



TANSZÉKVEZETŐ

SZAKDOLGOZAT FELADAT

Bellyei Boldizsár

szigorló villamosmérnök hallgató részére

Kétszabadságfokú robotkar ütközésmentes mozgástervezése

Autonóm robotoknak képesek kell lenniük arra, hogy önállóan tervezzék meg saját mozgásukat. A mozgással szemben több elvárás is támasztható. Legfontosabb, hogy biztonságos, ütközésmentes legyen. További szempont lehet a gyorsaság vagy a kis energiafelhasználás.

Amennyiben előre ismert, hogy a robot munkaterében hol találhatók akadályok, globális mozgástervezési módszerek alkalmazhatóak. Egyes algoritmusok a robot csuklópályáit által kifeszített, ún. konfigurációs térben tervezik meg a pályát. Az akadályok elhelyezkedése viszont a robot munkaterében ismert, azokat valamilyen módon át kell transzformálni a konfigurációs térbe.

A hallgató feladata ütközésmentes mozgást számító algoritmus fejlesztése egy kétszabadságfokú robotkarhoz.

A hallgató feladatának a következőkre kell kiterjednie:

- Ismertesse a robotkar felépítését, geometriai modelljét.
- Modellezze a robotkart Matlab Simscape segítségével.
- Mutasson be a szakirodalom alapján egy ütközésmentes mozgástervező módszert.
- Implementálja az algoritmust, szimulációban tesztelje működését.
- Vizsgálja meg, hogy a munkaterében elhelyezkedő akadályok miként képezhetők át a konfigurációs térre.
- Tervezzen pályakövetést biztosító szabályozót a robotkarhoz.
- Végezzen zártkörű teszteket.
- Értékelje az elért eredményeket, és tekintse át a továbbfejlesztési lehetőségeket.

Tanszéki konzulens: Gincsiné Szádeczky-Kardoss Emese, docens

Budapest, 2020. október 7.

/ Dr. Kiss Bálint /
docens
tanszékvezető



Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Irányítástechnika és Informatika Tanszék

Bellyei Boldizsár

KÉTSZABADSÁGFOKÚ ROBOTKAR ÜTKÖZÉSMENTES MOZGÁSTERVEZÉSE

KONZULENS

Gincsiné Dr. Szádeczky-
Kardoss Emese

Tartalomjegyzék

Összefoglaló	5
Abstract	6
1 Bevezetés	7
1.1 Felhasznált fejlesztői környezet	9
1.2 Részletes specifikáció	9
1.3 A robotkar bemutatása	9
1.3.1 Fizikai elrendezés és általános felhasználás	9
1.3.2 A robotkar geometriája	11
2 A robotkar modell elkészítése	12
2.1 A modell felépítése Simscape Multibody használatával	12
2.2 A modellhez kapcsolódó blokkok és függvények	15
3 Robotkar mozgás implementáció szimulációban	17
3.1 Forward kinematics	17
3.2 Inverse kinematics	18
3.3 Megvalósítás Matlab segítségével	21
3.3.1 Az Inverse Kinematics függvény	21
3.3.2 A Joint movements függvény	21
3.3.3 A teljes mozgás és a modell ellenőrzése	24
4 Ütközésmentes pályatervezés	26
4.1 Probabilistic Roadmap (PRM)	26
4.2 Az útvonaltérkép felépítése	27
4.2.1 Akadályok tárolása	27
4.2.2 Csúcsok és élek tárolása	28
4.2.3 Az <i>Init</i> függvény	29
4.3 Lokális útvonal keresés	30
4.3.1 A <i>trajectory</i> függvény	31
4.3.2 Egyéb függvények	32
4.4 Az algoritmus tesztelése szimulációban	32
4.4.1 Főprogram: main.m Matlab script	32
4.4.2 A módosított Simulink modell	33
4.4.3 Az ütközésmentes pályán való végig haladás	34

5 Akadályok a munkatérben.....	38
5.1 Akadályok tárolása Descartes-koordináta rendszer szerint	38
5.2 Akadályok felvétele	39
5.3 Transzformáció a konfigurációs térbe	41
6 Megvalósítás fizikai hardveren	42
6.1 Robotkar és PC kommunikáció	42
6.2 Simulink modell a fizikai robotkarhoz	43
6.3 PID szabályozás.....	45
6.4 A szabályozó hangolása.....	46
7 Akadálykerülő mozgás a valós robotkarral.....	57
7.1 Tesztelés.....	57
7.2 Eredmények analízisa.....	58
8 Összegzés.....	60
8.1 Eredmények értékelése	60
8.2 Továbbfejlesztési lehetőségek.....	60
Irodalomjegyzék.....	62
Függelék.....	63
8.3 Simscape modell felülnézeti képei	63
8.4 Második csukló nyomaték görbéi.....	64
8.5 Polinomiális pálya mátrixa.....	65
8.6 Fizikai robotkar szabályozásához kapcsolódó ábrák.....	66
8.6.1 Ugrás alapú jelek, szaturáció nélkül	66
8.6.2 Végleges szabályozó paraméterek	67
8.7 Első csukló nem 0 kezdeti szögből indítva	68
8.8 Látványos akadálykerülő pályán végig haladás	69
8.9 Akadálykerülés nem 0 kezdőszögből	71
8.10 További videók akadálykerülésről.....	71
8.11 Útmutató a szimuláció futtatásához (Matlab 2019b).....	72

HALLGATÓI NYILATKOZAT

Alulírott **Bellyei Boldizsár**, 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 hitelesített 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.

Kelt: Budapest, 2020. 12. 10.



.....
Bellyei Boldizsár

Összefoglaló

Az iparban alkalmazott autonóm robotkaroknak képesnek kell lenniük a mozgásuk önálló megtervezésére. Fontos szempont, hogy a tervezett mozgás gyorsaságra, áramfelvételre és pontosságra is optimalizálva legyen. További elvárás lehet, hogy a robotkar két pont között tudjon mozogni anélkül, hogy az előre betanított akadályoknak ütközne. A szakdolgozatom elkészítése során egy egyszerű felépítésű, DC motorokkal hajtott, két szabadságfokú robotkarhoz kapcsolódóan végeztem feladatokat.

A feladat része a robotkar fizikai paramétereinek ismertetése, illetve ezen síkban mozgó robotkar geometriájának bemutatása. A Simulink® segítségével a robotkarról egy jól használható modellt készítettem. A konfigurációs térben történő számításokhoz elengedhetetlenül szükséges direkt-, és inverz geometriai feladatokat részletesen bemutattam, illetve megadtam a nyomatékok optimalizálása szempontjából fontos mozgási egyenleteket. Ezek után részleteztem egy valószínűségi alapokon működő – a szakirodalomból vett – ütközésmentes pályatervező módszert, amely módszer központi eleme a térképkészítés, illetve az elkészült térkép bejárása egy lokális útvonalkeresési problémán belül. Az algoritmus implementálásra és tesztelésre került.

A szimuláció sikeres tesztjei után, a fizikai robotkarhoz kapcsolódó feladatok következnek: először ismertettem, hogy miképp lehet a munkatérben elhelyezkedő akadályokat megadni különböző módszerekkel, kiváltképp az iparban is elterjedt ún. betanítással. Az akadályok rögzítése miatt fontos kitérni a pontok transzformálására az XY térből a konfigurációs térbe. Ezek után a robotkar szabályozása következett, egy az iparban általánosan használt és elterjedt módszerrel (PID). A sikeres szabályozást tesztelésekkel igazoltam, a nehézségeket és a tapasztalatokat dokumentáltam. Végezetül, az akadálykerülő algoritmust és a tervezett mozgási pályát a fizikai robotkaron teszteltem, az eredményeket kiértékeltem.

Zárásként áttekintettem az elért eredményeket, illetve szót ejttem a feladathoz kapcsolódó, de a dolgozatban nem érintett fejlesztési lehetőségekről, illetve egyéb továbbfejlesztési potenciálról, valamint a feladat kiterjesztéséről.

Abstract

The autonomous robotic arms used in industry must have the ability to independently plan their motion path. Important aspects of the motion are speed, power consumption and precision, which have to be optimized for each use case. In addition, the robotic arm could have the capability to move between two discrete points, bypassing any predefined obstacles in the workspace. In my dissertation I had been working with a simple two degrees of freedom robotic arm, powered by DC drives.

Firstly, as part of the task, the real robot arm's physical parameters as well as its geometry was presented. Aided by a software called Simulink®, the robot arm was well modeled, the model was used throughout development. For better understanding of the calculations made in the configuration space, the forward and inverse kinematics had been explained, just like the equations of motion. I detailed a collision-free path design method based on a probability principle. The central element of this method is graph mapping and local route search on the built graph. The algorithm was implemented and tested.

After the simulation was successfully tested, the next steps are related to the real robot arm: first I explained how to specify obstacles located in the workspace with different methods, especially the widely used teaching. Also, I presented the method of transformation of points from XY coordinate system to the configuration space. Then I made a control system for the robot arm, using a widely spread industrial method (PID). I had been testing the control system, then documented the experiences and difficulties. Finally, I made several tests with the collision-free algorithm on the physical robot arm and evaluated the results.

Last but not least, I reviewed all the results achieved and mentioned the opportunities for improvement as well as future extensions of the project.

1 Bevezetés

Az iparban felhasznált autonóm robotkaroknak képesnek kell lenniük arra, hogy a mozgásukat önállóan tervezzék meg. Fontos, hogy ezen robotok optimalizálva legyenek az áramfelvételre, és a mozgás hosszára, gyorsaságára. További elvárás lehet, hogy a robotkar két pont között tudjon mozogni anélkül, hogy az előre betanított akadályoknak ütközne.

A két féléves munkám során egy egyszerű felépítésű, két szabadságfokú robotkarhoz kapcsolódóan végeztem feladatokat (1.2. ábra). Kezdetben a robotkar lemodellezése (ld. 2. fejezet), majd a mozgásának implementálása (ld. 3. fejezet) és ütközés mentes pálya keresése volt a feladatom (ld. 4. fejezet). Ezek után a fizikai hardveren végeztem szabályozást (ld. 6. fejezet) és az itt megvalósított szabályozót az elkészített modellben is implementáltam (ld. 7. fejezet).

Először röviden bemutatásra kerülnek a felhasznált fejlesztői környezetek (ld. 1.1. fejezet), illetve a feladat pontos és részletes specifikálása következik (ld. 1.2. fejezet), majd a robotkar felépítésének és geometriájának tömör ismertetésére kerül sor (ld. 1.3. fejezet). Ezt követően a megoldás elvi lépései és konkrét megvalósítási módszerei kerülnek bemutatásra. Végezetül az elért eredmények összefoglalása és értékelése lesz olvasható (ld. 8. fejezet).



1.1. ábra – Iparban gyakran használt FANUC SCARA robotkar [9]

KÉTSZABADSÁGFOKÚ ROBOTKAR ÜTKÖZÉSMENTES MOZGÁSTERVEZÉSE

Folyamatábra



1.2. ábra – Az elvégzett részfeladatok folyamatábrája

1.1 Felhasznált fejlesztői környezet

A modell implementálására a Matlab R2019b verzióját választottam. A felhasznált szoftverkomponensek a Simscape Multibody, a Control System Toolbox és a Robotics System Toolbox [7]. A Simscape Multibody-t a fizikai robotkar mechanikai modellezésére használtam, az így lemodellezett rendszer vizuális 3D megjelenítésére pedig a Mechanics Explorer segítségével volt lehetőségem. A különböző algoritmusok (pl. inverz geometria, trajektória tervezés) a Simulink modellben létrehozott Matlab függvények segítségével kerültek megvalósításra.

A fizikai hardveren végzett valós idejű implementációra a Quanser oktatási célra fejlesztett Quarc rendszerét használtam [8], mely egy a Windows alatt futó, ún. gyors prototípus tervező eszköz. A felhasznált szoftverkomponensek a laboratóriumi számítógépre előre telepítve és konfigurálva voltak.

1.2 Részletes specifikáció

A megvalósított feladatom egy ütközésmentes mozgást számító algoritmus fejlesztése volt kétszabadságfokú robotkarhoz. A feladat magában foglalta a robotkar felépítésének és geometriájának ismertetését, valamint egy a valóságos robotkart elvi szinten reprezentáló modell készítését. Ezek után – a szakirodalom segítségével – egy, az adott felhasználásban optimális ütközésmentes mozgástervező módszer kiválasztása és megismerése volt a dolgom. Ezt az algoritmust implementálnom kellett, valamint a működését a szimuláció segítségével tesztelnem. A munkatérben elhelyezkedő akadályok megadása és konfigurációs térbe való transzformálása után, egy pályakövetést biztosító szabályozó tervezése volt a feladatom a valós rendszeren. Végezetül az ütközésmentes mozgás tesztelésére került sor a fizikai robotkaron, valós akadályokkal. Az eredményeket kiértékeltem és az esetleges továbbfejlesztési lehetőségeket megemlítettem.

1.3 A robotkar bemutatása

1.3.1 Fizikai elrendezés és általános felhasználás

A munkám során felhasznált és modellezett 2 szabadságfokú robotkar a tanszéki laborban megtalálható, melynek egységei alumínium gépépítő profilok, ezek egymással párhuzamosak, függőleges tengelyük mentén elforgásra képesek (ld. 1.3. ábra). A robotkar csuklóinak beavatkozó szervei DC motorok, az aktuális pozíció pedig

inkrementális adó segítségével mérhető. A szögelfordulás mérése referenciázáshoz kötött minden indítást megelőzően, ugyanis a felszerelt szenzorok nem abszolút enkóderek.

A robotkar teljes (360° -os) körbefordulásra nem képes, az alaphelyzethez képest $\pm 175^\circ$ -os elfordulásra van lehetőség. Ez a végállás nem hardveresen van beiktatva, hanem a fejlesztés során szoftveresen került megvalósításra, elkerülve a kábelek felcsavarodását a robotra mozgás közben.

