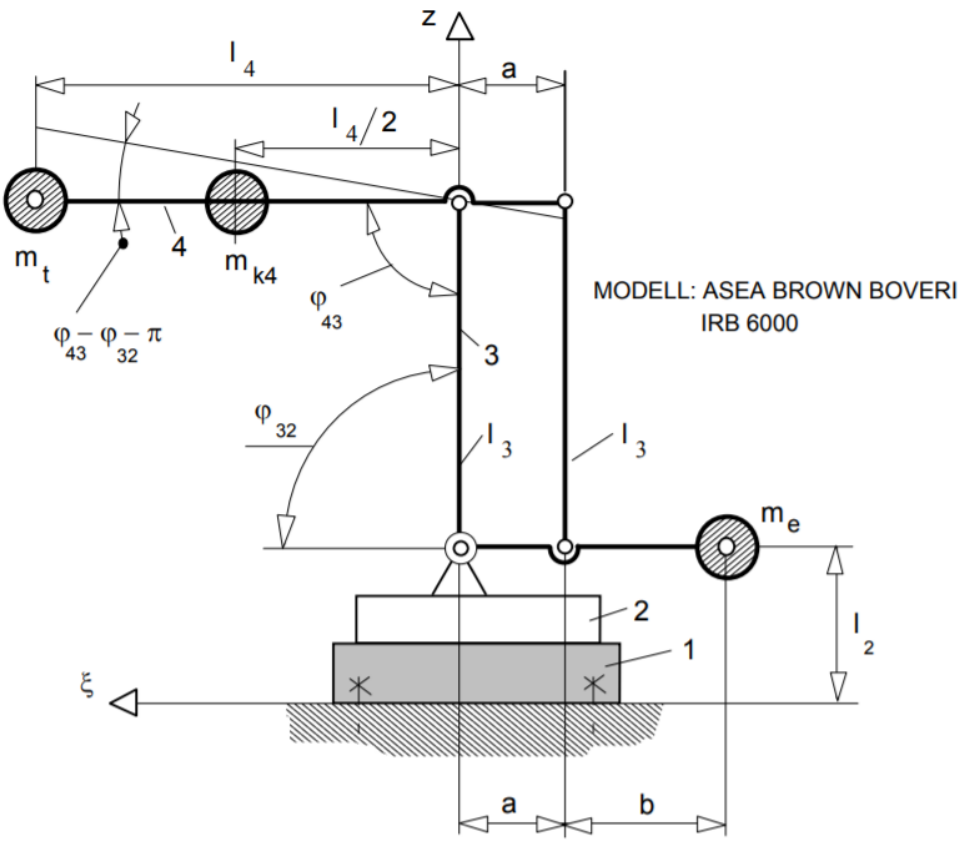
3.7. Robotok tömegkiegyenlítő rendszerei

A robotok mozgásához szükséges nyomaték olyan nagy értékek között mozoghat, hogy sok esetben a hajtómotor önmagában nem elegendő. A szegmensek tömegeit ezért különböző tömegkiegyenlítő rendszerekkel [4] igyekeznek egyenletesebbé tenni a mozgástartományban. Használatuk leginkább a függőleges síkú csuklókaros (RRR) robotoknál terjedt el. Elve az, hogy a tagok saját súlyából adódóan ne terhelje nyomaték a hajtómotorokat. A gyakorlatban a következő tömegkiegyenlítési módok terjedtek el:

- ellensúllyal való tömegkiegyenlítés

  - közvetlenül a kiegyenlítő karhoz kapcsolódó ellensúly
  - valamilyen áttételen keresztül kapcsolódó ellensúly
- vezérelt hidraulikus vagy pneumatikus hengerek segítségével való tömegkiegyenlítés
- vezérlés nélküli hidraulikus vagy pneumatikus hengerek segítségével való tömegkiegyenlítés
- rugós mechanizmussal való tömegkiegyenlítés

3.7.1. Áttételi mechanizmus segítségével kapcsolódó tömegkiegyenlítés



3.17. ábra. A tömegek és erőkarok IRB 6000 típusú robot esetén [4]

Ez a tömegkiegyenlítési módszer leginkább a KUKA és ASEA robotoknál terjed el. A 3.17. ábra jelöléseivel [4]

$$m_e = \frac{\left(m_t + \frac{m_{k4}}{2}\left(1 - \frac{a}{l_4}\right)\right) l_4 - m_{ki}\frac{a+b}{2}}{a+b} \tag{3.26}$$

tömeggel tudjuk kiegyenlíteni a 4-es kart a mozgás teljes tartományában, ahol m_{ki} a kiegyenlítő súlyt tartó kar tömege. A 3-as rúd tömege elhanyagolható.

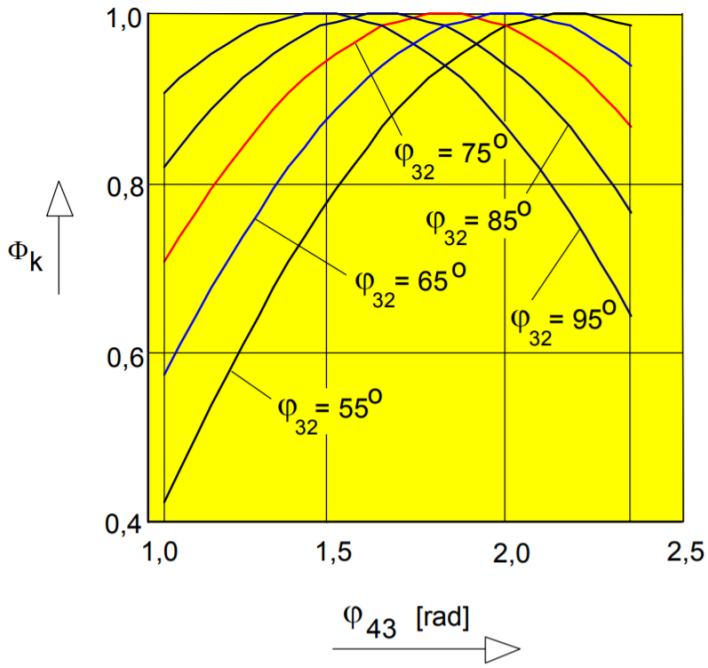
A hajtómotorokat terhelő nyomaték az

$$M_k = \left(m_t + \frac{m_{k4}}{2} \left(1 - \frac{a}{l_4} \right) \right) l_4 \; g \cos(\varphi_{43} + \varphi_{32} - \pi) \tag{3.27}$$

egyenlettel határozható meg. A kiegyenlítetlen önsúlyból és terhelésből adódó terhelő nyomaték egy konstans és Φ_k dimenziótlan tényező szorzataként állítható elő.

$$\Phi_k = \cos(\varphi_{43} + \varphi_{32} - \pi) \tag{3.28}$$

A 3.18. ábra mutatja Φ_k függését φ_{32}-től és φ_{43}-tól. A nyomaték változása kiegyenlítés nélkül elérheti akár a 70%-ot is, ami a motorok számára nem kívánatos, mert telítésbe mennek át.



3.18. ábra. Φ_k függése φ_{43} és φ_{32} értékétől [4]

4. fejezet

Robotkar

4.1. Hardver

Ez az alfejezet a szükséges hardverek kiválasztási szempontjait azok működését tárgyalja.

Manapság sok gyártó kínál eladásra kisméretű, teljesen összeszerelt robotkart vagy különálló vázat különböző szabadsági fokkal és hajtással. Én amellett döntöttem, hogy a vázat és a hajtást külön-külön választom ki, mivel nem találtam olyan párosítást, ahol mind a váz, mind a hajtás jó minőségű lett volna vagy olcsóbb lett volna mintha külön-külön vesszük meg őket.

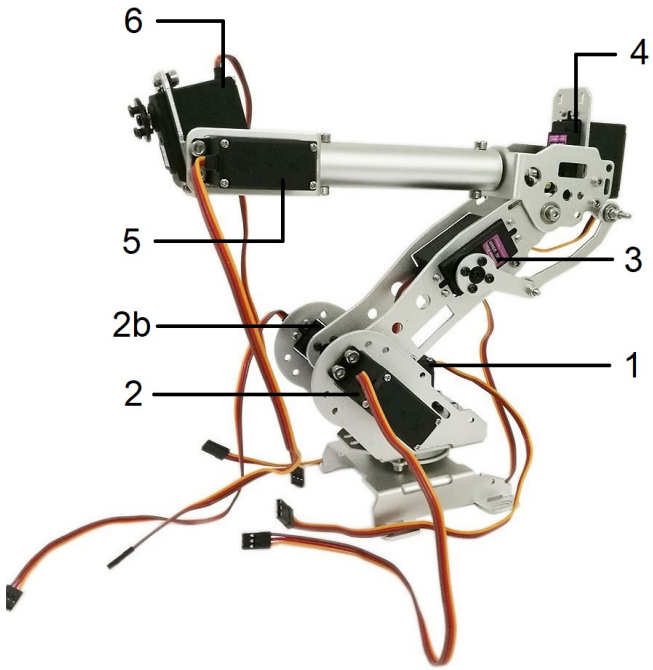
Hajtás szempontjából a szervomotoros megoldás mellett döntöttem, ugyanis a legolcsóbb léptetőmotoros robotkarok is egy nagyságrenddel drágábbak voltak, mint az átlagos szervomotoros robotkarok. További előnye, hogy a szervomotorok nagyobb forgatónyomatékot képesek biztosítani nagy sebesség mellett.

4.1.1. Váz

Az alábbi szempontok voltak a váz kiválasztásánál:

- 6 szabadsági fokkal rendelkezzen
- standard méretű szervomotorhoz legyen tervezve
- az összeszerelt robotkar méretét tekintve férjen el egy asztalon
- az ára minél alacsonyabb legyen

A választás a Shenzhen Doit Technology Co., Ltd. egyik termékére esett. Egy nagyon könnyű, alumíniumlemezből vágott és hajlított váz, amely 6 szabadságfokú, de 7 szervomotor szükséges hozzá, mert az egyik tengelyhez 2 szervomotor van hozzárendelve. Mind a 7 szervomotor helye megfelel a 40x20x38,5mm méretekkel rendelkező szabványnak.



4.1. ábra. A Shenzhen Doit Technology Co., Ltd. által forgalmazott robotkar [22] és a szervomotorok elnevezései

A vázon két változtatást végeztem. Az első változtatás oka az volt, hogy szükség volt egy ellensúlyra, ugyanis az 5-ös és 6-os szervomotorok súlya nyugalmi állapotban is nagyon nagy nyomatékkal terhelik a 3-as szervomotort. Ezt két alumínium kerettel és a 3-as szervo helyének megnagyobbításával értem el, amiknek köszönhetően egy tároló eszközt lehetett kapcsolni a tengelyhez az ellensúly számára. Az egyik keret körülöleli a 3-as szervot, így a másik keret (egy csapágyat felhasználva) minkét oldalához képes csatlakozni. A korábbi fejezet 3.7. egyenletét használva kiszámolhatjuk, mekkora kiegyenlítő tömegre van szükség.

Ismerjük a tömegeket és hosszokat:

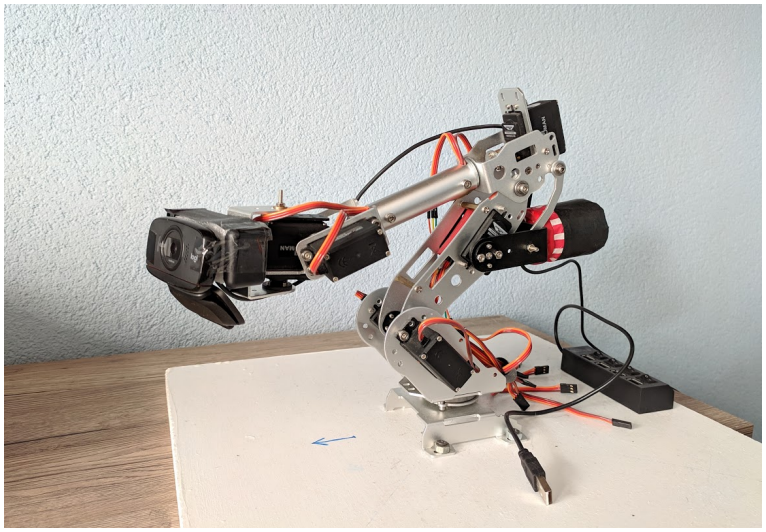
- A robotkar terhelése végén található 2db szervomotor és a kamera (amelyek később kerülnek bemutatásra) tömege: $m_t = 2 \cdot 56 + 88 = 200g$
- $m_{k4} = 24g$
- $m_{ki} = 7g$
- $l_4 = 18cm$
- $a = 3,5cm$
- $b = 6cm$

$$m_e = \frac{\left(m_t + \frac{m_{k4}}{2}\left(1 - \frac{a}{l_4}\right)\right) l_4 - m_{ki}\frac{a+b}{2}}{a+b} = \tag{4.1}$$

$$= \frac{\left(200 + \frac{24}{2}\left(1 - \frac{3,5}{18}\right)\right) 18 - 7\frac{3,5+6}{2}}{3,5+6} = 394g \tag{4.2}$$

a szükséges kiegyenlítő tömeg.

A másik változtatásra azért volt szükség, mert a 6-os szervomotorhoz kellett valamilyen módon rögzíteni a kamerát. Lehetett volna közvetlenül a szervo végéhez is csatlakoztatni a kamerát (vagy annak tartóját), de akkor nem lenne eléggé stabil és a kamera középpontja is el lenne tolva. Ezt szintén két alumíniumkerettel értem el az előző átalakításhoz hasonlóan. A változtatásokat a 4.1. ábrán lehet látni.



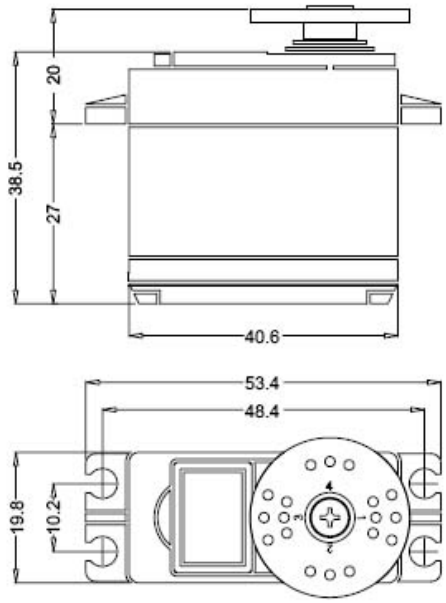
4.2. ábra. A módosított robotkar

4.1.2. Hajtás

Hajtómotornak a DOMAN S2000MD [13] és S2003MD [14] szervomotorokat választottam. A különbség köztük csak annyi, hogy az S2000MD 180°-os szögtartományban képes forogni, még az S2003MD 270°-os tartományban. A 4.1. ábra jelölései szerinti 1, 2, 2b és 3 szervomotorok hasznos szögtartománya nem nagyobb, mint 180°, ezért ide elég volt az S2000MD. A szervok tulajdonságai a következők:

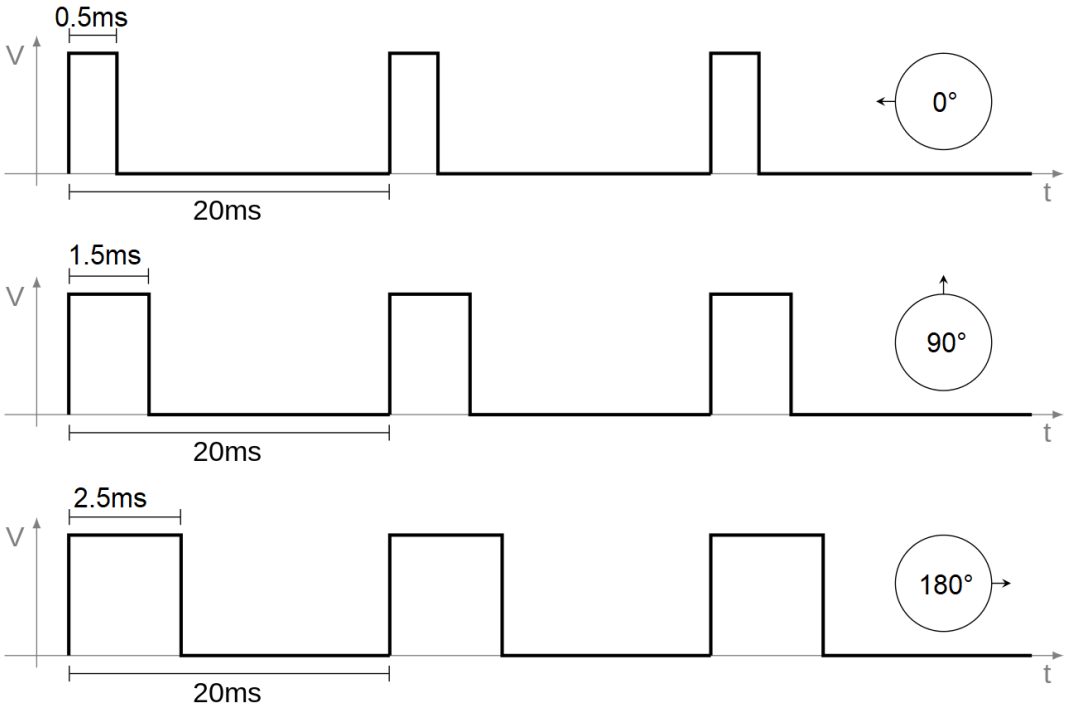
- 4,8V-7,2V tápfeszültségről üzemelnek
- sebességük 6V tápfeszültség mellett: $\frac{60°}{0,16s}$
- forgatónyomatékük 6V tápfeszültség mellett: $21,5kg \cdot cm$
- méreteik: 40x20x38,5mm
- súlyuk: 56 gramm

- fém fogaskerekekkel rendelkeznek



4.3. ábra. Az S2000MD és S2003MD szervomotorok méretei [14]

A kiválasztásnál a fő szempont a nagy forgatónyomaték és fém fogaskerekek mellett az volt, hogy a vezérlő elektronika digitális legyen a gyorsabb reakcióidő és egyenletesebb nyomaték miatt.



4.4. ábra. Az S2000MD szervomotorok szögének állítása PWM segítségével [16]

Ahogy a 4.4. ábra is mutatja, a szervokat vezérlő PWM jel 20ms periódusidejű és az impulzus 500μs és 2500μs között változhat, aminek hatására a szervo kezdő- és végállása között állítható. Az S2003MD szervo 270°-os szögben képes mozogni, ezért a szükséges impulzus hosszát az

$$l_1 = \frac{d_1}{270°} \cdot 2000\mu s + 500\mu s \tag{4.3}$$

képlettel lehet kiszámolni, ha l_1 az impulzus hossza μs-ban és d_1 a kívánt szög fokban. Az S2000MD szervo 180°-ban képes forogni, azért az

$$l_2 = \frac{d_2}{180°} \cdot 2000\mu s + 500\mu s \tag{4.4}$$

képlettel kell kiszámolni az impulzus hosszát.

Tápforrásnak egy 6V-os, 15A-es kapcsolóüzemű tápegységet használtam. A motorok adatlapján nem említi, hogy milyen teljesítménnyel üzemelnek, de hasonló szervomotorok adatlapja alapján maximum 1,3A az áramfelvételük. A 6V és földpont közé egy 1000μF-os kondenzátor helyeztem el, hogy ha a motorok egy gyors elfordulás hatására hirtelen nagy áramot vesznek fel, akkor is stabil maradjon a feszültségszint.

4.1.3. Elektronika

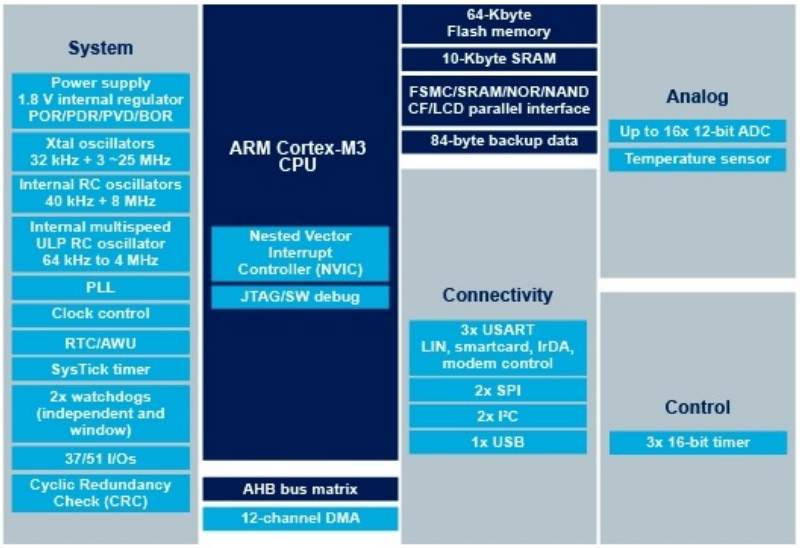
Az elektronikának mindössze annyi a feladata, hogy fogadja a MATLAB által küldött adatokat és ezek alapján állítsa elő a megfelelő PWM jelet a megfelelő lábon. Az elektronika része egy mikrovezérlő, aminek a virtuális soros portjára küldi a MATLAB mind a 6 szervonak szánt szöget, ciklikusan. (A 2b szervo szöge kiszámítható a 2-es szervo szögéből, ezért azt újra nem küldi el.) A szög értéke 0° és 360° közötti decimális érték két tizedesjegy pontossággal. Az USART jellegéből adódóan 8N1 paraméterbeállítás mellett ez fizikailag 4 üzenetben küldhető el. Mindig 4 üzenet generálódik, mivel $'\%06.2f'$ formátumban történik a kiírás MATLAB-ban. Ezt egy logikai analizátor segítségével ellenőriztem is. A 4.1.2. alfejezetben említett módon az általam használt szervomotorok 20 ms periódusidejű PWM segítségével vezérelhetők. Ha 6-szor szeretnénk 4 üzenetet küldeni soros porton keresztül 8N1 paraméterbeállítások mellett, akkor $\frac{6 \cdot 4 \cdot 10bit}{20ms} = 12000\frac{bit}{s}$ minimális sebességre van szükségünk.