Hasonló – az iparban gyakran használt – robotok az ún. *SCARA* (*Selective Compliance Assembly Robot Arm*) robotkarok (1.1. ábra), melyek szintén két csuklót tartalmaznak, így XY síkban képesek mozogni. A különbség annyi, hogy a végberendezés szerszáma (jellemzően egy csavarhúzó vagy fúró fej) függőlegesen (Z irányban) képes elmozdulni, így pl. egy furat fölé mozgatva a szerszámot, egy csavart el tud helyezni. Ezen alkalmazások sokszor fordulnak elő gyártócellákon belül, összeszereléseknél. Ebből látható, hogy az egyszerűbb, síkban mozgó robotkarok felhasználási területe még mindig jelentős a jellemzően rakodásra és termékmozgatásra használt 6 szabadságfokú karok mellett.



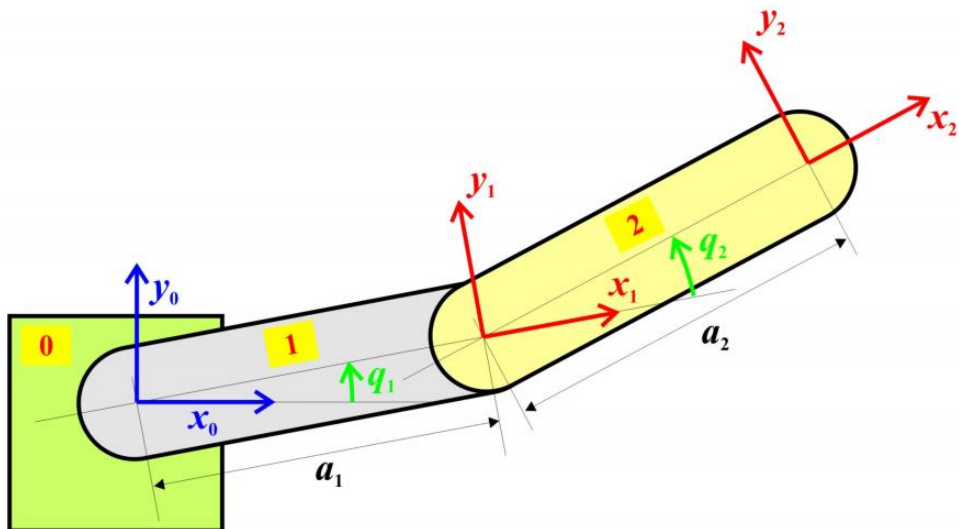
1.3. ábra – A valós robotkar

1.3.2 A robotkar geometriája

A robotkar az alábbi szegmensekből áll, melyek a lenti (1.4. ábra) ábrán számozva is láthatóak [1]:

0. a fix pozíciójú bázis,
1. az a_1 hosszú merev egység
2. az a_2 hosszú merev egység

A bázishoz rendelt koordináta rendszer állandó, nem mozog. Ez a koordináta-rendszer a szimulációban a bázis elem mértani középpontjába került. Ezen kívül még két kitüntetett koordináta rendszer felvétele szükséges, ezek a 2 mozgó egység végpontjai. Az 1. egység forgástengelye egybeesik a bázis koordináta-rendszer z (függőleges) tengelyével. Az első egység végpontjára illesztett koordináta-rendszer z (függőleges) tengelye pedig egybeesik a 2. egység forgástengelyével. A végberendezés koordináta-rendszere a 2. egység végére illesztett koordináta-rendszert jelenti. A megoldás szempontjából különös fontossággal bírnak az ábrán q_1 és q_2 -vel jelölt szögek, melyek az egységek orientációját mutatják a forgástengelyüknél elhelyezett koordináta-rendszerekhez képest. A tényleges robotkar esetén a csuklók szögeinek (q_1 és q_2) mérésére van lehetőség inkrementális adók segítségével, ezért aktuális pozíció meghatározásához is ezeket az adatokat használom fel a szimulációban.



1.4. ábra – A robotkar felülnézeti sematikus ábrája [1]

Könnyen belátható, hogy a két szegmens mozgása egymásra ráhatással van, hiszen az egyik szegmentet mozgatva a másik szegmens tehetetlenségénél fogva szintén elmozdul. Ezt a problémát megfelelő szabályozással lehet megoldani (lásd 6.4. fejezet).

2 A robotkar modell elkészítése

A fizikai elrendezés lemodellezése több okból kifolyólag is előnyös: a valós hardver nélkül lehet vizsgálni a robot mozgását, illetve az elkészített algoritmusok könnyen és egyszerűen ellenőrizhetők, az azonos paraméterű futtatások közt eltérés nincs, külső tényezők nem befolyásolják a működést (a szimulációban az inhomogenitások és külső zavarok az egyszerűség kedvéért elhagyhatóak), a rosszul paraméterezett robotkar a környezetében kárt nem tud tenni. Ezen előnyöket kihasználva, modell készítés mellett döntöttem, amely a fejlesztés minden szakaszát jelentősen megkönnyítette a későbbiekben. Az alábbi fejezetben a fizikai elrendezés modellje és a mozgási modell elkészítésének részletes lépései kerülnek bemutatásra.

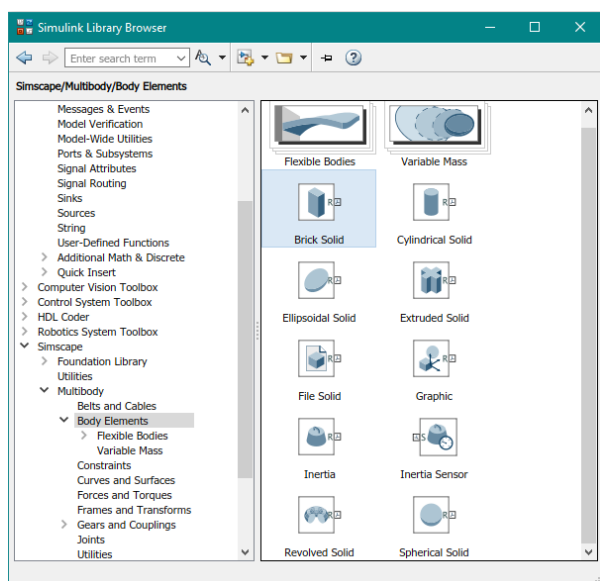
2.1 A modell felépítése Simscape Multibody használatával

A Matlab Simscape Multibody jól használható különböző mechanikai rendszerek modellezésére, a feladat mindössze a Simulink projektben, a megfelelő blokkok összekötése és felparaméterezése (2.6. ábra).

A robotkar három négyzetes hasábbal könnyen reprezentálható, ahol az egyik test a korábban ismertetett fix pozíciójú bázis (*Base*), a másik kettő pedig a robotkar két mozgó szegmense (*Arm1* & *Arm2*). Az egységek összekapcsolása rotációs csuklókkal történik, az első szegmens és a bázis, valamint az első és második szegmens között.

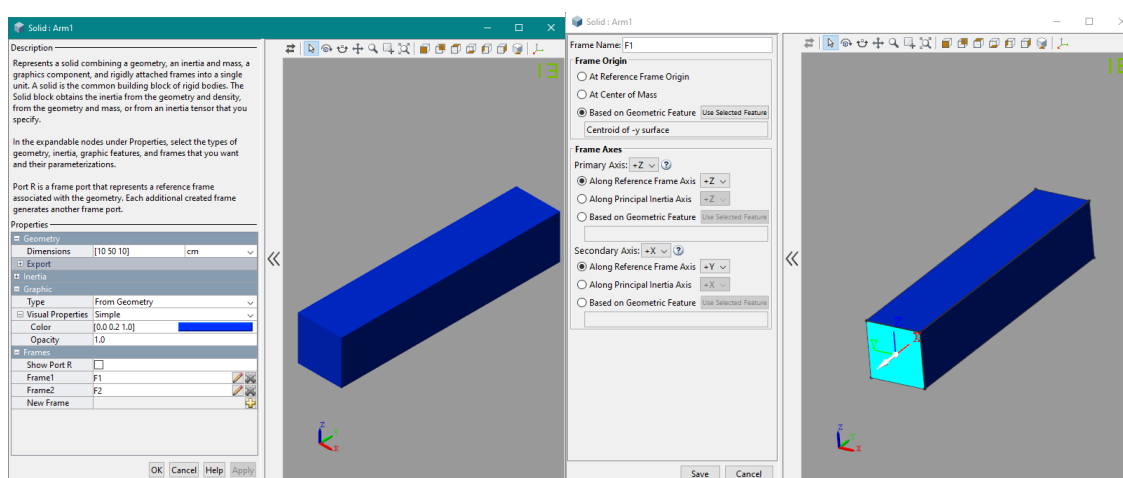
A definiált Simulink projektben először létrehoztam egy alrendszer, melynek bemenetei később kerülnek ismertetésre.

Ebben az alrendszerbe a korábban leírt testeket a Simulink Library Browser-ben található *Simscape/Multibody/Body Elements/Brick Solid* elemek közül szúrtam be (2.1. ábra).



2.1. ábra – Merev test hozzáadása

A létrehozott blokkban lehetőség van a test tulajdonságainak módosítására (lásd 2.2. ábra), inerciarendszerek megadására stb. A testek színei tetszés szerint variálhatóak, a vizualizálás céljából célszerűen eltérő színű szegmenseket választottam (2.3. ábra). A robotkar geometriája c. fejezetben (1.3.2. fejezet) bemutatott koordináta rendszerek (Frame) hozzáadása a testekhez egyszerűen megtehető a kívánt grafikus felületre kattintva. Itt beállítható a tengelyek pozitív-negatív irányítása is, fontos, hogy az egybeeső tengelyek azonos alaphelyzetűek legyenek.



2.2. ábra – Alakzat tulajdonságok beállítása

<i>Beállított paraméterek</i>	<i>Bázis (Base)</i>	<i>Első szegmens (Arm1)</i>	<i>Második szegmens (Arm2)</i>
<i>Szélesség [cm]</i>	20	10	10
<i>Hossz [cm]</i>	20	50	40
<i>Magasság [cm]</i>	10	10	10
<i>Szín</i>	zöld	kék	sárga

2.3. ábra – Beállított test paraméterek

A létrehozott testek elfordulását a *Revolute Joint* blokkok szolgáltatják. Itt a két koordinátarendszer bekötésére van szükség, valamint a mozgás mértékét adó jelre. Az elfordulás Z tengely körül történik. A csuklózó az elfordulás szöge radiánban, itt a későbbiekben (ld. 3.3.2. fejezet) indokolásra kerülő okok miatt ezen változók 1. és 2. rendű deriváltjai is bekötésre kerülnek egy konverteren keresztül.

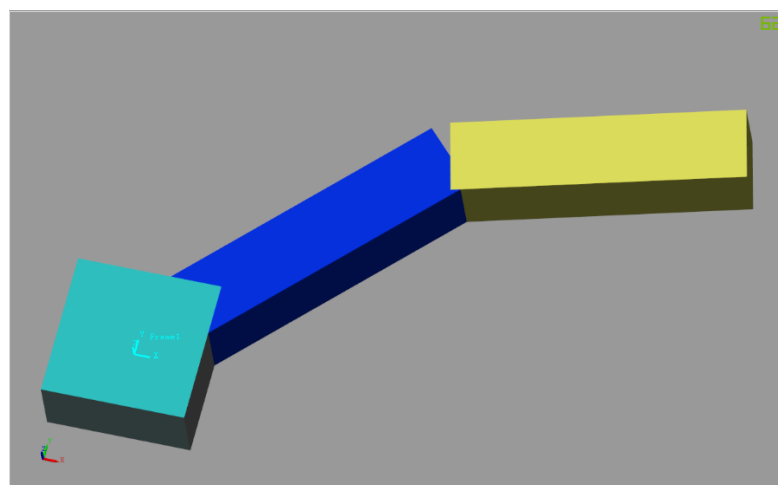
A szimulációhoz szükség van még egy világ-koordináta-rendszerre (*World Frame*), amihez a bázishoz rendelt koordináta-rendszert rögzítettem. Ezen kívül szükséges egy *Solver Configuration*, illetve egy *Mechanism Configuration* blokk. Utóbbiban a gravitáció nagysága és iránya az alapértelmezett állásban van, Z tengely mentén negatív irányban $g = 9.80665$.

Szemléltetés és hibaelhárítás céljából a következő jelek kerültek megjelenítésre:

1.) A két csukló érzékelés menüpontjában a beavatkozók nyomatéka, melyek *PS-Simulink* konverterekkel szkópon lettek megjelenítve. (*Revolute Joint -> Sensing-> Actuator Torque*)

2.) A legtávolabbi szegmens végberendezésének (*Arm2->Frame2*) kivezetett jelét egy *Transform Sensor* blokk segítségével vizsgálom. A blokkba bekötésre került a világ-koordináta-rendszer (ami a bázis koordináta-rendszere is egyben), kimenetei pedig a végberendezés koordináta-rendszer x,y,z értékei a világkoordináta-rendszerbe transzformálva. Ezáltal az x,y koordinátákat megjelenítve vizsgálható, hogy a végberendezés valóban elérte-e a kívánt koordinátákat a konfigurációs térben.

Az így felépített modell az alábbi képen látható (2.4. ábra). A $q_1 = 0 (+2k\pi)$ azt jelenti, hogy az *Arm1* koordináta-rendszerének X_1 tengelye előjel helyesen illeszkedik a bázis X_0 tengelyére. A $q_2 = 0 (+2k\pi)$ pedig azt, hogy az *Arm2* koordináta-rendszerének X_2 tengelye illeszkedik az *Arm1* koordináta-rendszerének X_1 tengelyére, szintén előjel helyesen. A modell a *Mechanics Explorers* ablakban megtekinthető. A világ-koordináta-rendszer orientációja a bal alsó sarokban jelezve van a szimuláció során. A modell felülnézeti képei a függelékben megtalálhatóak (ld. 8.3. fejezet).



2.4. ábra – A modell képe 3D nézetben ($q_1 = 45^\circ$ és $q_2 = -30^\circ$)

2.2 A modellhez kapcsolódó blokkok és függvények

A csuklóváltozókat egy Matlab függvény (*Joint Movements*) szolgáltatja, melynek bemenetei az alrendszer bemenetei is egyben. Az alrendszert egy külső 1 bites érték engedélyezi, mely a későbbiekben ismertetésre kerül.

Az alrendszer bemeneti:

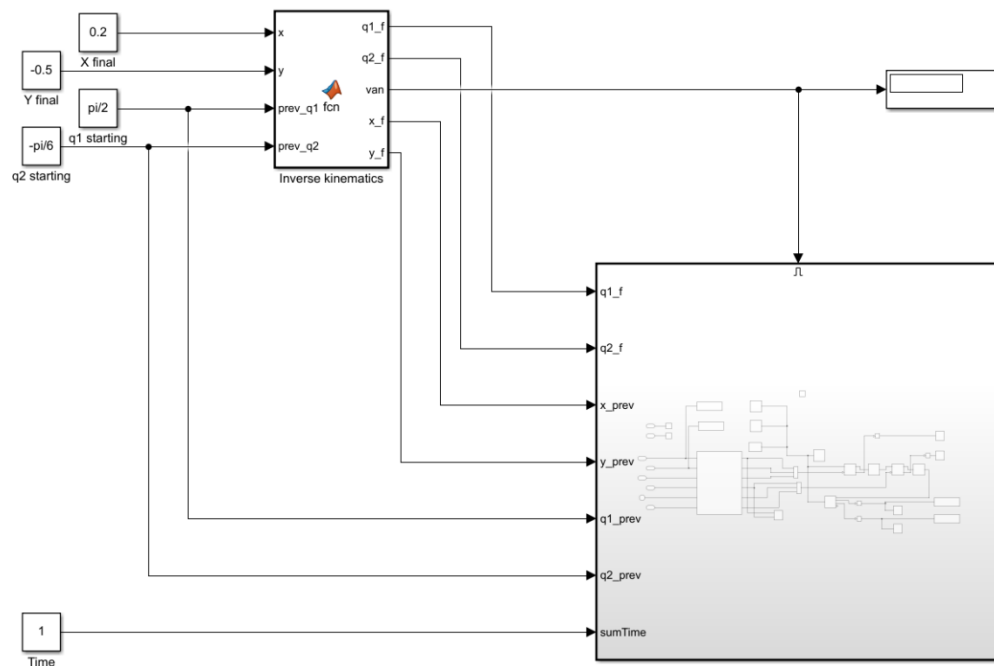
- **q1_f és q2_f:** Azok a csuklőszögek, amelyek a kívánt vég-koordinátákhoz tartoznak.
- **q1_prev és q2_prev:** A csuklőszögek, melyek a kezdeti (kiindulási) koordinátákhoz tartoznak.
- **sumTime:** A szimuláció hossza [s]
- **x_prev és y_prev:** A kiinduló koordináták (hibakeresés céljából)

Az alrendszerben található Matlab függvény a kiindulási-, és vég-koordinátákhoz tartozó csuklóváltozók, valamint a szimuláció futási ideje és aktuális időpillanata (melyet a *Clock* blokk szolgáltat) alapján kiszámítja a szükséges beavatkozó jeleket és deriváltjaikat a későbbiekben ismertetett módon.

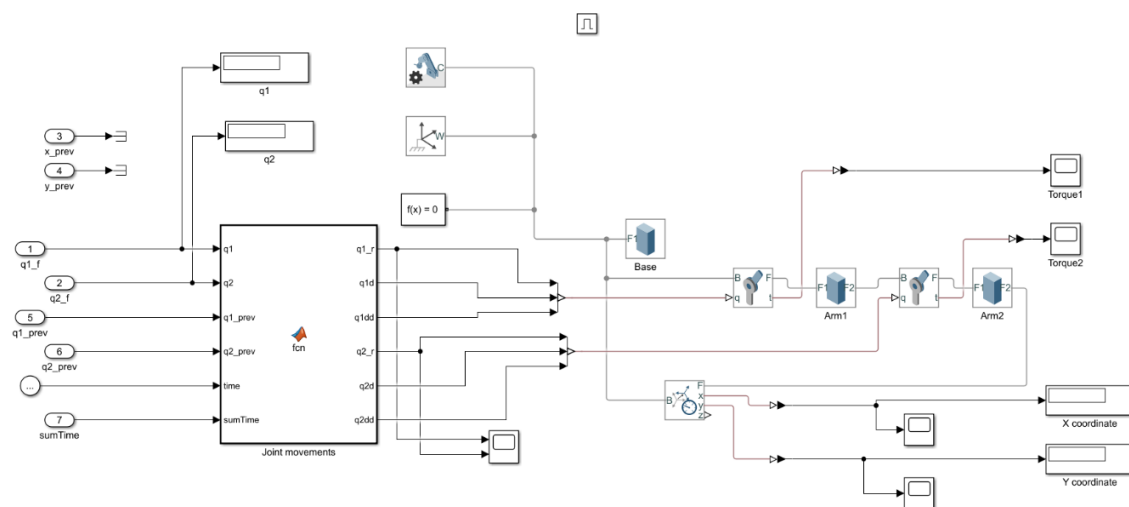
Az alrendszer felépítése a korábban ismertetett (ld. 2.1 alfejezet) blokkokból lentebb látható (2.6. ábra). Az alrendszer a *robotArm_basic* nevű Simulink modellben található. Itt négy darab konstans állítására van mód (2.5. ábra):

- **X final és Y final:** Az elérni kívánt pont koordinátái (a bázis koordináta-rendszerében)
- **q1 starting és q2 starting:** A mozgás megkezdése előtt „mért” ofszet csukló szögek
- **Time:** A szimuláció hossza [s]

Az elérni kívánt koordináták és a kezdeti szögállás segítségével az *Inverse Kinematics* függvény megnézi, hogy a koordináta elérhető-e a robot számára, amennyiben ez teljesül azt a „van” bit igaz értékre állításával jelzi és így engedélyezi az alrendszert. Ezen kívül, a majd későbbiekben ismertetett módon, kiszámítja az ezen koordinátákhoz tartozó csukló állásokat (ld. 3.2. fejezet). Az elmozdulás előtti ofszet állásból – szemléltetés céljából – meghatározásra kerülnek a kiinduló koordináták. Ezen értékek, valamint a szimuláció hossza bevezetésre kerülnek az alrendszer blokkjába.



2.5. ábra – A teljes Simulink projekt



2.6. ábra – Az alrendszerben lévő robotkar modell

3 Robotkar mozgás implementáció szimulációban

Az eredeti fizikai rendszerben a robotkar egységeinek egymáshoz viszonyított szögei mérésére van lehetőség inkrementális adók segítségével, azonban a végberendezés kívánt pozíciója az embernek kézenfekvőbb (x,y) koordinátás alakban adható meg, amiből a szögek, azaz a csuklók beavatkozó jelei származtathatóak.

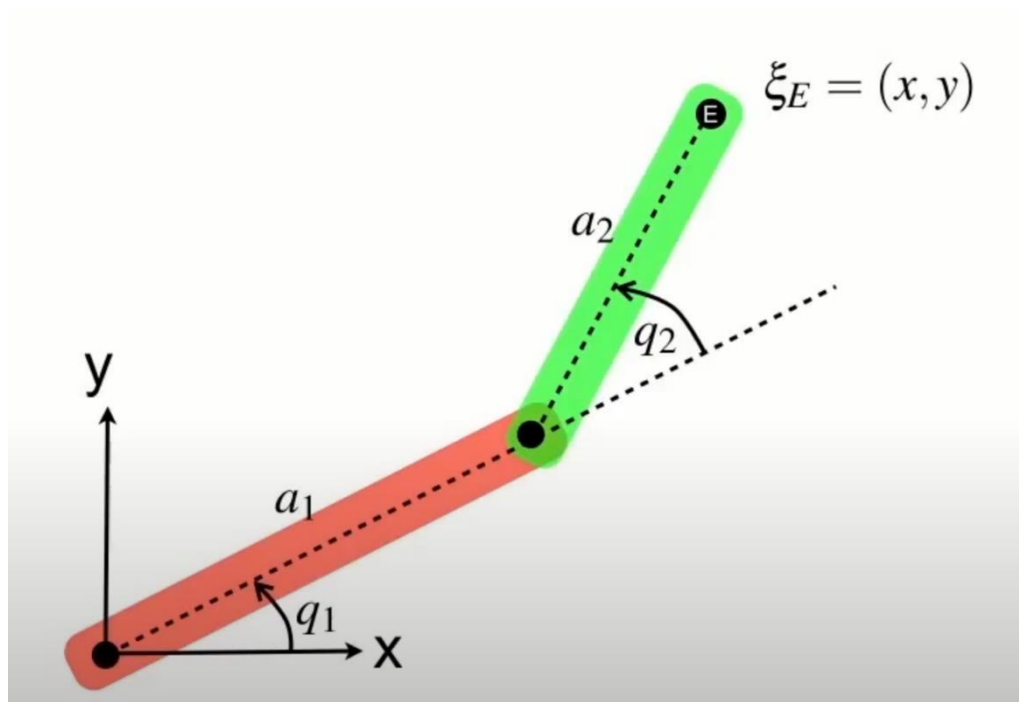
Tehát két különböző geometriai feladat megoldása szükséges:

- 1.) Forward kinematics: A q_1, q_2 szögekből számolni (x,y) koordinátákat (direkt geometriai feladat)
- 2.) Inverse kinematics: Az (x,y) koordinátákból számolni q_1, q_2 szögeket (inverz geometriai feladat)

A direkt geometriai feladathoz tartozó számításokat az ábra (3.1. ábra) alapján magam vezettem le. Az inverz geometriai képleteket Prof. Peter Cork videójából vettem át. [4]

3.1 Forward kinematics

Amennyiben a q_1, q_2 csuklószögek rendelkezésre állnak és a robotkar paraméterei ismertek, úgy egyértelműen megadható pontosan egy (x,y) koordináta az első szegmens forgástengelyéhez rögzített koordinátarendszerben, mely a robotkar második szegmensének végpontját (E) írja le.



3.1. ábra – Robotkar sematikus ábrája [4]

Jelölje (3.1. ábra) az első szegmens hosszát a_1 , a másodikét pedig a_2 , az első szegmens X tengellyel bezárt szögét q_1 , a második szegmens az elsőnek a hosszmenti tengelyével bezárt szögét pedig q_2 .

Ekkor a berajzolható derékszögű háromszögekből a második szegmens forgástengely-pontjának koordinátája:

$$x_1 = a_1 * \cos q_1 \quad y_1 = a_1 * \sin q_1$$

Hasonlóan számítható a végberendezés (E pont) koordinátája, a 2. szegmens forgástengelyének koordináta-rendszeréhez viszonyítva, ahol a szögfüggvények argumentumaiba a q_1 -hez a q_2 szöget is hozzá kell adni, előjel helyesen:

$$x_2 = a_2 * \cos(q_1 + q_2) \quad y_2 = a_2 * \sin(q_1 + q_2)$$

Tehát a végpont (E pont) koordinátái a szögek és a szegmens hosszok paramétereivel kifejezve:

$$x = a_1 * \cos q_1 + a_2 * \cos(q_1 + q_2) \quad y = a_1 * \sin q_1 + a_2 * \sin(q_1 + q_2)$$

3.2 Inverse kinematics

Belátható, hogy a robotkar végberendezése a bázis X_0, Y_0 koordinátarendszerében két kör által határolt belső pontokat tudja elérni (a $\pm 175^\circ$ -os korlátozást nem tekintve). Ha az első szegmens hossza a_1 , a másodiké pedig a_2 , akkor a két kör sugara a bázis koordinátarendszer origójától számítva:

$$R_{belső} = a_1 - a_2 \quad R_{külső} = a_1 + a_2$$

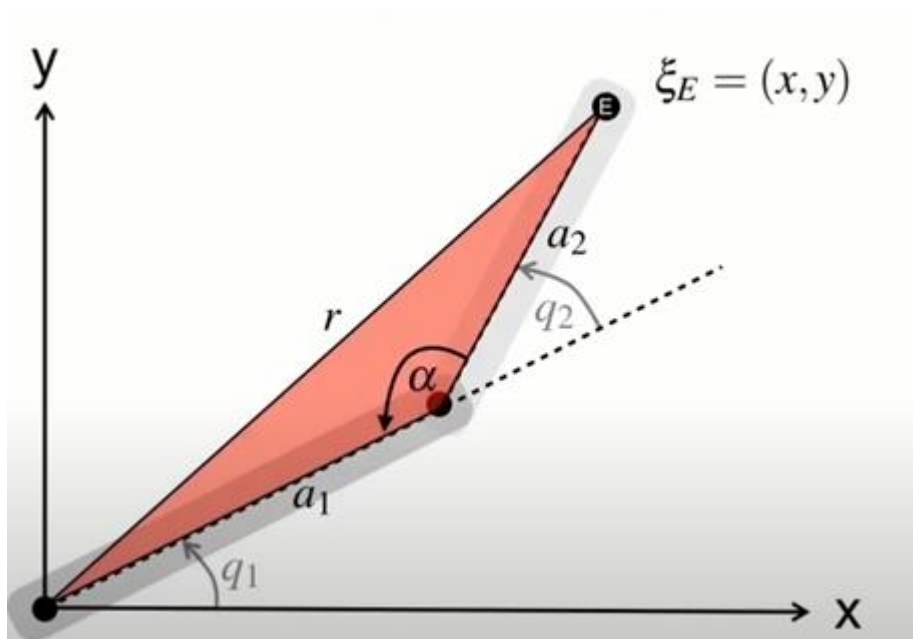
A körön lévő pontokhoz pontosan egy (q_1, q_2) konfiguráció tartozik: mikor teljesen „nyitva” (azaz $q_2 = 0^\circ$), illetve mikor teljesen „zárva” (azaz $q_2 = \pm 180^\circ$) van a második szegmens. Minden másik – a tartományban lévő – ponthoz két ilyen szögpár tartozik: egy negatív és egy pozitív q_2 szögű.