Ezek alapján az alábbi elvárások voltak a mikrovezérlővel szemben:

- virtuális soros port (VCP) kialakításának lehetőség, legalább 12 $\frac{kbit}{s}$ sebességgel
- legalább 7 időzítő (timer) modul vagy csatorna
- megfelelő számú I/O láb
  - 2db a VCP számára
  - 7db a szervomotorok számára

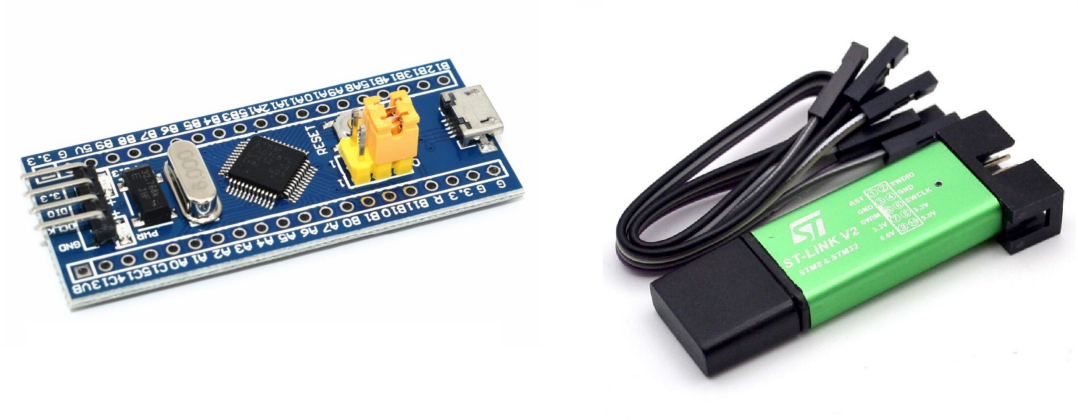
– 1db egy nyomógomb számára

A szervomotorokat nem lett volna muszáj közvetlenül a mikrovezérlő lábaira kötni. Ha sok szervomotor lett volna a rendszerben vagy kevés lábbal rendelkező mikrovezérlőt használtunk volna, akkor egy külső IC-vel is megoldható lett volna. Egy ilyen IC a PCA9685, amivel I^2C interfészen keresztül tud kommunikálni a mikrovezérlő. Ezzel lecsökkenthetnénk a szükséges lábak számát 2-re, akár 16 PWM előállítsa mellett 12 bites pontossággal. Jelen esetben erre nincs szükség, mert kevés szervomotort kell vezérelni és csak a költséget növelné.



4.5. ábra. F103C6T8 tulajdonságai [21]

Az STM32F103C8T6-ot választottam, ami egy ARM Cortex M3-as maggal rendelkező mikrovezérlő. Tulajdonságai a 4.5. árbán láthatóak.



4.6. ábra. STM32F103C6T8 minimum system developmnet board [20] és az ST-LINK/V2 programozó és debugger [19]

Az általam választott fejlesztői kártya egy úgynevezett „minimum system development board”, ami a 4.6. ábrán látható, kapcsolási rajza pedig a F.1.1. ábrán. A hozzá forraszthető tüskesorokkal egy próbapanelre lehet illeszteni és a szervomotorokat, kapcsolót és tápfeszültségét is ezen keresztül lehet hozzácsatlakoztatni. Ahogy a neve is mutatja, a kártyán mindössze a mikrovezérlő működéséhez szükséges alap áramköri elemek vannak jelen. Találhatók rajta még BOOT0 és BOOT1 felirattal ellátott tüskék. Ezeket egy-egy jumper segítségével lehet 0-ra vagy 1-re állítani. Ezek befolyásolják, hogy újrainduláskor honnan kezdje el olvasni a kódot. Egy micro USB csatlakozó is van a panelon, ahova a VCP-hoz használt lábak vannak kivezetve, így könnyen lehet csatlakoztatni számítógéphez.

Mikrovezérlő lábai	**Programozó lábai**
PA14	SWCLK
PA13	SWDIO
VDD	3.3V
VSS	GND

4.1. táblázat. Az STM32F103C8T6 mikrovezérlő és ST-LINK/V2 programozó összekötése [15]

A programozáshoz is egy külső programozóra van szükség. Az általam használt programozó egy ST-LINK/V2 típusú programozó és debugger, amit a 4.1. táblázatban látható módon kellett csatlakoztatni a mikrovezérlőhöz, ha Serial Wire Debug (SWD) interfészen keresztül szeretnénk programozni.

4.1.4. Kamera

A videórögzítés és képfeldolgozás is a MATLAB-ban lett megvalósíva. Kamerának egy webkamerát választottam, mivel MATLAB-ban jól használható. Az Image Aquisition Toolbox segítségével könnyen kérhetünk le képet a webkamerától, valamint ha a webkamera engedi, akkor állíthatók egyéb paraméterek is, például:

- felbontás
- fókusz
- írisz (a kamera apertúrájának mérete)
- zoom (közelítés)

Nekünk elsősorban az első kettőt kell állítanunk, de a későbbiekben a többi is hasznos lehet. A MathWorks oldalon listázva vannak az ajánlott gyártók, amik biztosan támogatva vannak. (Link) A választás egy Logitech C525 webkamerára esett a következők miatt:

- Maximálisan 1280x720 felbontásra képes.
- Állítható fókusszal rendelkezik.

- A zoom funkció is szoftveresen alkalmazható a teljes képszenzort kihasználva.

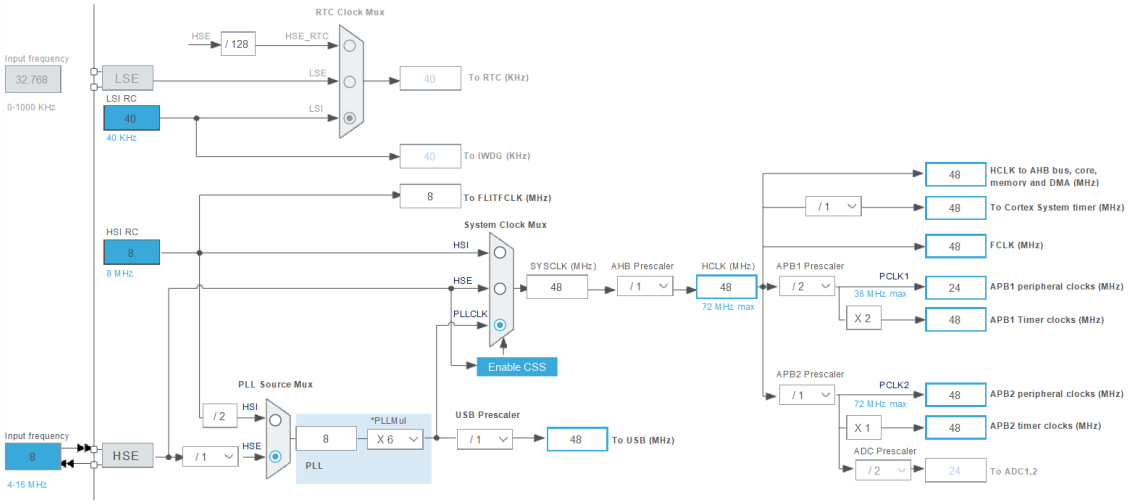
4.2. Szoftver

Ez az alfejezet tartalmazza a robotkarhoz kapcsolódó szoftverelemek fejlesztésének menetét és működését. A képfeldolgozásról a 5. fejezetben lesz szó.

4.2.1. A mikrovezérlő programkódja

A fejlesztés STM32CubeMX és Atollic TrueSTUDIO for STM32 programok segítségével történt.

A CubeMX program egy grafikus felületet biztosít arra, hogy az általunk választott mikrovezérlőt konfigurálni lehessen és létrehozzuk vele az inicializáló kódot. Ki lehet jelölni, hogy a mikrovezérlőn milyen modulokat, interfészeket szeretnénk használni, előkészíti a számunkra szükséges könyvtárakat, valamint meg lehet benne adni az egyes modulok paramétereit. *Clock Configuration* menüjében lehet beállítani az órajeleket. A rendszeróra-jelet 48 MHz-re állítottam, azért hogy az USB által használt 48 MHz-es órajel előállítható legyen. Ennek forrása a panelen található 8 MHz-es kristályoszcillátor.



4.7. ábra. Az órajel konfiguráció

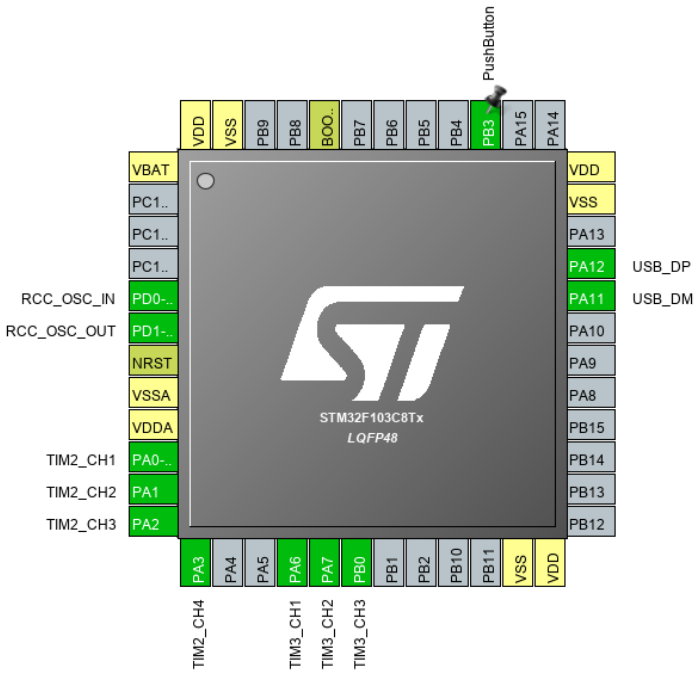
A választott mikrovezérlő TIM2 és TIM3 timer moduljait használtam, amik a következő módon lettek beállítva:

- Prescaler: 47
- Counter period: 19999

Így amilyen számra állítjuk az impulzust, annyi μs lesz a hossza. TIM2 és TIM3 általános célú timer modulok, amik 4-4 csatornával rendelkeznek, amik közül 7-et használtam a 7 szervomotornak. Az egyes csatornákra külön lehet állítani szoftveresen az impulzus hosszát. A virtuális soros port az USB micro-B csatlakozóra lett állítva és a kommunikáció 115200 bit/s sebességgel működik. A $PB3$ lábra egy kapcsoló került, amely egy felhúzó

ellenálláson keresztül csatlakozik a tápfeszültséghez. A kapcsoló megnyomásához rendelt feladat, hogy az összes timer csatornáihoz rendelt impulzushosszt felülírja egy fix értékkel, ami egy alaphelyzetbe állítja a szervomotorokat.

A 4.8. ábrán látható, hogy hogyan lettek konfigurálva a mikrovezérlő lábai.

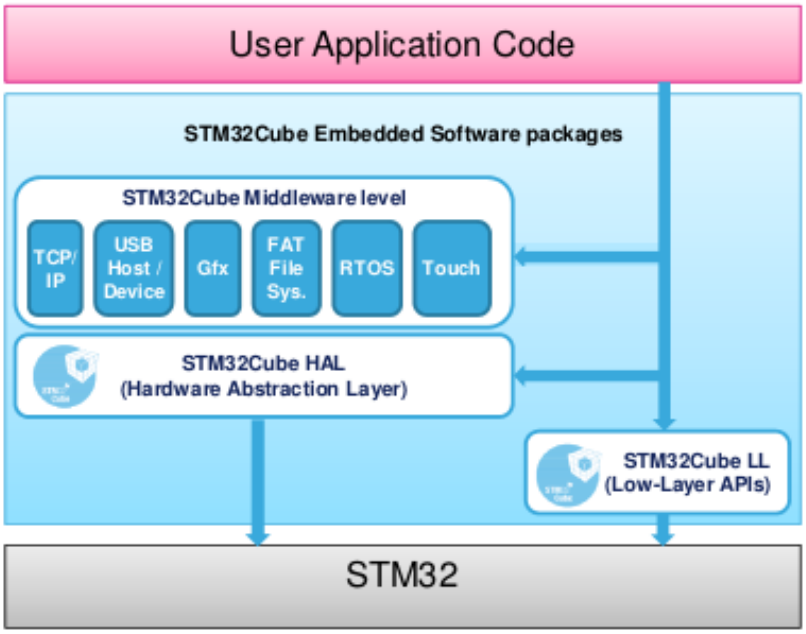


4.8. ábra. A mikrovezérlő lábkiosztása

Miután elvégeztünk minden beállítást, az STM32CubeMX legenerálja nekünk a kiindulási kódot, amit az Atollic TrueSTUDIO fejlesztői környezetben meg tudunk nyitni és továbbfejleszteni. Ahogy a 4.9. ábra mutatja, a felhasználói kód három lehetséges úton érheti el a hardvert:

- A middleware-en keresztül.
- A hardware abstraction layer-en (HAL) keresztül.
- A low-layer (LL) API-n keresztül.

A HAL könyvtáraival valósítottam meg a feladatokat. Ez ugyan nem olyan hardverhez közeli és gyors, mint a LL, de nem volt szükség nagyon nagy sebességre és ezzel hordozható kódot lehet vele létrehozni, ami más mikrovezérlőkön is működik.



4.9. ábra. A hardver elérési módjai a felhasználói kódból. [10]

4.2.2. Vezérlő szoftver

A vezérlő szoftver az a programkód, ami a MATLAB-ban fut és az útvonaltervezést, vezérlést végzi. A Peter Corke által készített Robotics Toolbox [8] 10.3.1 verzióját használtam.

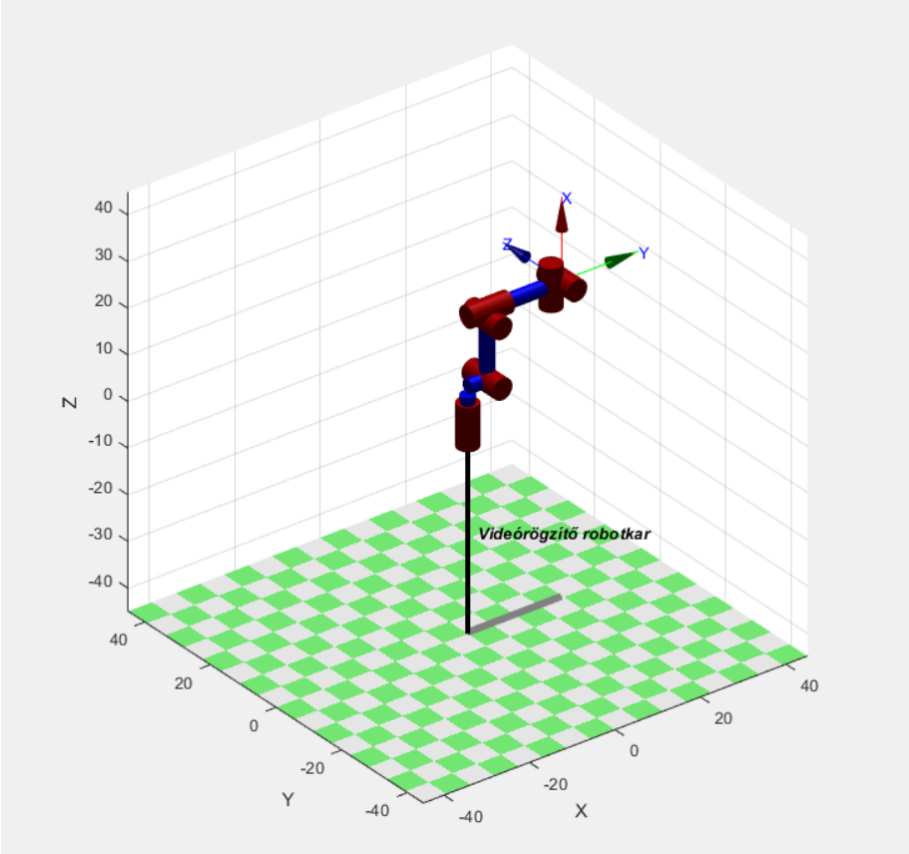
4.2.2.1. A robotkar reprezentálása

Megmértem a robotkar fizikai méreteit és a 3.4.4. alfejezetben tárgyalt módon felvettem a Denavit-Hartenberg paramétereket az egyes szegmensekre:

j	θ	d	a	α	$ofszet$
1	q_1	8	4,5	$-\frac{\pi}{2}$	0
2	q_2	0	13	π	$-\frac{\pi}{2}$
3	q_3	0	-2	$-\frac{\pi}{2}$	$-\pi$
4	q_4	15	0	$\frac{\pi}{2}$	$\frac{\pi}{2}$
5	q_5	0	-2,5	$-\frac{\pi}{2}$	$-\frac{\pi}{2}$
6	q_6	0	0	0	$-\frac{\pi}{2}$

4.2. táblázat. A robotkar Denavit-Hartenberg paraméterei. d és a centiméterben lettek megadva. θ, $alpha$ és az ofszet radiánban értendők.

Egy 6 szabadsági fokkal rendelkező, RRRRRR típusú robotról van szó, ezért q a csuklóváltozó, amelyből 6db van. Az ofszet értéke azt befolyásolja, hogy q_n értéke mennyivel van kezdetben elforgatva. A szögek radiánban, a távolságok centiméterben lettek megadva.



4.10. ábra. A robotkar a 4.2. táblázatban megadott paraméterek alapján, ha minden csuklóváltozó 0-ra van állítva. A nyilak a TCP koordináta-rendszerét mutatják.

4.2.2.2. Útvonaltervezés

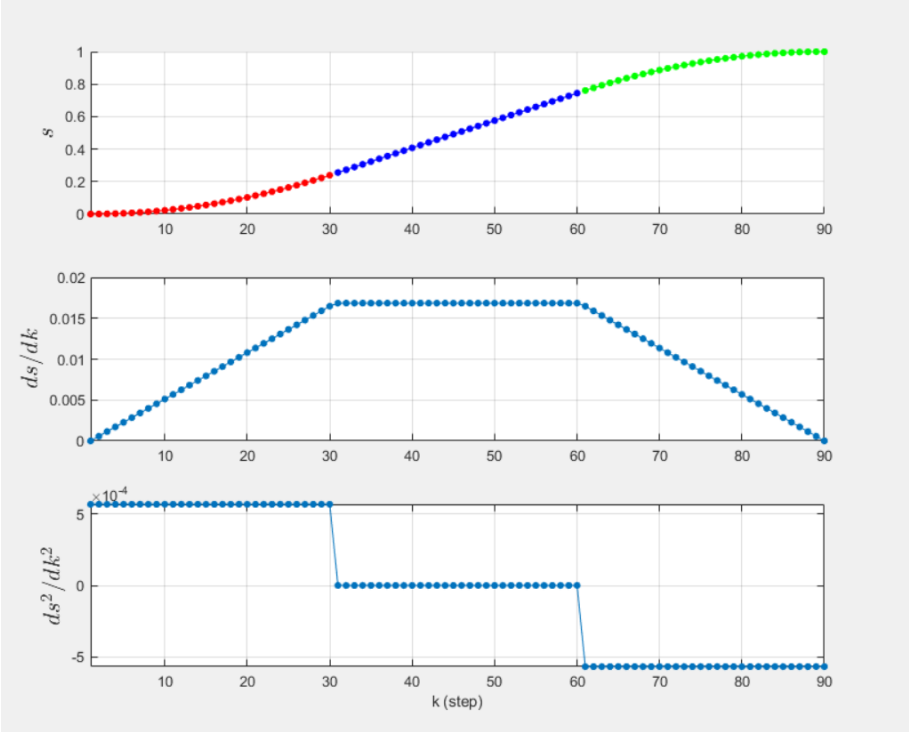
Az útvonalak pontjait a TCP koordináta-rendszerének világkoordináta-rendszerbeli helyével jellemezzük és leírásához homogén transzformációs mátrixot használunk.

Lineáris útvonal esetén a 3.6.1. alfejezetben leírt módszert tudjuk alkalmazni. A következő adatokból kell kiindulnunk:

- kiindulási pont
- végpont
- sebesség-törvényszerűség
- Δt időegység

A Δt időegység a szervomotorok PWM jelének periódusideje: 20ms, ennél sűrűbben ugyanis nem érdemes kiszámolni a pozíciókat.

A 4.11. ábrán látható trapéz alakú sebességtörvényszerűséget alkalmaztam, amelynél a lépések egyharmada alatt gyorsul fel maximális sebességre és egyharmada alatt lassul le. A kezdeti- és végsebesség egyaránt 0.



4.11. ábra. Az alkalmazott sebesség-törvényszerűség távolsága, sebessége és gyorsulása a lépések függvényében, 90 lépés esetén.

A felhasználó által megadott idő alatt kell megtenni az utat. Más paraméter alapján is meg lehetne határozni a pontok számát, mint a maximális sebesség vagy gyorsulás, azonban a mozgás teljes időtartama a legszemléletesebb. Az időtartam (t_{len}) és Δt hányadosa adja meg a pontok számát. A kezdő és végpont közötti többi pontot interpoláció segítségével meghatározzuk. A trapéz alakú sebesség-törvényszerűség hatására az útvonal elején és végén sűrűbben helyezkednek el pontok, ami lassabb mozgást eredményez.

Ezek után megoldjuk az inverz kinematikai feladatot az összes pontra. A robotkar geometriájából adódóan csak a numerikus módszer jöhet szóba, mivel az orientációt nem gömbcsuklóval állítja. A használt algoritmus a Levenberg–Marquardt módszert alkalmazza, ami hasonló a Newton-Raphason módszerhez, csak robusztusabb. Ha távol vannak a kezdő csuklóváltozók értékei a megoldástól, akkor kevesebb lépésben megtalálja a megoldást, mint a Newton-Raphason módszer, közeli csuklóváltozóknál ugyanúgy működik. Addig fut, ameddig végre nem hajt 500 iterációt vagy a hiba le nem csökken 10^{-10} alá. Mivel sorban az útvonal pontjaira oldjuk meg az inverz kinematikai feladatot, ezért az egyik ponthoz talált csuklóváltozók elég jó kezdeti értéket adnak az algoritmusnak a soron következő ponthoz.