1.) Pozitív q_2 szögű szögpár [4]:

A lentebbi rajzon (3.2. ábra) bejelölt szakaszok és szögek alapján a következő egyenletek írhatóak fel (Pitagorasz-tétel és Koszinusz-tétel):

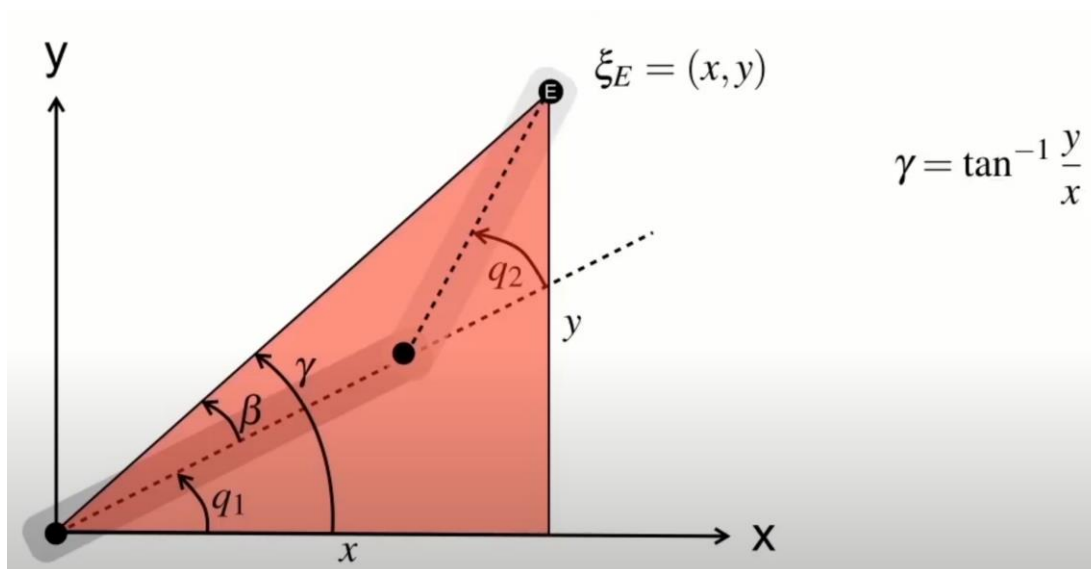
$$r^2 = x^2 + y^2 = a_1^2 + a_2^2 - 2a_1a_2 \cos \alpha \quad \cos \alpha = \frac{a_1^2 + a_2^2 - r^2}{2a_1a_2} = \frac{a_1^2 + a_2^2 - x^2 - y^2}{2a_1a_2}$$

$$\cos q_2 = -\cos \alpha \quad \Rightarrow \quad q_2 = \cos^{-1} \frac{x^2 + y^2 - a_1^2 - a_2^2}{2a_1a_2}$$



3.2. ábra – Pozitív q_2 számítása [4]

Az előbbi elrendezésben egy derékszögű háromszöget rajzolva, az alábbi szögek érdekesek:



3.3. ábra – q_1 számítása pozitív q_2 esetén [4]

A fentebb látható (3.3. ábra) háromszögek segítségével az alábbi egyenletek írhatóak fel:

$$\beta = \tan^{-1} \frac{a_2 \sin q_2}{a_1 + a_2 \cos q_2} \quad \gamma = \tan^{-1} \frac{y}{x} \quad q_1 = \gamma - \beta$$

Behelyettesítve megkapható a keresett q_1 szög:

$$q_1 = \tan^{-1} \frac{y}{x} - \tan^{-1} \frac{a_2 \sin q_2}{a_1 + a_2 \cos q_2}$$

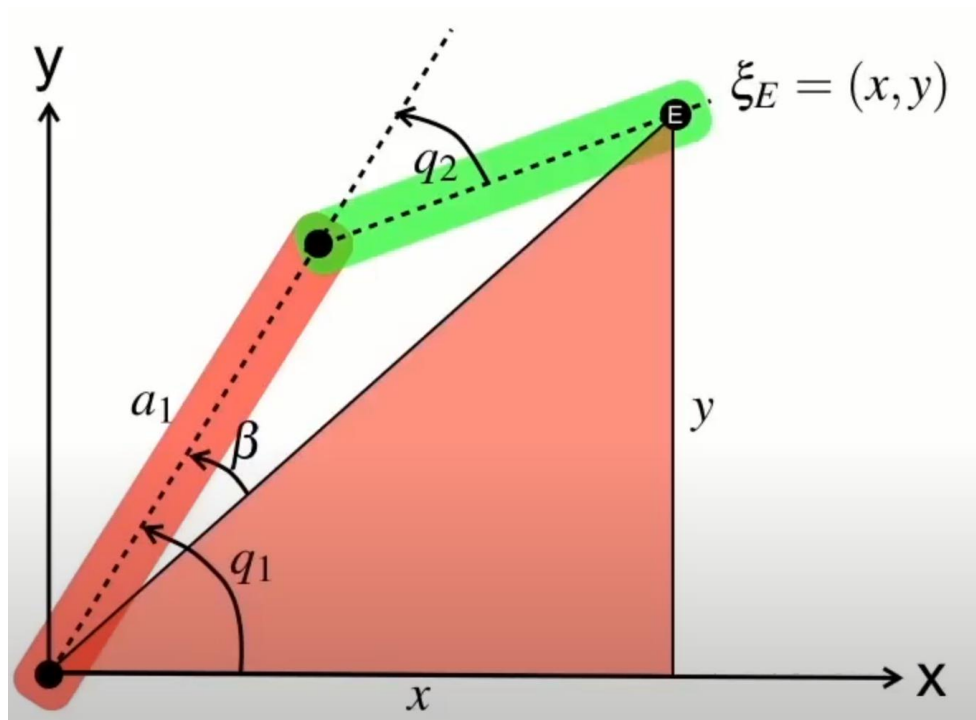
2.) Negatív q_2 szögű szögpár [4]:

Ebben az esetben a q_2 számításához használt háromszög „lefele néz”, q_2 negatív, ezért a korábbiakhoz hasonló módon:

$$q_2 = -\cos^{-1} \frac{x^2 + y^2 - a_1^2 - a_2^2}{2a_1a_2}$$

A q_1 számításához használt háromszög ebben az esetben a lentebb (3.4. ábra) látható. Ekkor $q_1 = \gamma + \beta$, ezáltal a végeredmény szintén egy előjelben tér el:

$$q_1 = \tan^{-1} \frac{y}{x} + \tan^{-1} \frac{a_2 \sin q_2}{a_1 + a_2 \cos q_2}$$



3.4. ábra – q_1 számítása negatív q_2 esetén [4]

Látható, hogy a Forward kinematics c. (3.1. számú) fejezetben látottakkal ellentétben, q_1 szöge q_2 -ből származtatandó. A két esetben q_2 szögek egymás ellentettjei, a q_1 szögek azonban nem csak előjelükben különböznek.

3.3 Megvalósítás Matlab segítségével

A korábban bemutatott (ld. 2.2. fejezet) Simulink modellben lévő függvények számításai Matlab script-ben vannak implementálva.

3.3.1 Az Inverse Kinematics függvény

Bemenetek:

- A robotkar kezdeti szögei ($prev_q_1$, $prev_q_2$);
- A mozgás végeztével a végberendezés elvárt koordinátái (x, y)

Kimenetek:

- A mozgás végeztével elvárt szögek (q_{1_f} , q_{2_f})
- A robotkar kezdeti koordinátái (x_f , y_f)
- Engedélyező kimenet (van)

Az engedélyező bemenet akkor igaz értékű ($van = true$), ha a cél-koordinátákat a robotkar képes elérni. Ez akkor tud megtörténni, ha azok a korábban ismertetett (ld. 3.2. fejezet) körök által határolt területbe esnek:

$$\sqrt{x^2 + y^2} \leq a_1 + a_2 \quad \text{ÉS} \quad \sqrt{x^2 + y^2} \geq a_1 - a_2$$

Ezen kívül itt már azt is figyelembe kell venni, hogy a kar nem éri el valamelyik végállást, azaz q_{1_f} és q_{2_f} sem lehet a $\pm 175^\circ$ -os tartományon kívül.

A kimeneti értékek az Inverse kinematics fejezetben ismertetett képletek szerint kerülnek kiszámításra. A két megoldás közül a rövidebb úttal rendelkező kerül kiválasztásra.

3.3.2 A Joint movements függvény

Bemenetek:

- q_1 és q_2 : A végpozícióhoz tartozó szögek
- q_{1_prev} és q_{2_prev} : A kezdeti állapothoz tartozó szögek
- $time$: szimuláció kezdete óta eltelt idő, *Clock* blokk szolgáltatja
- $sumTime$: A szimuláció hossza időben

Kimenetek:

- q_{1_r} , q_{1d} , q_{1dd} : az aktuális q_1 csuklóváltozó és annak 1. és 2. deriváltjai
- q_{2_r} , q_{2d} , q_{2dd} : az aktuális q_2 csuklóváltozó és annak 1. és 2. deriváltjai

A függvény felel a szimulációban a robotkar tényleges mozgásáért. A mozgáshoz, a szimuláció során (adott kvantálás mellett) az összes időpillanatban a kezdeti és a végleges

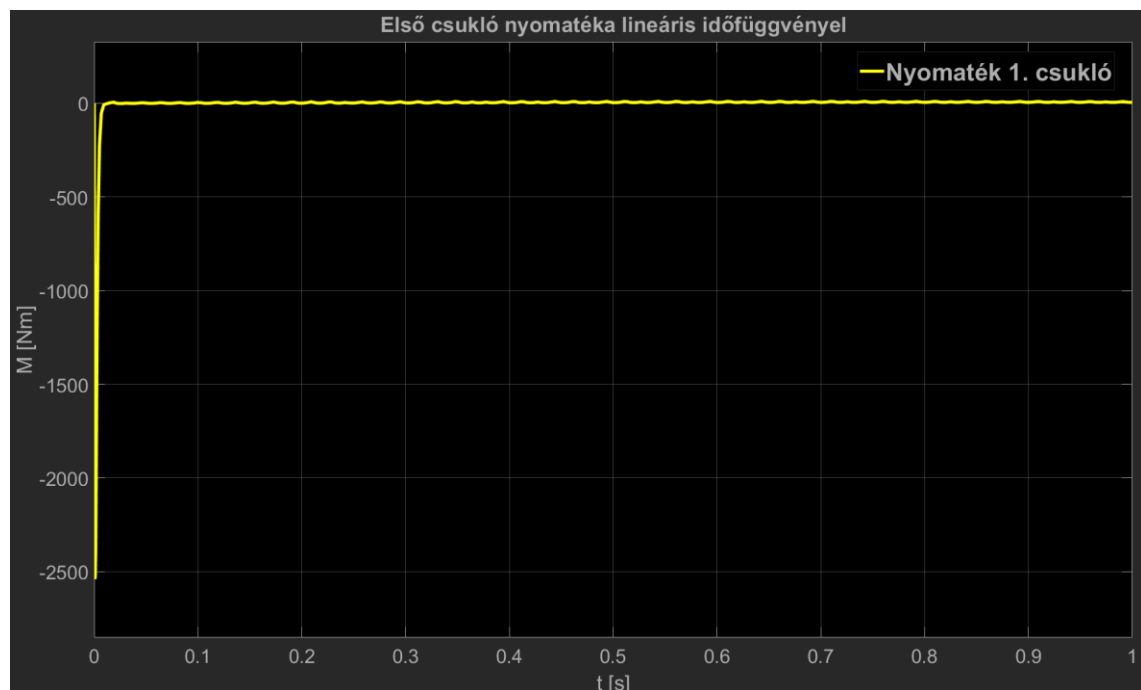
szögállás közti számított szögek kerülnek a csuklók bemenetére, ami összefüggő mozgásnak látszik kellően kis időalap mellett.

Első körben a csuklóváltozók időfüggvényeit lineáris függvénynek választottam, azonban ebben az esetben a lentebb (3.5. ábra) látható csuklónyomatékok voltak mérhetőek. A szkópon jól látszik, hogy indításkor nagyon nagy nyomaték szükséges (tüske), majd utána szinte semmi.

Lineáris csuklóváltozók esetén a q_r aktuális csuklóváltozó függvénye:

$q_{unit} = \frac{q - q_{offset}}{T}$, ahol T a szimuláció hossza, q a végső szögérték, q_{unit} az egy időpillanathoz tartozó szögelfordulás felbontása, ami függ a szimuláció hosszától.

$q_r = q_{offset} + q_{unit} * time$, ahol $time$ a szimuláció kezdete óta eltelt idő.