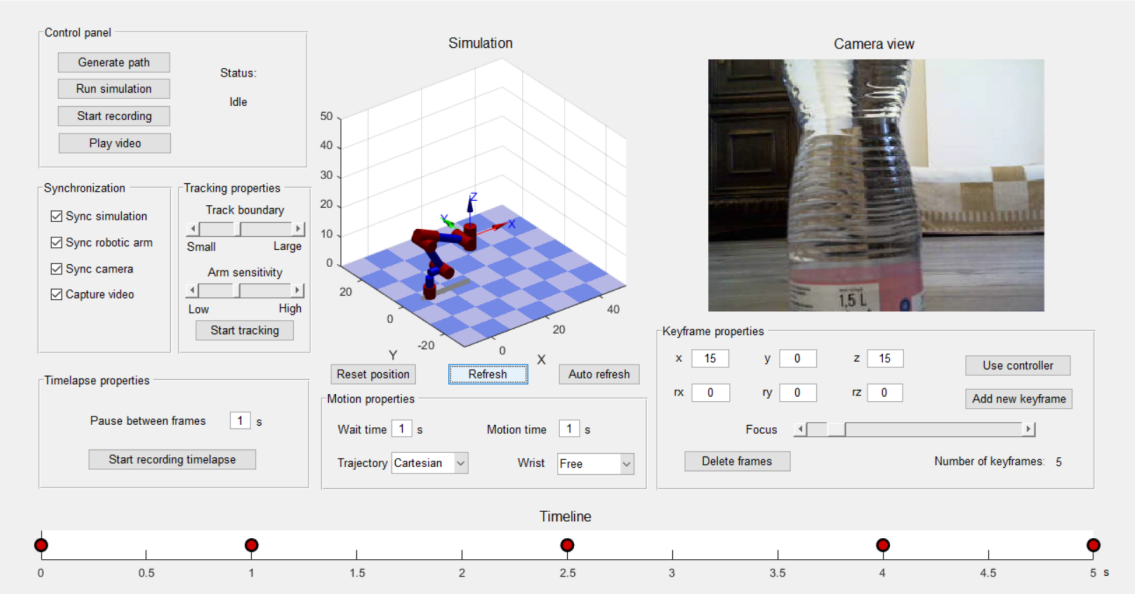
Spline alapú útvonaltervezés Ha nem egyenes, hanem másfajta útvonalon szeretnénk mozogni, azt is megtehetjük, csak az út pontjait kell a térben meghatároznunk. Erre egy jó példa, ha néhány pontra spline-t illesztünk és ezen megyünk végig. A spline egy

szakaszosan polinomokkal leírható görbe. Ha a megadott pontokat szétbontjuk x, y és z koordináták sorozatára és ezekre legeneráljuk egyenként a spline-okat, akkor a hármuk összessége megadja a térbeli spline-t. Ezen felveszünk $\frac{t_{len}}{\Delta t}$ számú pontot, amikre elvégezzük az inverz kinematikai feladatot. Az egyes pontokhoz tartozó orientációt meghatározhatjuk a lineáris útvonaltervezésnél használt módon interpolációval.

4.2.2.3. Grafikus felhasználói felület

A könnyebb kezelhetőség érdekében egy grafikus felhasználói felület is készült a robotkarhoz, amelyen keresztül vezérelhetjük a robotkart és vizuális visszajelzést kapunk.

A *Keyframe properties* panelen a világkoordináta-rendszerben adhatjuk meg TCP koordináta-rendszerét. (A keyframe szót az animálásnál szokták alkalmazni. A mozgás elején és végén lévő helyre és orientációra utal.) Az *Add new keyframe* gombra kattintva ezt a koordináta-rendszert hozzáadja egy tömbhöz, ahonnan pályageneráláskor kerül kiolvasásra. A gomb megnyomásakor egy másik tömbbe raktározza el a program a *Motion properties* panelon található információt. A *Motion time* megadja hány másodperces legyen az út a korábbi és az újonnan hozzáadott koordináta-rendszerek között. A *Trajectory* menüben a *Cartesian* és *Spline* opciók közül választhatunk, hogy a pontok között egyenes vonalon mozogjon vagy egy spline-t illesszen rájuk. A *Wrist* menüben választhatunk, hogy *Free*, azaz szabad mozgást vagy *Z axis only*, azaz csak Z tengely körüli forgást engedünk meg az orientációnak. A *Wait time*-nál megadhatjuk, hogy egyenes vonalú mozgás esetén mennyi időt várjon a robotkar, mielőtt megkezdi a mozgását a következő útvonalon. A *Keyframe properties* panelon állíthatjuk még a kamera fókuszát vagy törölhetjük a megadott TCP koordináta-rendszereket. A *Number of keyframes* mellett található szám jelzi, hogy aktuálisan hány TCP koordináta-rendszert adtunk hozzá. A *Timeline* azt mutatja, hogy a hozzáadott mozgások mennyi ideig tartanak, sorrendben.



4.12. ábra. A grafikus kezelőfelület egy tervezett útvonallal

A *Control panel* feliratú panelen található *Generate path* gombbal tudunk a létrehozott tömbökből útvonalakat generálni. A *Run simulation* gomb megnyomásával elindíthatunk egy szimulációt. A *Simulation* felirat alatt látható virtuális robotkar végigmegy az útvonalakon olyan sebességgel, mint a valódi. Ez tesztelésnél hasznos funkció, mert nem kell közvetlenül a robotkaron kipróbálni, hogy mit generáltunk.

A *Synchronization* panelon választható ki, hogy a *Refresh* gomb megnyomása után mi frissüljön. Attól függően, hogy mi van bepipálva, az alábbiak történhetnek:

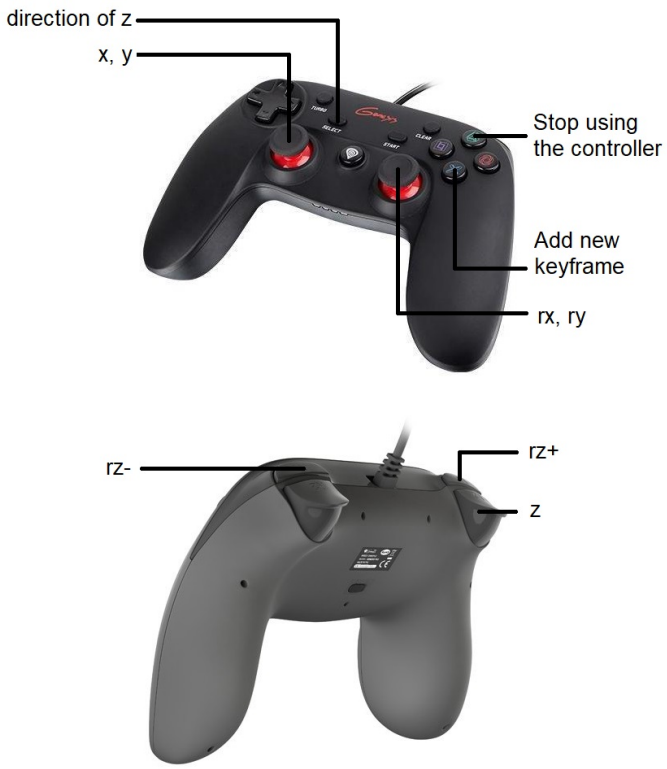
- a virtuális robotkar felveszi a *Keyframe properties* panelon aktuálisan jelen lévő pozíciót és orientáció
- a program a *Keyframe properties* panelon aktuálisan jelen lévő pozíció és orientáció alapján meghatározza a csuklóváltozók értékét és soros porton elküldi a robotkarnak.
- a program kér egy képet a kamerától és a *Camera view* felirat alatti képet lecseréli

Az *Auto refresh* gomb elindít egy időzítőt, ami rendszeresen meghívja a *Refresh* gomb callback függvényét. A MATLAB elég lassan tudja végrehajtani ezeket a frissítéskor szükséges feladatokat, és 3 másodpercnél rövidebb idő alatt nem tud minden végrehajtódni, ezért 3 másodpercenként hívódik meg a frissítő függvény. Az időzítő mindaddig működik még a felhasználó újra meg nem nyomja az *Auto refresh* gombot. A *Use controller* gomb megnyomása után a program csatlakoztatja az USB kontrollert és egy ciklusba ugrik, ahol folyamatosan olvassa a kontroller gombjainak állását. A gombokkal lehet növelni vagy csökkenteni a *Keyframe properties* panel változóinak értékét. A 4.13. ábra mutatja, hogy melyik gombhoz melyik változót rendeltem. Ez nagyban megkönnyíti a TCP koordinátarendszerek felvételét, mert a kontrollerrel intuitívan tudjuk mozgatni a robotkart, miközben látjuk a kamera képét is, ezáltal pontosan a nekünk tetsző pózba tudjuk beállítani. Mivel minden rész frissítése sok ideig tart, ezért lehetőségünk van csak a robotkart szinkronizálni, ilyenkor az *Auto refresh* gomb egy 0,25 másodperces időzítőt indít el. Ez jóval gyorsabb mozgatást teszt lehetővé.

Ha a *Generate path* gombbal legeneráltuk az útvonalat, akkor a *Start recording* gombbal elindíthatjuk a „felvételt”. A program elkezdi vezérelni a robotkart a szimuláción látható módon. Ennél az üzemmódnál is azokat a feladatokat hajtja végig, ami be van pipálva a *Synchronization* panelon. Ha minden be van pipálva, akkor 50 Hz-es frekvenciával:

- A robotkar számára kiküldi a csuklóváltozók értékét.
- A virtuális robotkar is felveszi ugyanazt a pózt, mint a valódi robotkar.
- A kamera képét lekéri a program és a *Camera view* felirat alatt kirajzolja.
- A képet hozzáfűzi egy videó fájlhoz.

Ha ez megvolt, akkor a *Play video* gombbal lejátszhatjuk a legutóbb felvett videót. Minden második képkocka felhasználásával egy 25 fps (képkocka másodpercenként) sebességű videót kapunk, ami pont az Európában elterjedt szabvány.



4.13. ábra. A kontroller és a hozzá rendelt változók [11]

A Timelapse funkció lényege, hogy sokkal lassabban rögzítjük a felvételt, mint ahogy visszajátsszuk, ezáltal olyan illúziót keltve, mintha a videóban sokkal gyorsabban telne az idő. Ezt szintén filmeknél, reklámoknál használják. (Például azt veszik fel, ahogy egy virág nő.) Az útvonalat ugyanúgy kell megtervezni, mint korábban, csak az képkockák közötti szünetet kell megadni és a *Start recording timelapse* gombbal tudjuk indítani a felvételt.

A *tracking* funkcióról a 5. fejezetben lesz szó.

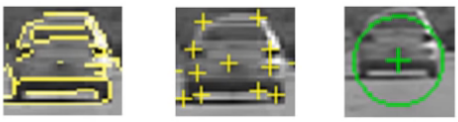
5. fejezet

Képfelismerés és követés

Ez a fejezet a képfelismerésről és követésről szól, valamint arról, hogy hogyan implementáltam.

5.1. SURF (Speeded-Up Robust Feature)

A jellemző (feature) egy kép egy területének pontja vagy leírója, amit képesek vagyunk megbízhatóan felismerni több képen is. Ilyenek például az 5.1. ábrán látható él, sarok vagy SURF jellemző.