3.5. ábra – Első csukló nyomaték görbe lineáris időfüggvényel

Ahhoz, hogy indításkor és megálláskor ezek a nyomatékok ne legyenek ilyen kiugróan nagyok magasabb fokszámú polinomokat kell használni a csuklóváltozók számításához. Elő lehet írni azt a feltételt, hogy induláskor és megálláskor is 0 legyen a gyorsulás. A kezdeti és a mozgás végi szöghelyzet felhasználásával ötödfokú polinomot lehet használni. [2]

A csuklóváltozók időfüggvénye (ezen időfüggvények mindkét csuklóhoz egyformák) [2]:

$$q(time) = a_0 time^5 + a_1 time^4 + a_2 time^3 + a_3 time^2 + a_4 time + a_5$$

Ennek 1. és 2. deriváltjai:

$$\dot{q}(time) = 5a_0time^4 + 4a_1time^3 + 3a_2time^2 + 2a_3time + a_4$$

$$\ddot{q}(time) = 20a_0time^3 + 12a_1time^2 + 6a_2time + 2a_3$$

Kezdeti (time = 0) és végfeltételek (time = T): [2]

- time = 0 esetén $q(0) = q_{offset}$ és $\dot{q}(0) = \ddot{q}(0) = 0$
- time = T esetén $q(T) = q$ és $\dot{q}(T) = \ddot{q}(T) = 0$

Ezekből felírható egy 6 ismeretlenes, 6 egyenletű lineáris egyenletrendszer, amely mátrixos formában [2] :

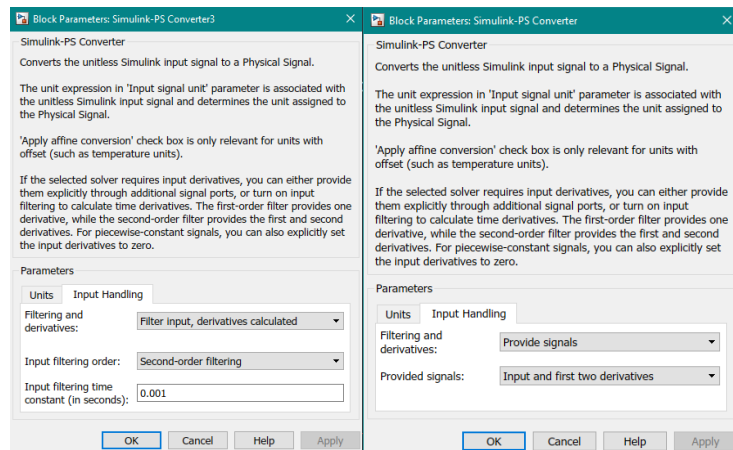
$\mathbf{M} \cdot \mathbf{A} = \mathbf{Q}$, ahol

$$\mathbf{M} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 \\ T^5 & T^4 & T^3 & T^2 & T & 1 \\ 5T^4 & 4T^3 & 3T^2 & 2T & 1 & 0 \\ 20T^3 & 12T^2 & 6T & 2 & 0 & 0 \end{pmatrix} \quad \mathbf{A} = \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{pmatrix} \quad \mathbf{Q} = \begin{pmatrix} q_{offset} \\ 0 \\ 0 \\ q \\ 0 \\ 0 \end{pmatrix}$$

Az \mathbf{A} oszlopvektor elemeit keressük, \mathbf{M} mátrix és \mathbf{Q} oszlopvektor ismert, ezek alapján:

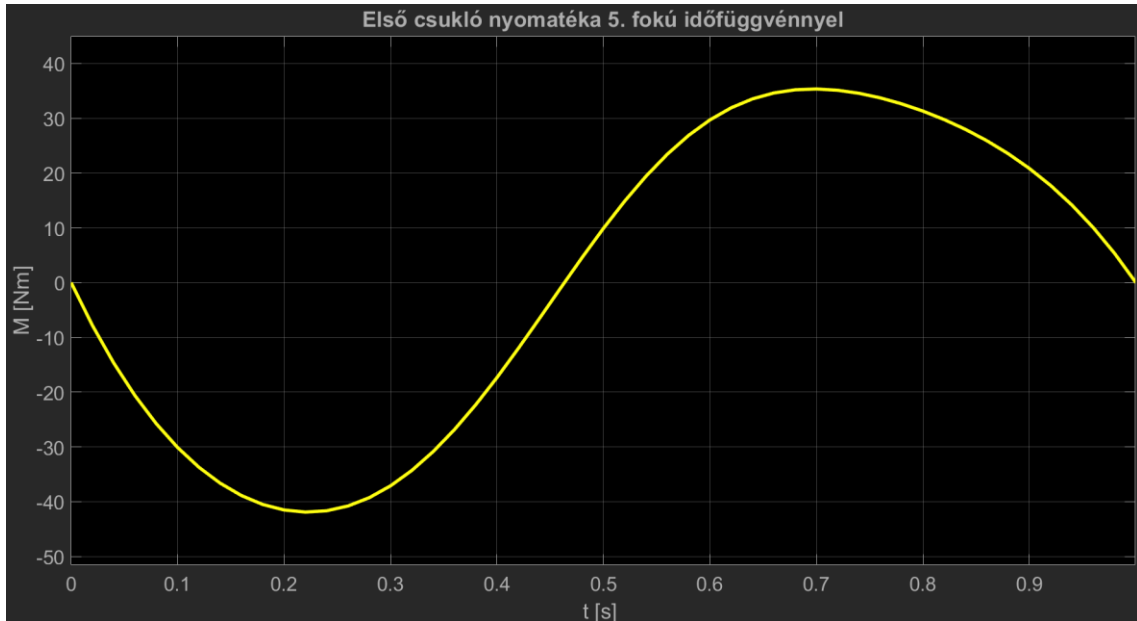
$$\mathbf{A} = \mathbf{M}^{-1} \cdot \mathbf{Q}$$

A *Joint Variables* függvényben ezen mátrixok feltöltése és ezek alapján \mathbf{A} vektor számítása történik. A szimuláció során, mindig az adott pillanatnak megfelelő csuklózváltozók és azok deriváltjai kerülnek a fent ismerttetett képletekkel kiszámításra, ahol *time* a függvény számára szolgáltatott, a szimuláció kezdete óta eltelt idő. A csuklóknak a deriváltak bevezetésére a *Simulink-PS Converter* blokkban az *Input Handling* fülön az alábbi módosításokat kell végezni (3.6. ábra):



3.6. ábra – A deriváltak automatikus számítása (b) és azok manuális bekötése (j)

Ezek után a szimulációt futtatva a szkópon (3.7. ábra) látható, hogy a nyomaték a várakozásoknak megfelelően alakult: a változás egyenletes, ugrások nincsenek benne.



3.7. ábra – Első csukló nyomaték görbe 5. fokú időfüggvénnyel

A második csuklóhoz tartozó nyomaték görbék megtalálhatóak a függelékben (ld. 8.4. fejezet).

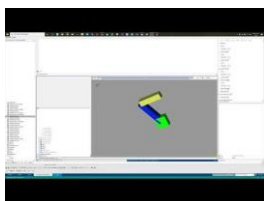
3.3.3 A teljes mozgás és a modell ellenőrzése

A modell ezek után teljes és futtatható. A Simulink modellben *q1 starting* és *q2 starting* konstansok beadásával lehet a kezdeti szöghelyzetet beállítani (ahonnan a mozgás indulni fog). Ez a szöghelyzet radiánban értendő. Ugyanitt az *X final* és *Y final* konstansokkal lehet megadni azt a koordinátát, ahova a végberendezés a mozgás végén kerüljön. A *Time* a szimuláció hosszát jelenti másodpercben, szintén állítható.

Amennyiben a megadott x és y koordináta elérhető a kar számára, a *Mechanics Explorer* ablakban végig lehet követni a robotkar mozgását. Az alrendszerben elhelyezett megjelenítőkön pedig ellenőrizhető, hogy a megfelelő koordinátába került a végberendezés. A mozgásról készült felvétel, az adott bemeneti konfiguráció mellett (3.8. ábra):

X final	Y final	q1 starting	q2 starting	Time
0.2	-0.5	$\pi/2$	$-\pi/6$	1

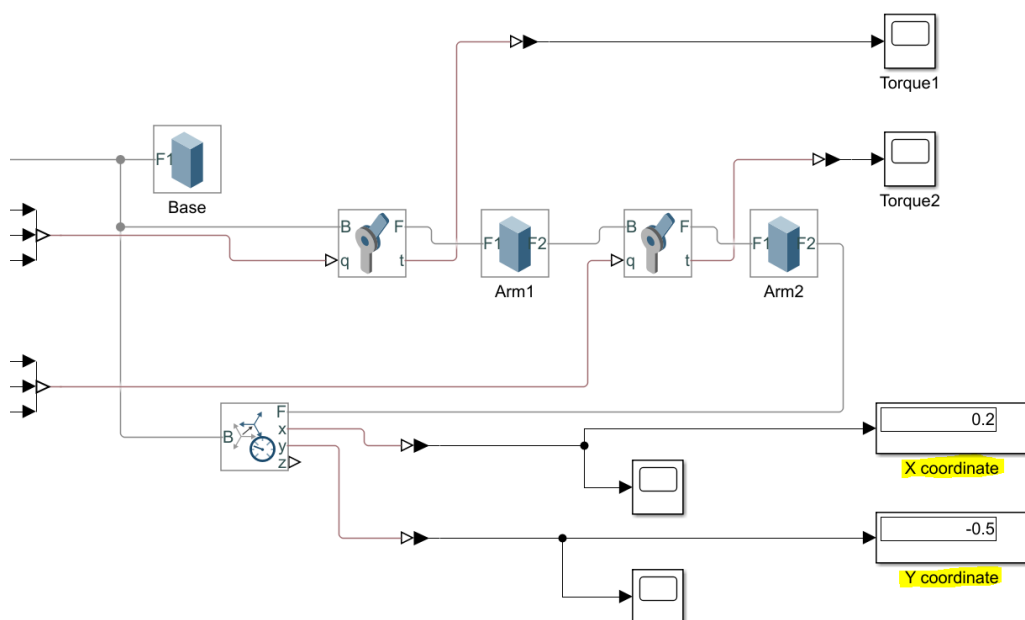
3.8. ábra - A teszt során beadott adatok



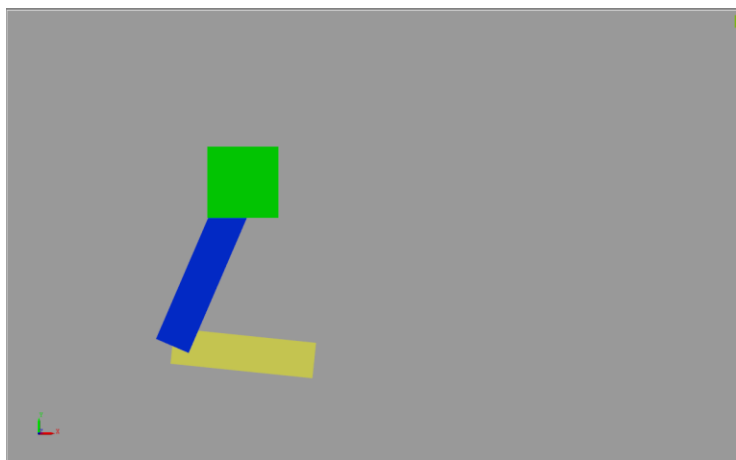
<https://youtu.be/03HE0dinZZU>

A videó kattintásra elindul (internet kapcsolat szükséges).

A futás végeztével a Simulink modell megjelenítőiről leolvashatóak a különböző jelek, pl. a nyomatékok, illetve a kalkulált szögek. Az *X coordinate* és *Y coordinate* megjelenítőn pedig megnézhető, hogy valóban beállt-e a kívánt pozícióba: a tesztek során látható, hogy ez tökéletesen működik.



3.9. ábra – A futás végeztével a mért pozíciók megegyeznek a megadott pozíciókkal



3.10. ábra – A robotkar a kívánt állásban a futást követően

4 Ütközésmentes pályatervezés

Az alábbi fejezetben egy olyan algoritmus kerül bemutatásra, ahol adott egy kiindulási q_0 és egy véghelyzeti q_f konfigurációs pár, ezek egy adott szöghelyzetet írnak elő mindkét csukló számára (q_1, q_2). Ezen kívül a munkatérben az akadályok ismertek, melyek szintén q konfigurációs térben vannak megadva. A feladat két konfiguráció között utat találni, és a robotkar végberendezését ezen az úton végig-mozgatni anélkül, hogy akadálnak ütközne. Először a felhasznált algoritmus alapjának a szakirodalomban megtalálható elmélete kerül ismertetésre, majd a Matlab segítségével implementált konkrét fejlesztés bemutatása következik.

4.1 Probabilistic Roadmap (PRM)

Az algoritmus elméleti háttérének megértéséhez Dr. Harmati István 2008-ban írt segédletét használtam. [3]

Mivel a robotkar fix munkatérben dolgozik, érdemes útvonaltérképet készíteni, mellyel egy inicializálást követően két pont közti út találása felgyorsítható. „A többszörös lekérdezésű térképmódszerek olyan algoritmusok, melyek feltételezik, hogy a feladatot nem csak egy adott konfigurációs párra kell megoldani, hanem ilyen párok sorozatára, miközben a robot fizikája és az akadályok helyzete, nagysága nem változik.” [3]

Ezen módszerek közül a legismertebb és leggyakrabban használt módszer az ún. valószínűség alapú útvonal-térkép készítő módszer (PRM), én is ezt használom.

A többszörös lekérdezésű módszerek két főbb lépésből állnak:

1. Előfeldolgozó fázis: egy útvonal-térkép készítése
2. Lekérdező fázis: egy adott konfigurációs párra konkrét útvonal megadása

A PRM módszer, a nevéből adódóan valószínűségi alapon működik, ami ez esetben azt jelenti, hogy a csúcsok kiválasztása/felvétele véletlenszerűen történik. A véletlenszerűséget a lekérdezési fázisban is kihasználom, azonban optimalizáció céljából, a determinisztikus működést és a véletlenszerűséget ötvözve.

A következőkben a két fázis alapvető algoritmikus megvalósításának, illetve a különböző adatok (gráf csúcsok, élek) tárolási módszerének bemutatása történik.

4.2 Az útvonaltérkép felépítése

Belátható, hogy a pályatervezés egy gráf probléma: a gráf jó felépítése, majd annak optimális bejárása kulcsfontosságú. Ezért is fontos a felépített gráf csúcsainak és éleinek hatékony és kényelmesen lekérdezhető tárolása.

A feladat kezdetén három alapvető dolgot kellett eldönteni, miképp legyen tárolva:

- Az akadály
- A felépített gráf csúcsai
- A felépített gráf élei

A megvalósítást Matlab kódban összesen 6 függvénnyel és egy main.m scripttel oldottam meg, majd a munkatérből egy végleges mátrixot kötöttem a Simulink modellre. A függvények közül kettő kerül részletesebb bemutatásra, illetve a főprogram felületes ismertetése történik.

4.2.1 Akadályok tárolása

A feladat megoldása legegyszerűbben a (q_1, q_2) konfigurációs térben valósítható meg, ahol q_1 az első csukló, q_2 a második csukló korábban részletezett szöge. Mivel az akadályok geometriájának és kiterjedésének a két alak (az (x, y) koordinátás alak és (q_1, q_2) csuklózváltozós alak) közti megfeleltetése nem triviális, célszerűen (q_1, q_2) csuklózváltozók szerint tárolom az akadályokat. Ezt érdemes úgy megtenni, hogy a munkatérrel diszkrétizálom N felosztásban. Mindkét csukló $\pm 175^\circ$ -os elforgásra képes, ami $N = 1^\circ$ -os felosztással 351 állapotot jelent mindkét csuklónak ($2 \cdot 175$ és a 0).

N értéke a programban változtatható egyéb paraméterek változatlanul hagyása mellett is, ezáltal a felosztás akár finomítható, azonban 1° -os pontosság kellően reprezentálja a valós robotkar képességeit, ezért a továbbiakban ezt használom.

Így egy ún. „bitmap” -ben tárolhatóak az akadályok, ami egy $N \times N$ -es mátrix (ez esetben 351×351), ahol a sorok q_1 (az első csukló szöge), az oszlopok pedig q_2 (a második csukló szöge) szerinti felosztása a bejárható tartománynak. A bitmap elemei 0 vagy 1 értéket vehetnek fel, attól függően, hogy ahhoz a konkrét elemhez tartozó q_1 és q_2 szögeken akadály van-e. Elhatározás alapján: 1: szabad, 0: akadály.

bitmap = $\begin{pmatrix} 1 & \dots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \dots & 1 \end{pmatrix}$ $N \times N$ -es mátrix, ahol akadály található ott 0 az elem értéke.

4.2.2 Csúcsok és élek tárolása

Egy csúcsot, azaz (x_1, y_1) koordinátát, két csuklóváltozó pár is reprezentálhat, azonban az akadályok helyzete miatt előfordulhat, hogy csak az egyik állásban nem ütközik akadályba a robotkar. Ezért célszerűen, egy csúcsot (q_1, q_2) szög párral írok le. Ez a szögpár lehetséges, hogy egy másik párral ugyanazt a végberendezés koordinátát jelenti, azonban ez a feladatból kifolyólag nem hátrány.

A *bitmap* sorainak és oszlopainak sorszámozása a Matlab gyakorlatának megfelelően 1-től kezdődik. Tehát az 1. elem a -175° -ot, a 351. elem pedig a $+175^\circ$ -ot reprezentálja. Az egyszerűség kedvéért, ezt az 1—351 tartományt használok a csúcsok reprezentálására is, és minden számítást is így végzek. Csak a legutolsó lépésben váltom vissza a csuklók számára is megfelelő radián értékekre a kijelölt tartományon. A csúcsok tárolására egy olyan $M \times 2$ -es mátrixot használok, ahol M a tárolt csúcsok száma. Így minden sor két elemet tartalmaz, a két szöget. Minden csúcsot egyértelműen azonosít a sorának a sorszáma.

$$\mathbf{V} = \begin{pmatrix} q_{11} & q_{12} \\ q_{21} & q_{22} \\ \dots & \dots \\ q_{M1} & q_{M2} \end{pmatrix}, \text{ ahol minden „} q_{xy} \text{” 1 és 351 közti egész számú értéket vehet fel.}$$

Egy él tárolását kézenfekvően a két összekötött csúcs nyilvántartásával oldottam meg. Az élek egy olyan $K \times 2$ -es mátrixban vannak tárolva, ahol a sor két eleme egy-egy index, ami a csúcsok \mathbf{V} mátrixában egy sort jelöl, így kijelöli a konkrét csúcsot.

$$\mathbf{E} = \begin{pmatrix} V_{11} & V_{12} \\ V_{21} & V_{22} \\ \dots & \dots \\ V_{K1} & V_{K2} \end{pmatrix}, \text{ ahol minden „} V_{xy} \text{” a } \mathbf{V} \text{ mátrix egy sorára mutató érték, pozitív egész.}$$

A \mathbf{V} és \mathbf{E} mátrixok nagysága nem magától értetődő. A térkép elkészítése során megadható, hogy hány csúcsot tartalmazzon az erdő (jobb esetben fa). Az is megadható, hogy hány csúccsal próbáljon meg összekötni egy újonnan felvett csúcsot, azonban lehetséges, hogy csak kevesebbel, vagy akár egyel sem sikerül összekötni. Ebből kifolyólag, az \mathbf{E} mátrix nincsen előre lefoglalva, hanem új él találása esetén bővül, ez nem nagy baj, hiszen sok újra foglalásra csak az egyszer futtatandó előfeldolgozó fázisban van szükség. A térképkészítésnek éppen ez a funkciója, hogy kissé számításigényesebben felépít egy útvonalterképet a változatlanul hagyott munkatérben, de ezt mindössze egyszer teszi meg, így amikor konkrét útvonalat kell keresni a futás jelentősen gyorsítható.

A lekérdező fázisban, egy lekérdezés után **V** legfeljebb 2 sorral bővül, azonban előfordulhat, hogy a kezdeti-, vagy célcúcs már hozzá lett adva a térképhez, tehát **V** akár változatlan is maradhat. Ebben a fázisban **E** hasonlóan viselkedik, attól függően hány élt sikerül találni az újonnan felvett csúcsokhoz, ha egyáltalán fel kellett venni új csúcsokat.

4.2.3 Az *Init* függvény

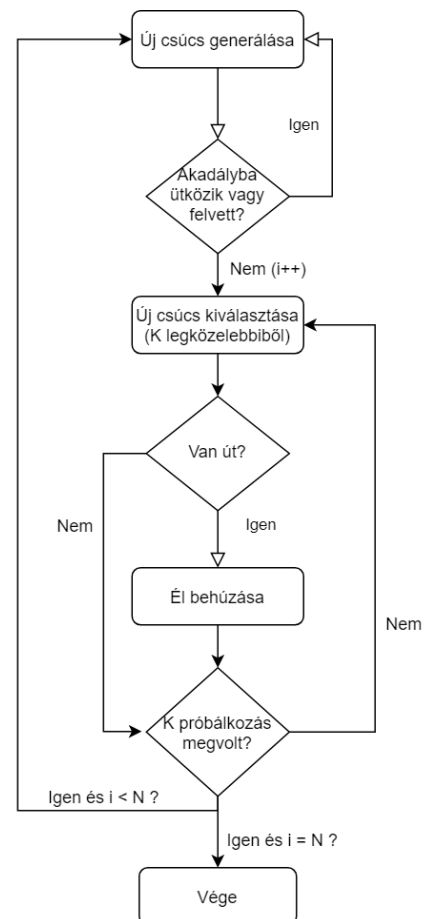
Az *Init* nevű Matlab függvény végzi el a kiindulási fa felépítését, a *trajectory* függvényből kerül meghívásra egyszer.

Bemenetei: a *bitmap*, ennek a mérete (*bm_size*), valamint a felépítendő fa nagysága, azaz a csúcsok száma (*iterationNum*).

Kimenetei: a felépített **V** és **E** mátrixok, valamint az élek száma (*edgeCount*), utóbbira azért van szükség, mert a talált élek száma nem tudható előre, így nem kell az **E**-t később megszámolni.

Működése röviden: (4.1. ábra)

A Matlab beépített *randi* függvényével egy kételemű sorvektort generálok véletlenszerűen egyenletes eloszlással, melynek két eleme 1 és *bm_size* közötti egész, ezek a két csuklóváltozók, melyek egy új csúcsot reprezentálnak. Ezek után az általam implementált *isFree* függvénnyel megnézem, hogy ez nem ütközik-e akadályba, majd ellenőrzöm, hogy nem része-e a már felvett csúcsok halmazának. Ha nem, akkor felveszem és távolságot számítok a többi csúcstól a *dist* függvényemmel, a távolság adatokat mátrixban tárolom, majd ezt a mátrixot távolság szerinti növekvő sorrendbe rendezem. Az első *K* (= 15-öt választottam) elemre megnézem az *isRoute* függvényemmel, hogy van-e ütközésmentes út az újonnan felvett csúcsához, amennyiben igen, azokat az élek halmazához adom. A folyamat ciklikusan ismétlődik, amíg *iterationNum* számú csúcs megtalálásra nem kerül.



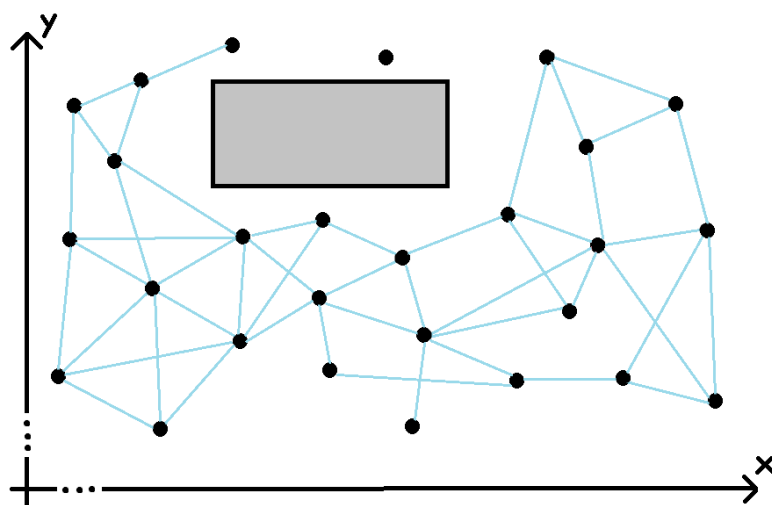
4.1. ábra - Inicializálás

Tervezés során úgy döntöttem, hogy megengedem a köröket a gráfban, ezzel a lokális útkeresés probléma jelentősen bonyolódik, azonban nagyobb eséllyel találok optimális utat, hiszen több elem lesz a gráfban.

4.3 Lokális útvonal keresés

Az útvonaltérkép felépítése c. fejezetben (4.2. fejezet) ismertettek szerint az útvonaltérkép felépült. Ezek után a feladat: tetszőleges csuklópontok közt egy optimális utat találni. A két új pont a felépített térképbe kerül bekötésre, ezek után a meglévő ütközésmentes útvonalakat biztosító éleken kell gráfbejárást végezni és utat találni két csúcs közt.

A felépített gráf köröket tartalmazhat, a bejárás nem triviális. Az egy csúcsból induló maximális élek száma paraméterként állítható, azonban az optimális utak számának növelése érdekében 15 és 20 közé lett választva, de ez a szám futás során természetesen kisebb, akár 0 is lehet, attól függően sikerült-e utakat találni. A felépített fa rengeteg élt tartalmaz, ezért az útvonal kereső algoritmus alapja a véletlen sétálás módszer azzal a finomítással, hogy az egyes élek súlyozottan szerepelnek aszerint, hogy egy a végponthoz közelebbi vagy távolabbi csúcsba vezetnek. Ezek közül az élek közül kerül fix számú élt véletlenszerűen kiválasztásra, majd a leoptimálisabb súlyún halad tovább az algoritmus. A körbe-körbe sétálás valószínűsége a súlyozott élek miatt csekély, de körbeérés esetén visszalép addig, amíg biztosítva nincs, hogy ne körbe haladjon. Ugyanígy, ha távolabb kerül az elérni kívánt ponttól szintén visszalép és próbálkozik jobb út keresésével.



4.2. ábra – Egy felépített gráf egyszerűsítve és XY koordinátarendszerben ábrázolva

4.3.1 A *trajectory* függvény

Az előbbiekben leírtakat a *trajectory* függvény valósítja meg, azaz a tényleges útvonalkeresést. A *main* scriptben kerül meghívásra.

Bemenetei: A kiindulási szögek (*q1_start_rad*, *q2_start_rad*) és a célszögek (*q1_final_rad*, *q2_final_rad*) radiánban, nem eltolt módon (nem a bitmaphez igazítva) megadva.

Kimenetei:

- Egy bool változó, ami igaz, ha talált utat (*isPath*)
- Egy egész típusú változó, ami a mozgási szakaszok számát adja meg, hány csúcsot érintve jut el a véghelyzetbe (*compLength*)
- Illetve egy mátrix, nagysága: *compLength* x 2, ami a konkrét úthoz tartozó csúcsokat tartalmazza, radiánban, a bemenetek formátumában. Ezen csúcsok közt van már út, ami a *q1*, *q2* lineáris változtatásával bejárható (*finalQ*).