5.1. ábra. Különböző jellemzők. Balról jobbra: él, sarok, SURF jellemző. [23]

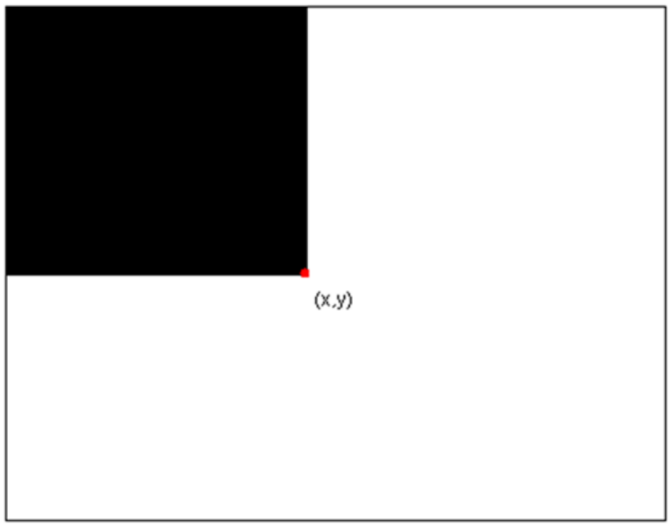
A SURF [3] módszer egy gyors és robusztus algoritmus képek reprezentálásra és összehasonlítására. Alacsony számításigénye miatt gyors, ezért valósidejű képfelismeréseknél és követésnél alkalmazzák.

A jellemzők detektálásának alapja, hogy a Gauss szűrő másodrendű deriváltjának közelítő változatát helyezzük a képre különböző méretekben és irányokban.

5.1.1. Integrálkép

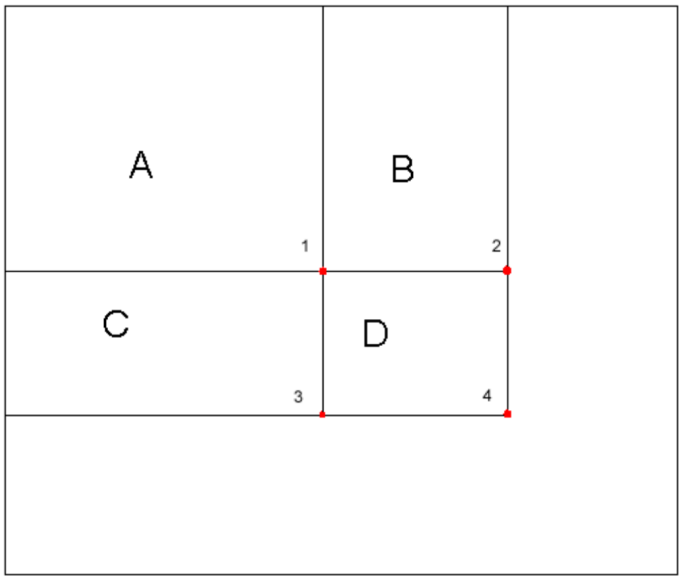
A számításhoz az integrálképet használunk, ami nem más, mint a kijelölt területen az intenzitások összege. Az (x, y) pontban az értéke így számítható ki [1]:

$$ii(x, y) = \Sigma_{x' \leq x, y' \leq y} i(x', y') \tag{5.1}$$



5.2. ábra. (x, y) ponthoz tartozó terület. [1]

Az 5.3. képen a D téglalap intenzitása: $ii(4) - ii(3) - ii(2) + ii(1)$. Az integrálkép lehetővé teszi a gyors számításokat dobozszűrők segítségével.



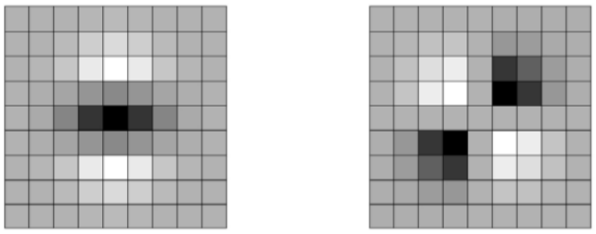
5.3. ábra. A kép A, B, C és D területekre osztva. [1]

5.1.2. Hesse-mátrix alapú detektálás

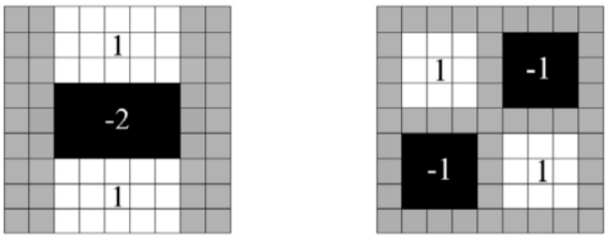
Az érdekes pontok detektálásához Hesse-mátrixot alkalmaznak. Vesszük a Hesse-mátrixot $X = (x, y)$ pontban, σ paraméterrel [3]:

$$H(X, \sigma) = \begin{bmatrix} L_{xx}(X, \sigma) & L_{xy}(X, \sigma) \\ L_{xy}(X, \sigma) & L_{yy}(X, \sigma), \end{bmatrix} \tag{5.2}$$

ahol $L_{xx}(X, \sigma)$ a Gauss szűrő másodrendű deriváltjának és a képnek a konvolúciója X pontban. A Hesse-mátrix determinánsa adja meg a jellemző erősségét. Az 5.5. képen látható, hogy milyen közelítéseket használnak az 5.4. képen látható Gauss szűrő másodrendű deriváltjához képest. Ilyen közelítő értékű dobozszűrőkkel nagyon gyors számításokat lehet végezni. Más módszerekkel ellentétben, a SURF módszernél nem a képet méretezzük át, hanem a szűrőt. Növekvő méretben végigfuttatjuk az integrálképen a szűrőket, 9x9-es mérettel kezdve.



5.4. ábra. 9x9-es méretű Gauss szűrő másodrendű deriváltja y és xy irányban, diszkretizálva. $\sigma = 1{,}2$. A szürke területek értéke 0. [3]

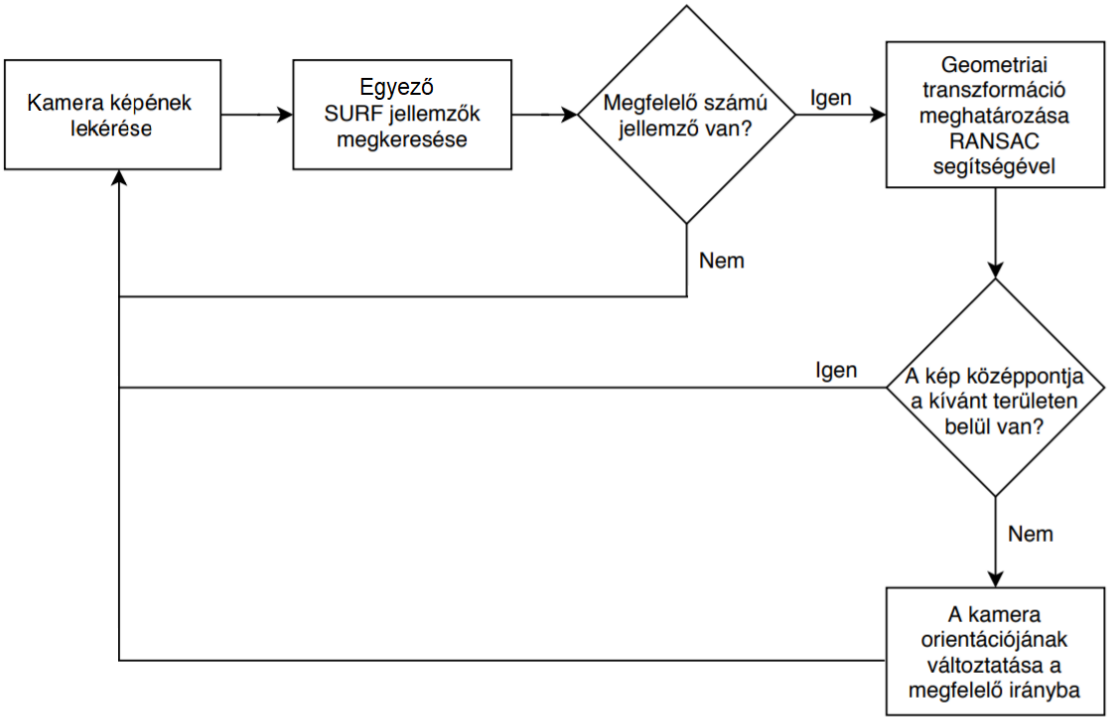


5.5. ábra. Az 5.4. ábrán látható dobozszűrők közelítése. [3]

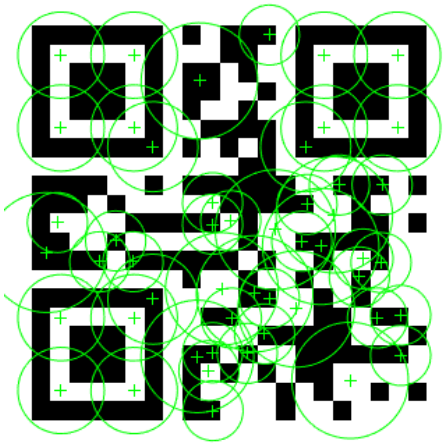
Az algoritmus a ponthoz a körülötte lévő kör alakú terület alapján egy orientációt rendel. Ezután a körülötte lévő négyzet alakú területből meghatározza a SURF leírót.

5.2. Implementáció

Első lépésként kiválasztunk egy képet arról az objektumról, amit detektálni szeretnénk, ez lesz a referenciakép. Átalakítjuk fekete-fehérre, mivel a színeket nem vesszük figyelembe ennél a fajta detektálásnál. Az 5.1. alfejezetben bemutatott módszerrel megkeressük rajta a megfelelő erősségű SURF jellemzőket. (Ez azt jelenti, hogy az erősségük elér egy határértéket.) Én detektálandó objektumnak egy QR-kódot választottam, mivel azon sok SURF jellemző található. A képet a telefonomon jelenítettem meg. Az 5.7. ábrán látható ennek a képe és rajta bejelölve az 50 legerősebb jellemző.

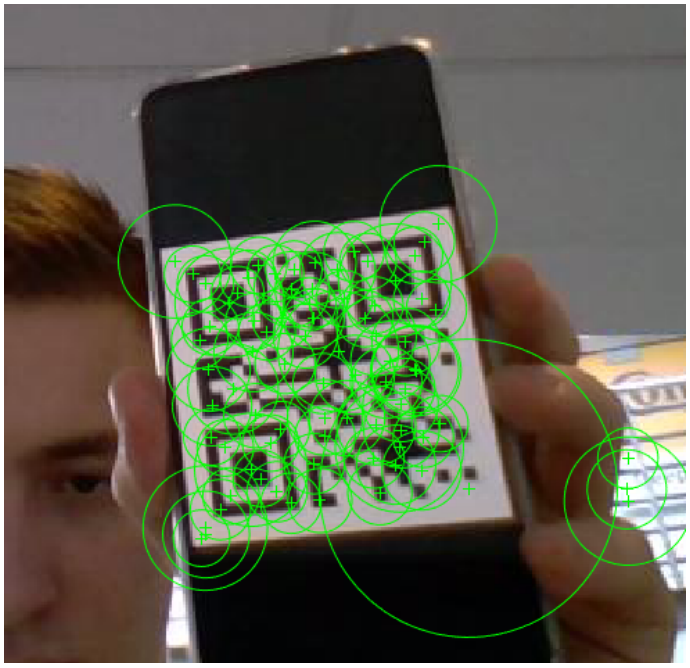


5.6. ábra. A program *Tracking* funkciójának működése blokkdiagramon ábrázolva.