Működése röviden:

A bemenetek átváltását követően, az *Init* függvényben látható módszerrel bekötésre kerül a két új csúcs (amennyiben már be voltak kötve, akkor az megjegyzésre kerül). Ha nem akadályra esnek a pontok, akkor a gráf bejárása történik *D* és *W* paraméter szerint, ahol *D* az útvonal próbálkozások száma, *W* pedig ezen útvonalak maximális hossza. A gráfbejárás egy olyan véletlen sétáláson alapuló módszer, amely célirányosan (súlyozottan) a kezdeti pontból a végpontba tart. Mivel a gráf köröket tartalmaz(hat), ha egy már bejárt csúcsra érkezünk újra, az algoritmus visszalép és újra próbálkozik, ez a súlyozás miatt nem túl nagy eséllyel fordulhat elő. A bejárás a felépített fán gyakorlatilag mindig megtalálja a legjobb utat, azonban a *main* újra futtatásakor új fa (vagy erdő) épül fel, így mindig új fán történik az útvonalkeresés, ebből ered az újra futtatások közti különbség az utakban (az új fa felépítése valós felhasználás során természetesen elhagyható, amennyiben az *init* függvényt (ld. 4.2.3. fejezet) csak egyszer hívjuk meg és utána a *trajectory* függvényből eltávolításra kerül a meghívása).

A talált utaknak ezek után kiszámolom a teljes hosszát (ez nem feltétlen arányos a komponensek számával), majd a legrövidebbet választom végleges útnak (itt a végberendezés koordináták szerinti úthosszát számolom, ez könnyen módosítható, hogy a csuklófordulás útja legyen a legkisebb). Végül az értékeket visszaváltom radiánba és ezekkel feltöltöm a kimeneti *finalQ* változót. Ha nincs út, akkor *isPath* = false kerül átírásra, ekkor *finalQ* a bemeneti kezdeti értéket tartalmazza csak, és *compLength* = 0.

A függvény néhány futási paramétere tetszőlegesen változatható, az ilyen paraméterek a függvény elején vannak összegyűjtve, ahol ezek magyarázattal vannak ellátva, hogy az algoritmus futását miképp változtatják. Az akadályok hozzáadása is itt történhet többféle módszerrel, minden esetben XY koordinátarendszer szerint, téglalapok és hengerek is különböző méretekben, akár betanított pozíciókból. Az akadályok transzformálásáról és betanításáról a későbbiekben lesz szó (ld. 5.2. fejezet).

4.3.2 Egyéb függvények

Néhány gyakran használt funkciót külön függvénybe mentettem ki, ezek működése egyszerű, különösebb magyarázatot nem igényel. A négy függvény és röviden a funkciójuk:

- **isFree**: a beadott csúc és bitmap alapján eldönti, hogy a csúc nem esik-e akadályba.
- **isRoute**: két darab beadott csúc és a bitmap segítségével eldönti, van-e útvonal a két csúc között, itt az útvonal q_1 és q_2 szögek lineáris változtatásával értendőek, ez az (x,y) térben nem egyenes.
- **dist**: távolságot számol két csúc között, itt a q_2 és q_1 mozgása is azonos súlyú, ez változtatható.
- **dist_XY**: távolságot számol, de a csúcsokat átváltja koordinátás alakba és Euklideszi távolságot számít

4.4 Az algoritmus tesztelése szimulációban

Mielőtt a fizikai hardverrel foglalkoztam volna, az elkészített algoritmust A robotkar modell elkészítése c. fejezetben (2. fejezet) megalkotott modell segítségével és a Robotkar mozgás implementáció szimulációban c. fejezetben (3. fejezet) bemutatott mozgásegységeket felhasználva teszteltem, egyelőre szabályozási kör nélkül szimulálva az akadálykerülést.

4.4.1 Főprogram: main.m Matlab script

A korábban ismertetett Simulink modell változtatásra került. A csuklók bemeneti jeleinek számítása a *main* scriptben történik, ez a kód az eddig leírtakat foglalja össze: az időfüggvény szerinti csuklóváltozók számítása az útvonalkereső algoritmusok által kijelölt pontok közt és ezek betöltése egy a Simulink számára értelmezhető mátrixba.

A script elején be kell állítani a kezdeti és az elvárt szögeket radiánban, ezen kívül a szimulációs idő (*sumTime*) megadása is szükséges. Az utolsó állítható paraméter (*dec*)

a mozgás „finomságát” állítja, azaz, hogy a szimulációs időre milyen sok csuklóváltozó számítása történjen. Minél nagyobb ez a szám, a felbontás is annál nagyobb lesz: ha pl. $dec = 100$, akkor századmásodperces (0.01) felbontással kerül behelyettesítésre a csuklók időfüggvényébe az idő változó. Alapjáraton $dec = 1000$ lett beállítva, tehát 1 másodperc során 1000 érték kerül a Simulink blokk bemenetére.

A *trajectory* lefuttatása után, amennyiben van talált út, a két mátrix feltöltése történik: a **finalQ1** és **finalQ2**. Mindkét mátrix 4 oszlopot és $sumTime*dec$ sort (alapjáraton 10 000 sort) tartalmaz. Ahol a maradékok miatt nem jön ki a 10 000, ott az elemeket értelemszerűen pótoltam: q_x állandó, a deriváltak 0, mozgás nincs, az idő tovább növekszik.

$$\mathbf{finalQ1} = \begin{bmatrix} 0 & q_s & \dot{q}_s & \ddot{q}_s \\ t_1 & q_1 & \dot{q}_1 & \ddot{q}_1 \\ t_2 & q_2 & \dot{q}_2 & \ddot{q}_2 \\ \vdots & \vdots & \vdots & \vdots \\ t_n & q_n & \dot{q}_n & \ddot{q}_n \end{bmatrix}, \mathbf{finalQ2} \text{ hasonlóan.}$$

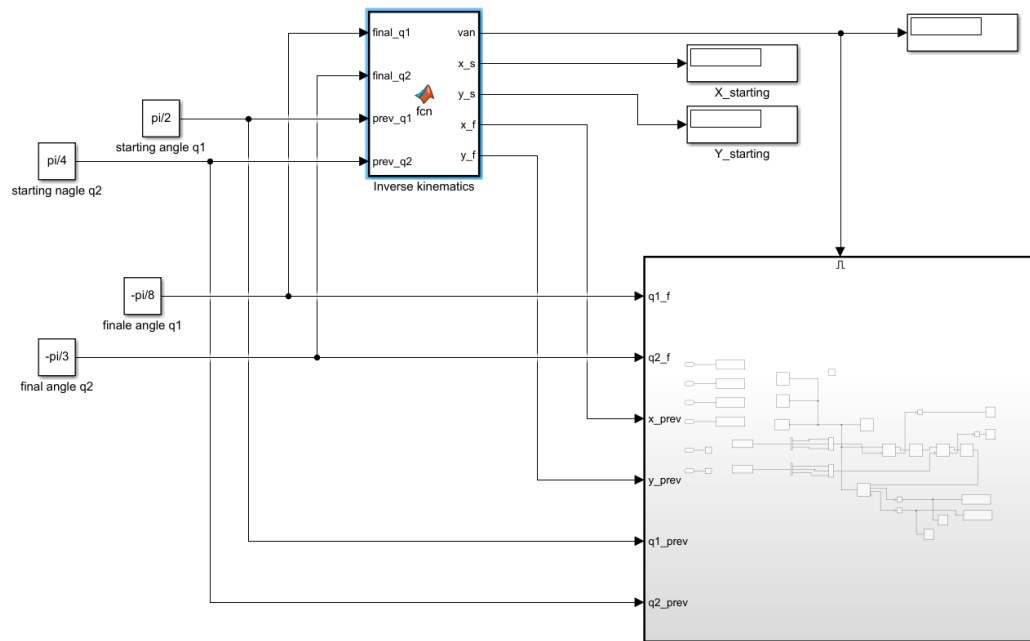
$n = sumTime*dec$. q_s = kiinduló szög, q_n = végső szög

A Simulink modellben ezt a két változót használok fel, ezért az értékeinek az első oszlopa az idő szerinti függést jelenti, azaz itt $\frac{1}{dec}$ felosztással 0-tól $(sumTime - \frac{1}{dec})$ -ig vannak növekvő sorban a számok. A második oszlop az ahhoz a sorhoz tartozó időpillanat szerinti csuklóváltozót tartalmazza, a harmadik és negyedik oszlop pedig az 1. és 2. deriváltak értékeit szintén ehhez az időpillanathoz. Ezek a képletek részletes magyarázattal már ismertetésre kerültek a 3.3.2. fejezetben.

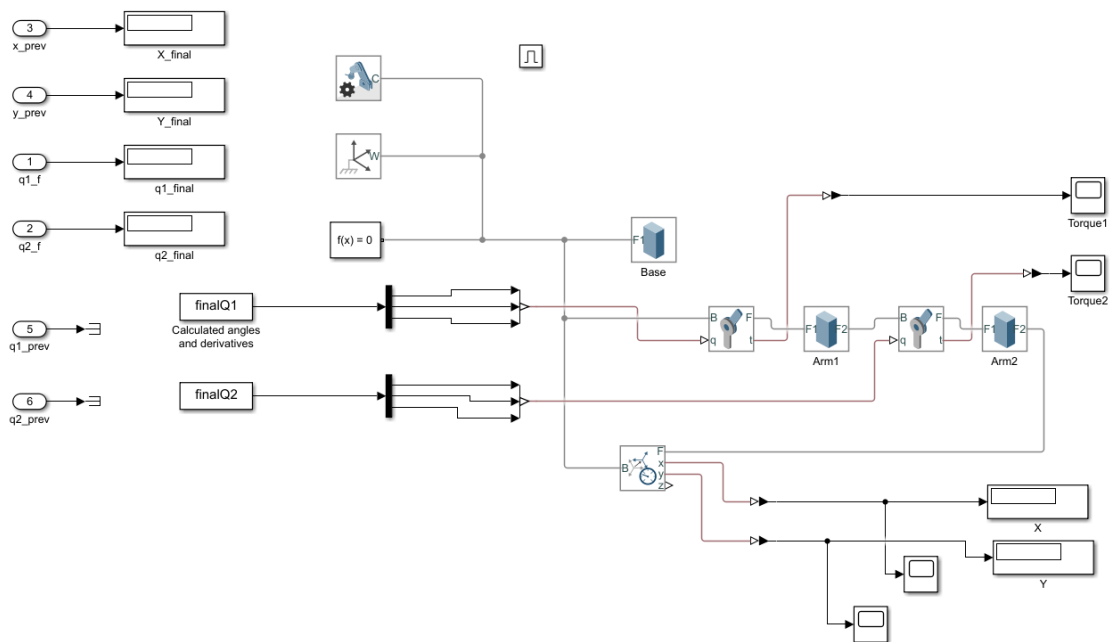
4.4.2 A módosított Simulink modell

Mivel az adatok a munkatérből kerülnek kiemelésre, ezért a *Joint Movements* függvény eltávolításra került, a csuklóváltozókra egy demultiplexeren és egy *Simulink-PS* konverteren keresztül közvetlenül a **finalQ1**, **finalQ2** mátrixok kerültek. Az alrendszeren kívül az *Inverse kinematics* függvény csak azért van meghagyva, mert azt átírva jelenítettem meg koordinátás alakban a vég és kezdeti pontokat.

Az akadályok a konfigurációs térben vannak megadva, ezek megjelenítve rendkívül amorf alakot vennének fel, ezért a szimuláció során nem látszódnak. A későbbiekben (ld. 7.1. fejezet), a valós rendszer tesztelésekor az akadályok természetesen látványosak lesznek és könnyen megadhatóak a munkatérben.



4.3. ábra - Ütközésmentes modell



4.4. ábra – Ütközésmentes modell: Alrendszer

4.4.3 Az ütközésmentes pályán való végig haladás


Az ismertetettek szerinti implementálásban, a szimuláció futtatható, és a futás eredménye egy adott konfiguráció mellett megtekinthető a videón. Az akadályok csuklóváltozóknak lettek megadva, kizárólag tesztelés céljából, ezek nehezen

követhetőek, azonban futás során a kódban ellenőrizhető volt az ütközés mentesség. Ez a későbbiekben bemutatásra kerülő valós rendszeren végzett teszteken jobban vizualizálható.

q₁ kiindulási	q₂ kiindulási	q₁ végpozíció	q₂ végpozíció
$\frac{3\pi}{4} \approx 2.356$	$\frac{\pi}{2} \approx 1.571$	$-\frac{\pi}{2} \approx -1.571$	$-\frac{\pi}{2} \approx -1.571$

4.5. ábra – Teszt során megadott kezdeti és végső csuklópozíciók

A fenti táblázatban (4.5. ábra) szereplő kezdeti-, és végpozíciók lettek megadva az algoritmus számára. Az akadályok a konfigurációs térben kerültek definiálásra, szándékosan olyan helyre, hogy a kezdeti- és a végpozíció közt csak több komponensű út legyen. Az akadályok megadására a kódban a korábban említett (ld. 4.2.1. fejezet) *bitmap* rendszerben volt lehetőség, ez a későbbiekben egyszerűsítésre kerül (ld. 5.2. fejezet). Az algoritmus a futtatást követően a két pont közt egy 3 komponensű utat talált, mely azt jelenti, hogy két köztes pont beiktatásával, összesen 3 – a konfigurációs térben értendő – egyenes mentén való mozgás során jut el a végpozícióba. A két köztes pont és a végpozíció a *finalQ* változóba kerül bemásolásra (4.6. ábra): a mátrix első oszlopa az első csuklóhoz tartozó pontokat tartalmazza, a második oszlop a második csuklóhoz tartozó pontokat.