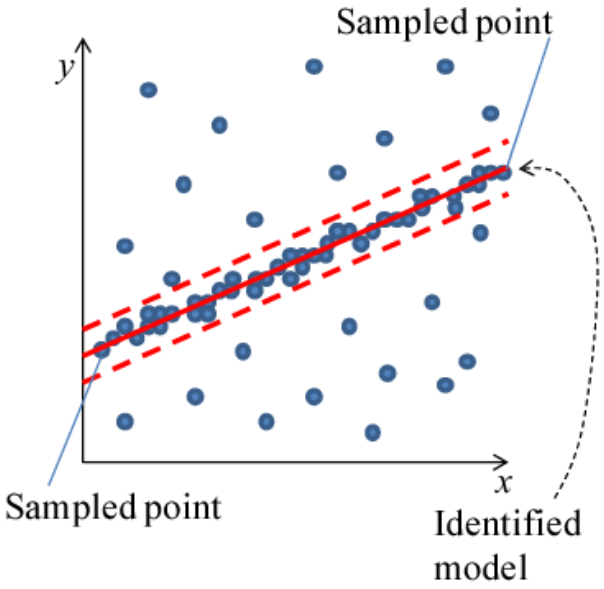
5.7. ábra. A detektálandó kép 50 legerősebb jellemzője. A körök mérete azt jelzi, hogy milyen méretű szűrővel lettek megtalálva.

A grafikus kezelőfelületen található egy *Tracking* feliratú panel, amin a *Start tracking* gomb megnyomásával elindíthatjuk az objektumfelismerést, amelynek folyamatát az 5.6. blokkdiagram mutatja tömören. Kezdetben a kamerától érkező képeken is megkeresi a megfelelő erősségű SURF jellemzőket, ahogy az 5.8. ábrán látható. A színekkel itt sem foglalkozik az algoritmus. A jellemzők közül csak azok érdekelnek, amik egyeznek egy (vagy

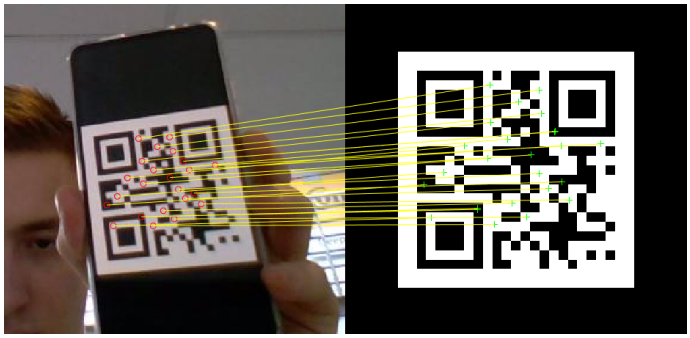


5.8. ábra. A kamera egy képének 70 legerősebb jellemzője. Látható, hogy nem csak a QR-kódon talált a program jellemzőket.

több) jellemzővel a referenciaképről. Attól, hogy egy jellemző egyezik a referenciakép egy jellemzőjével még nem biztos, hogy hasznos. Ha megfelelő számú egyező jellemzőt talált, akkor a RANSAC (random sample consensus) módszer segítségével meghatározza, hogy milyen geometriai transzformációval írható le a kapcsolat a kamera képe és a referenciakép között. A RANSAC az 5.9. árbán látható módon működik. Egy iteratív módszer, amely megpróbál egy minél jobb modellt alkotni és ez alapján eldönteni, hogy mely jellemzők illeszkednek erre a modellre (inlier) és melyek nem (outlier). A modellhez kellő mértékben illeszkedő jellemzők alapján határozza meg a transzformációt. Az hogy mennyi jellemző elegendő választás kérdése. Elméletben legalább 3 darab kell, hogy a transzformációt meg lehessen becsülni, azonban kevés jellemző esetén pontatlan az algoritmus. Kísérletezés alapján a 8 megfelelőnek tűnt. Az 5.10. ábrán azok a jellemzők vannak bejelölve, amik illeszkednek a RANSAC által talált modellre. A referenciakép méretei és a geometriai transzformáció alapján be tudjuk keretezni az objektumot. A grafikus kezelőfelületen az 5.11. ábrán látható *Tracking* módban. A program zöld színnel jelöli be, ha megtalálta az objektumot. Lehet állítani a *Track boundary* csúszkával, hogy mekkora piros téglalapot rajzoljunk a képre. Ha az objektum piros kereszttel jelölt középpontja nincs ezen a téglalapon belül, akkor a program elmozdítja a kamera orientációját a megfelelő irányba. Az *Arm sensitivity* csúszkával azt lehet változtatni, hogy mennyivel változzon az orientáció, ha az objektum kitér a téglalapból.



5.9. ábra. A RANSAC egy modellt alkot a megadott jellemzők alapján és ami egy adott tartományon belül esik az lesz inlier. [7]



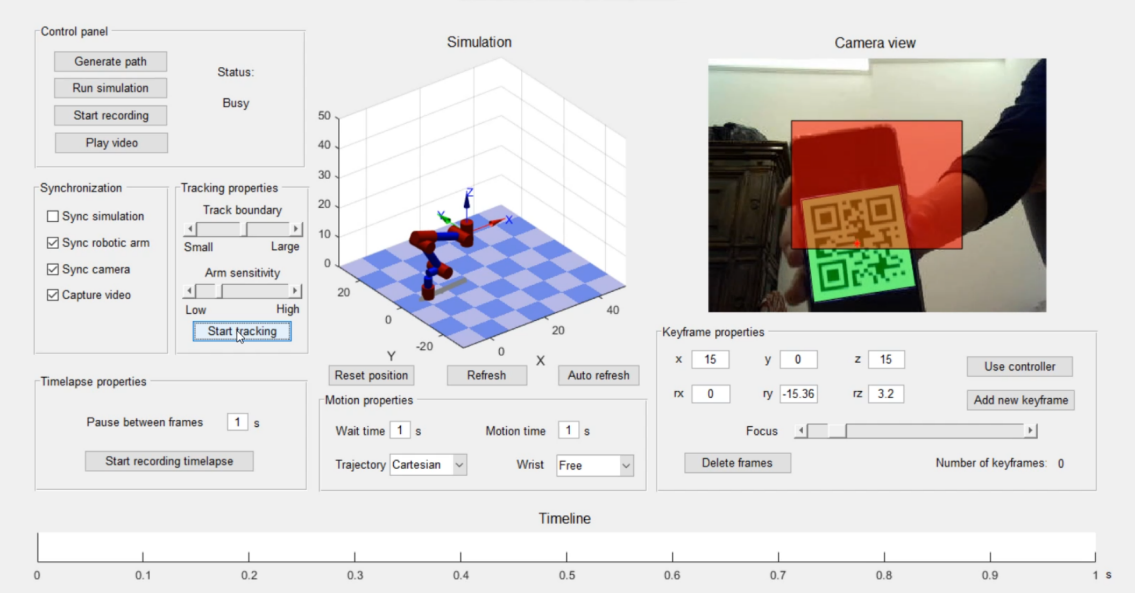
5.10. ábra. A modellre illeszkedő jellemzők. Ezek alapján határozható meg a geometriai transzformáció.

5.2.1. Használt függvények

A MATLAB rengeteg beépített függvénnyel rendelkezik amik ilyen feladatoknál hasznosak lehetnek. Jelen esetben a Computer Vision Toolbox és Image Processing Toolbox függvényeire [17] van szükségünk. A legfontosabbak az alábbiak voltak:

- Az *rgb2gray* segítségével alakíthatjuk át fekete-fehérre a képet, ugyanis a SURF módszer szerint erre van szükség.
- A *detectSURFFeatures* megadja, hogy mely pontokban vannak elég erős SURF jellemzők.

- Az *extractFeatures* függvénynek meg kell adni egy képet meg rajta a jellemzőket és visszaadja nekünk egy vektortömböt a jellemzők tulajdonságaival. Ez a jellemzővektor.
- A *matchFeatures* függvénynek megadhatjuk két kép jellemzővektorainak tömbjét és megadja nekünk, hogy melyek egyeznek.
- Az *estimateGeometricTransform* függvénynek meg kell adni a két kép egyező jellemzőinek koordinátáit két tömbben. RANSAC segítségével meghatározza, hogy milyen geometriai transzformáció segítségével írható le a kapcsolat közöttük.
- A *tformfwd* segítségével geometriai transzformációt hajthatunk végre pontokon. Ennek segítségével tudjuk meghatározni, hogy hol kell bejelölni az objektumot a kamera képén.



5.11. ábra. A grafikus kezelőfelület egy pillanatképe *Tracking* módban.

6. fejezet

Eredmény értékelése és továbbfejlesztési lehetőségek

Az alábbi videók mutatják, hogy hogyan képes végrehajtani a rendszer a feladatokat:

- Itt látható egy szimuláció, amin előre kipróbálhatjuk, hogy hogyan fog mozogni a robotkar. 1. videó linkje[1]
- Ezen a videón látható, ahogy a robotkar önállóan mozog egy előre megtervezett egyenes útvonalon. 2. videó linkje[2]
- A kamera által rögzített videó, miközben a robotkar egy előre megtervezett útvonalon mozog. A felvétel egy külső programmal lett felvéve, mert ha a MATLAB rögzítette volna, akkor nem lett volna tartható a 20ms-os frissítési idő a robotkar számára. 3. videó linkje[3]
- A robotkar irányítása kontrollerrel. Csak a robotkar szinkronizálása volt bekapcsolva. 4. videó linkje[4]
- Az alábbi videón látható a kezelőfelületen keresztül, ahogy a program felismeri a megtanított képet és a kamera orientációját változtatja. 5. videó linkje[5]

A rendszer sikeresen végre tudta hajtani a kitűzött feladatokat. Ahhoz azonban, hogy ténylegesen használható legyen forgatásoknál, minőségbeli javulást kellene elérni. Az alábbi limitációkba ütközött a rendszer:

1. A MATLAB segítségével magas szintű programot tudunk írni, ami miatt a program futása sok időt vesz igénybe. A frissítésnek (virtuális robotkar kirajzolása, inverz kinematikai feladat megoldása, kép lekérése és videó fájlba fűzése, adatok soros porton való elküldése) 20ms alatt végre kellene hajtódnia ahhoz, hogy minden valós

[1] https://youtu.be/VlB4FKXYEJk
[2] https://youtu.be/dx61KkUTgiI
[3] https://youtu.be/ya1TivxxAEk
[4] https://youtu.be/GeaCOQQs5RI
[5] https://youtu.be/GeLof7Y1XY0

időben történjen. Ez egy modern számítógéptől levárható, ha egy alacsonyabb szintű programozási nyelvvel van megírva a program. Szerencsére a soros portra írás, ami a legkritikusabb feladat kellő sebességgel végrehajtható volt, ezért az off-line mozgatás megfelelően működött szoftver szempontjából. A *Synchronization* panel is azért lett hozzáadva, hogy csak a szükséges részek fussanak és azokat ne akadályozza semmi.

2. A MATLAB nem képes grafikus feladatokat külön szálon futtatni, ami még jobban megnehezítette a megfelelő sebességű futást. A virtuális robotkar mozgatása, a kamera képének kirajzolása mind olyan feladatok voltak, amik futását meg kellett várni egymás után mire továbbléphettünk a következő pózra.

3. A robotkar váza nem volt teljesen szilárd és a mozdulatok során ez meglátszott.

4. A szervomotoroknál, különösen a 270°-os mozgástartományúaknál megfigyelhető, hogy nem elég pontos a pozíció-visszajelzésük. Ha a jelenlegi és az új szög között nagyon kicsi a különbség, akkor nem mozdul meg.

5. A kamera egy webkamera volt, aminek a képminősége visszajelzésnek jó, de annyira nem, hogy filmekben lehessen használni.

Ha tovább akarnánk fejleszteni a rendszert, akkor át kellene térni egy alacsonyabb szintű programozási nyelvre. Egy lehetséges út lenne, ha a Qt framework-öt használnánk, amire C++ nyelven fejleszthetünk. Ez megoldaná a szoftverproblémát és a fejlesztési idő is alacsony lenne, mivel a meglévő kódot kellene átírni egy másik nyelvre. A hardverprobléma leginkább csak pénz kérdése. Lehet kapni jobb minőségű robotkarokat, amiken csak minimális átalakítást kell végezni a kamera miatt. A képminőség is javítható másmilyen kamera beszerzésével. Lehet kapni modernebb videokamerákat, amik rendelkeznek olyan interfésszel, amin keresztül össze tudjuk kötni számítógéppel. Ezen keresztül tudjuk vezérelni, valamint a képét is láthatjuk.

Összességében a rendszer prototípusnak megfelelt. Láttuk, hogy milyen kihívásokkal kell szembenézni egy ilyen rendszer elkészítésénél és ezeket milyen eszközökkel oldhatjuk meg, valamint tudjuk milyen irányba kell menni, ha tovább szeretnénk fejleszteni.

Köszönetnyilvánítás

Köszönöm konzulensemnek, Dr. Orosz Györgynek, hogy hasznos tanácsaival segítette ennek a szakdolgozatnak a létrejöttét.

Köszönöm családomnak, hogy támogattak és megteremtették nekem a lehetőséget a tanuláshoz.

Irodalomjegyzék

[1] Dr. Rövid András – Dr. Vámossy Zoltán – Dr. Sergyán Szabolcs: *A gépi látás és képfeldolgozás párhuzamos modelljei és algoritmusai*. 2014, Typotex Kiadó.

[2] Tukora Balázs: Robotok irányítása, 2004. URL `http://tukora.com/download.php?download_file=robotok_iranyitasa.pdf`. (utolsó hozzáférés: 2019. 12. 12.).

[3] Herbert Bay – Tinne Tuytelaars – Luc Van Gool: Surf: Speeded up robust features. 2006.

[4] Kulcsár Béla: *Robottechnika I.* 2012, Typotex Kiadó.

[5] Kulcsár Béla: *Robottechnika II.* 2012, Typotex Kiadó.

[6] Marques Brownlee. URL `https://youtu.be/UIwdCN4dV6w`. (utolsó hozzáférés: 2019. 12. 12.).

[7] Conceptual figure of ransac. URL `https://d3i71xaburhd42.cloudfront.net/50d0bea94cc9ee8abc79c2015bc6cdbc1d0cbb42/2-Figure1-1.png`. (utolsó hozzáférés: 2019. 12. 12.).

[8] Peter Corke: Robotics toolbox 10.3.1.
URL `http://petercorke.com/wordpress/toolboxes/robotics-toolbox`. (utolsó hozzáférés: 2019. 12. 12.).

[9] Peter Corke: *Robotics, Vision and Control*. 2017, Springer.

[10] EMCU: Stm32cube and stm32cubemx.
URL `http://www.emcu.it/STM32Cube/STM32Cube.html`. (utolsó hozzáférés: 2019. 12. 12.).

[11] Gamepad genesis p65 for ps3/pc. URL `https://gameroom.lt/en/controllers/products/gamepad-genesis-p65-for-ps3pc-1538`. (utolsó hozzáférés: 2019. 12. 12.).

[12] Mester Gyula: *Robotika*. 2011, Typotex Kiadó.

[13] DOMAN RC Hobby: S2000md metal gear 20kg digital servo. URL `http://www.domanrchobby.com/content/?113.html`. (utolsó hozzáférés: 2019. 12. 12.).

[14] DOMAN RC Hobby: S2003md metal gear 270degree 20kg digital servo. URL http://www.domanrchobby.com/content/?120.html. (utolsó hozzáférés: 2019. 12. 12.).

[15] Zoltan Hudak: Stm32f103c8t6 board, alias blue pill.
URL https://os.mbed.com/users/hudakz/code/STM32F103C8T6_Hello/. (utolsó hozzáférés: 2019. 12. 12.).

[16] Brandon Lewien: Detailed writeup: Autonomous color seeking robot. URL https://twoandmany.wordpress.com/engineering/detailed-writeup-autonomous-color-seeking-robot/. (utolsó hozzáférés: 2019. 12. 12.).

[17] MathWorks: Feature detection and extraction. URL https://www.mathworks.com/help/vision/feature-detection-and-extraction.html. (utolsó hozzáférés: 2019. 12. 12.).

[18] Robotic arms. URL http://www.ecomorder.com/techref/robot/arms.htm. (utolsó hozzáférés: 2019. 12. 12.).

[19] St-link v2.
URL https://besplatka.ua/aws/10/38/37/96/c5fdf4483455.jpg. (utolsó hozzáférés: 2019. 12. 12.).

[20] Stm32 minimum system development board.
URL https://ae01.alicdn.com/kf/HTB1OJY8aYus3KVjSZKbq6xqkFXaN/STM32F103C8T6-ARM-STM32-Minimum-System-Development-Board-Module-For-Arduino.jpg. (utolsó hozzáférés: 2019. 12. 12.).

[21] STMicroelectronics: Stm32f103c8t6.
URL https://www.st.com/en/microcontrollers-microprocessors/stm32f103c8.html. (utolsó hozzáférés: 2019. 12. 12.).

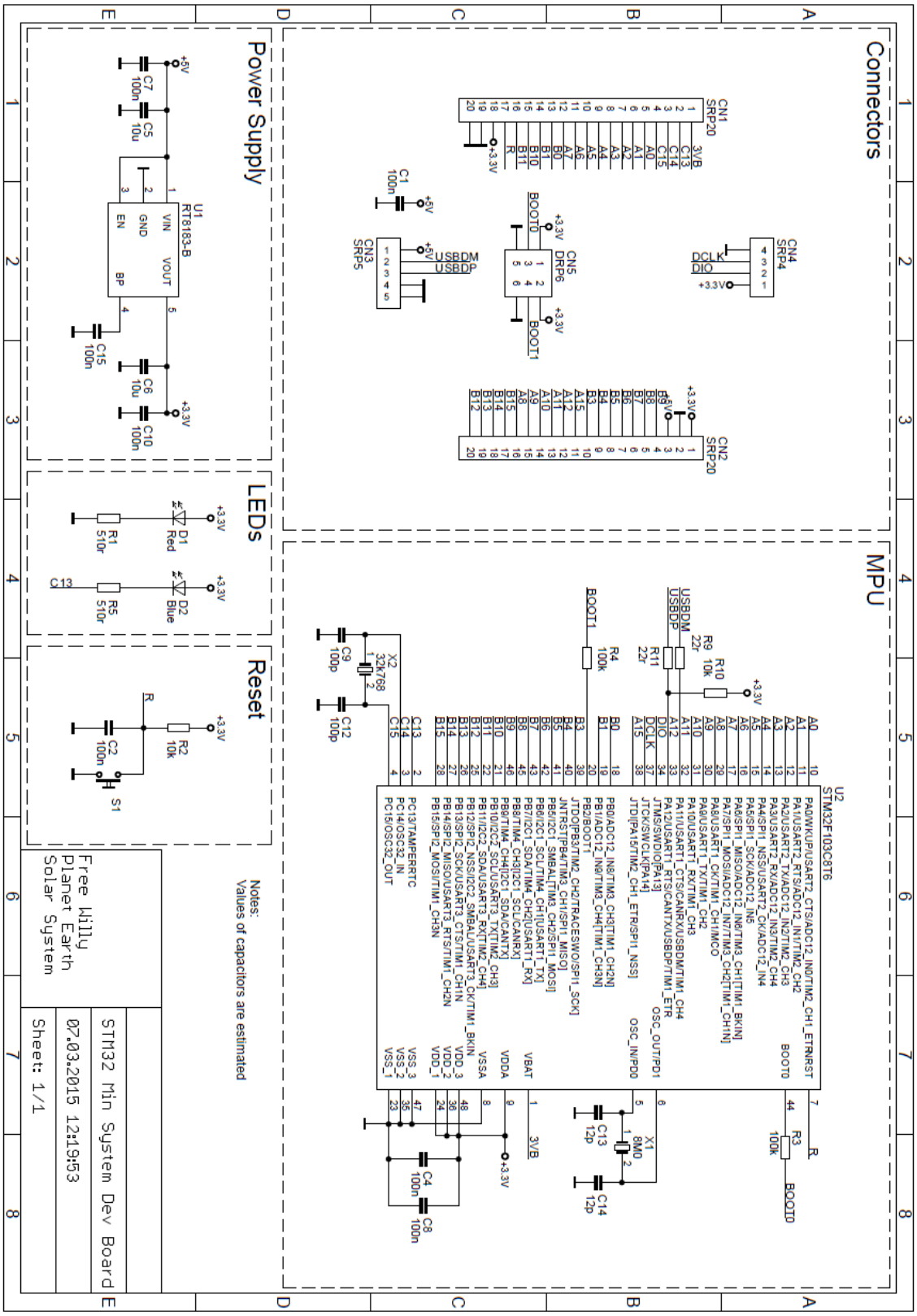
[22] SZDoit: 7 dof robot arm. URL https://www.amazon.com/Mechanical-Stainless-Manipulator-Machinery-Structure/dp/B076CLD8P1. (utolsó hozzáférés: 2019. 12. 12.).

[23] Bruce Tannenbaum: Computer vision with matlab for object detection and tracking.
URL https://youtu.be/9_6utqvsCtA. (utolsó hozzáférés: 2019. 12. 12.).

[24] Prof. Wei Zhang: Ece5463: Introduction to robotics - lecture note 8: Inverse kinematics, 2019. URL http://www2.ece.ohio-state.edu/~zhang/RoboticsClass/docs/LN8_InverseKinematics.pdf. (utolsó hozzáférés: 2019. 12. 12.).

Függelék

F.1. A fejlesztői kártya



F.1.1. ábra. Az STM32F103C6T8 minimum system developmnet board kapcsolási rajza [15]