 3x2 double

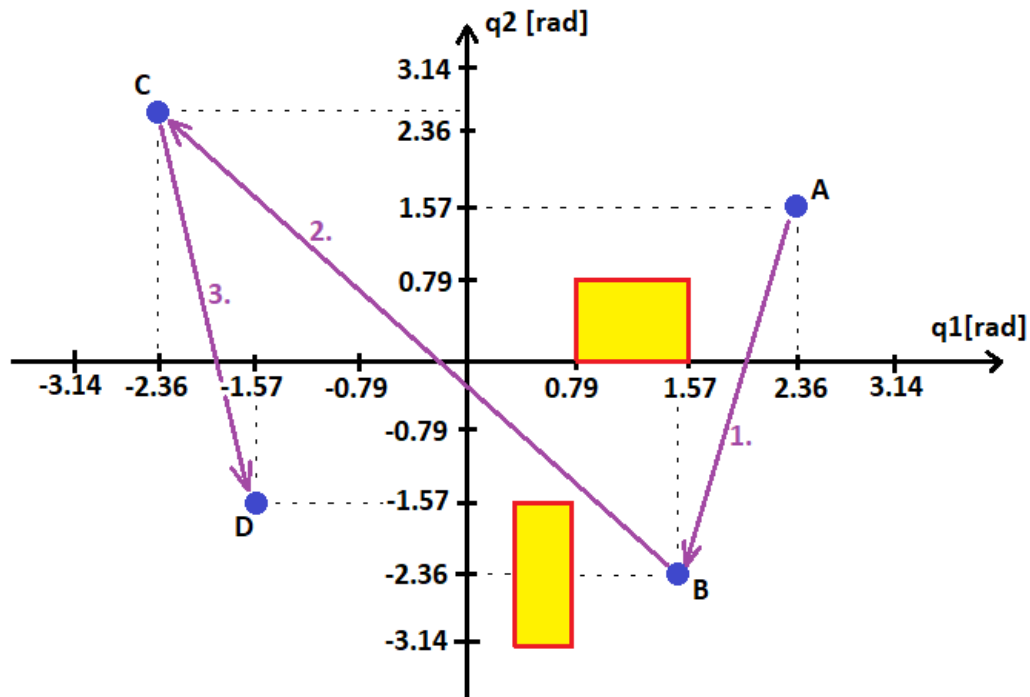
	1	2	3
1	1.4835	-2.3562	
2	-2.3911	2.5482	
3	-1.5708	-1.5708	
4			

4.6. ábra – A mozgáshoz tartozó két köztes pont és végpozíció (1. oszlop: q₁, 2. oszlop: q₂)

Ezen pontok a felépített gráf részei, ezek közt létezik ütközésmentes, „egyenes” út. A pontok közt ezred másodperces felbontással kerülnek kiszámításra (ld. függelék 8.5.) az egyenes pontjai, a korábban ismertetettek szerint (ld. 3.3.2. fejezet).

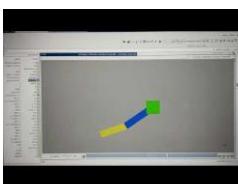
Az akadályok és a bejárt pálya szemléltetéséhez a lenti ábra (4.7. ábra) nyújt segítséget. Fontos, hogy az ábra a konfigurációs teret mutatja, nem pedig az XY tér pontjait, azaz a tengelyek a csuklótérbe vannak ábrázolva: a vízszintes tengely az első-, a függőleges a második csuklóhoz tartozó értékeket jelöli. A konfigurációs tér és az XY tér pontjai közti transzformációról a későbbiekben (ld. 5.3. fejezet) lesz szó. Az ábrán (4.7. ábra) a mozgás

kezdeti pontja A-val, a végpontja D-vel, a köztes beiktatott pontok pedig B-vel és C-vel vannak jelölve. A felvett akadályokat sárgával kitöltött téglalapok szemléltetik, a pontok közti bejárt utat pedig a lila számozott vonalak, melyek számozása megegyezik a tényleges bejárás sorrendjével. Az akadályok alakja XY térben jelentősen eltér a konfigurációs térben ábrázolttól, jelen esetben az akadály amorf formájú lenne.



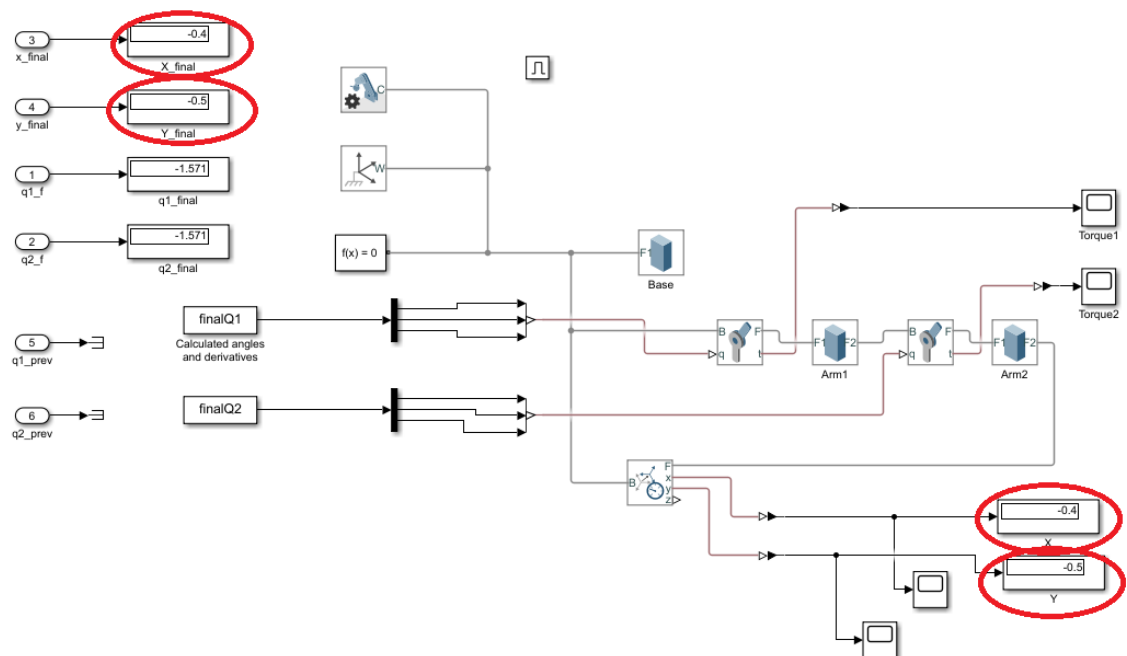
4.7. ábra – A mozgáshoz tartozó vizualizáció a konfigurációs térben (akadályok: sárga, bejárt gráf pontok: kék, útvonalak: lila)

Látható, hogy A és D pontok között nincs egyenes út, ezért két köztes pont került beiktatásra. A PRM sajátossága miatt nem feltétlen a legoptimálisabb út kerül megtalálásra: most is látható, hogy létezik kisebb csuklómozgással járó pálya. Ezen felül a komponensek száma is jelentősen változhat az újra futtatások során. Ennek ellenére az algoritmus működik, az akadályokat a végberendezés elkerüli. (A teljes robotkar, azaz a szegmensek ütközésmentességének vizsgálata nem ezen szakdolgozat témaköre (ld. 8.2. fejezet).) A mozgásról készült videó megtekinthető (internetkapcsolat szükséges):



https://youtu.be/v0W9N_HT4Q8

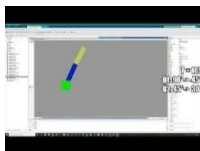
A futás végeztével ellenőrizhetők a megjelenítőkön, hogy a robotkar valóban a megfelelő végkoordinátába került. Itt a számítási kerekítések és a diszkretizálás folytán előfordulhatnak néhány ezredes jegy eltérések az elméletileg számított értékektől, ez a hiba sosem volt nagyobb 1%-nál, jelen esetben 0.



4.8. ábra – A robotkar az elméleti pozícióba beáll ($X = -0.4$ és $Y = -0.5$)

A valószínűségi módszerek előnye itt is érvényesül, kevesebb számítással, kellően jó eredmény kapható. Azonban pont a teljes determinisztikusság hiánya miatt előfordulnak esetek, mikor az algoritmus futása során a talált út nem kifejezetten optimális és felesleges mozgásokat tartalmaz, ez a futási paraméterek és a térkép részletességének számbeli növelésével kiküszöbölhető, ugyanakkor a számítási igényeket jelentősen növelheti.

Egy másik elrendezésről készült teszt videója (más akadályok és pozíciók):



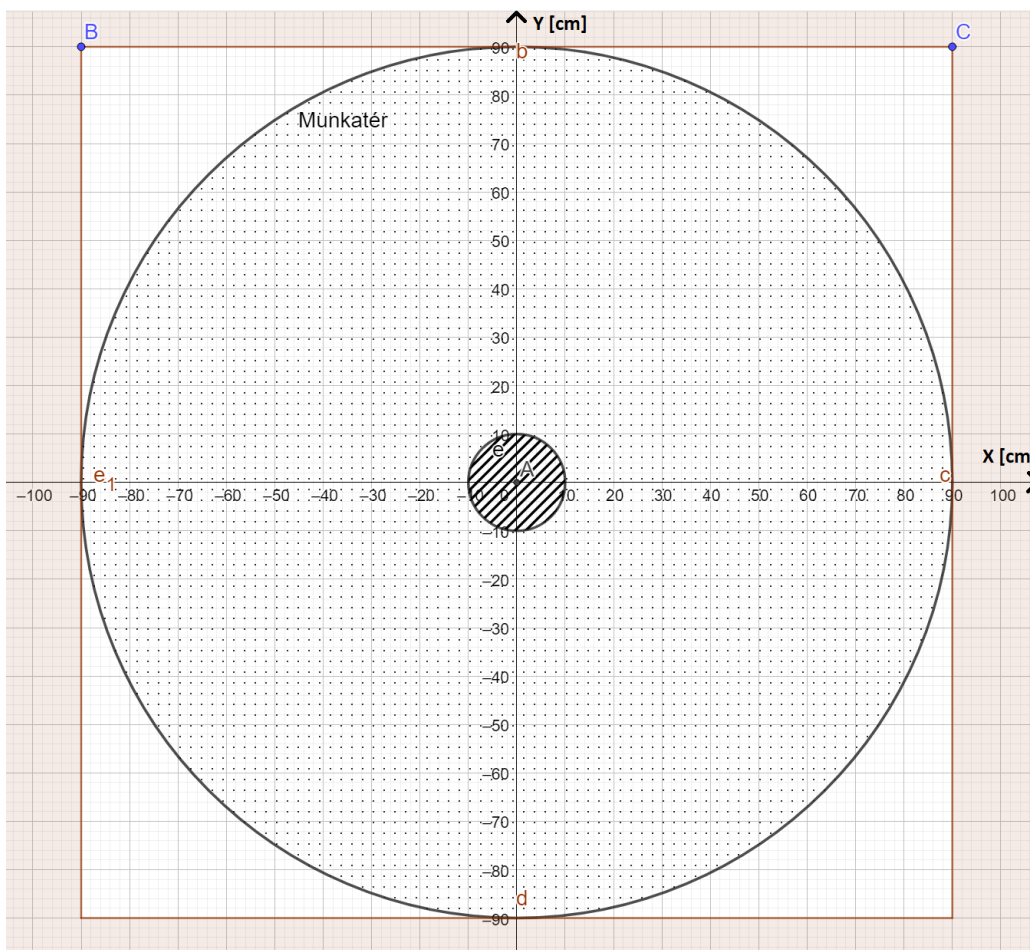
<https://youtu.be/VYzvW8Ny9GM>

5 Akadályok a munkatérben

Az előző fejezetben (4. fejezet) végig azzal a feltételezéssel éltem, hogy minden csuklópozícióhoz meg van adva, hogy akadályhoz tartozik vagy sem, azonban azzal nem foglalkoztam, hogy ez az akadályokat tároló *bitmap* hogyan áll elő. Ebben a fejezetben, arról lesz szó, hogy milyen lehetőségek állnak rendelkezésre az akadályok felvételére, illetve az XY térben megadott akadályok miként kerülnek transzformálásra a csuklópályázók rendszerébe.

5.1 Akadályok tárolása Descartes-koordináta rendszer szerint

Az XY koordinátarendszerben megadott akadályokat könnyebb vizualizálni, ezért szükségszerűen felmerül a kérdés, hogy ezek miként legyenek tárolva, illetve miként lehet a mozgástervező algoritmus által elvárt csukló koordinátarendszerbe transzformálni az akadály pontjait.



5.1. ábra – A robotkar által elérhető pontok (végállásokat nem tekintve)

Az akadályok által elfoglalt pontok tárolására kézen fekvő egy – a korábban használt konfigurációs térhez hasonlóan – bitmap definiálása. Ehhez a munkateret diszkrét pontokra kell felosztani, a felosztás „finomsága” önkényes és tetszőlegesen módosítható. Mivel a korábbiakban a felosztás 1° -os pontosságú volt, ami (a végállásokat nem tekintve) egy 361×361 nagyságú mátrixot jelentett, így kezdetnek az XY koordinátarendszer bitmapjét is egy 361×361 nagyságú mátrixnak definiáltam.

Mivel a pályatervező algoritmus a csuklókoordinátás bitmap szerint dolgozik, ahol le vannak kezelve a robotkar csuklóinak végállásából adódó korlátozások, ezért az XY koordinátarendszeres bitmapben a végállásokkal nem foglalkozom, hiszen ezek a pontok egész egyszerűen a transzformálás során nem kerülnek bele a végleges bitmapbe.

A robotkar végberendezése az origótól 90 cm-től nem messzebbi és a 10 cm-nél nem közelebbi pontokat éri el, ami a fenti ábrán a körön belüli pontozott terület (5.1. ábra). A kör átmérője tehát 180 cm, amit fél centis felbontással reprezentálva előállítható egy 361×361 nagyságú mátrix ($180 \cdot 180 + 1$ az origó miatt). Ez a mátrix egy négyzet pontjait reprezentálja fél centis pontossággal (5.1. ábra), ez a felbontás megfelelően kicsi a fizikai robotkaron való tesztekhez. Ez a tárolási mód kézenfekvő, azonban a mátrix tartalmaz olyan pontokat, amelyeket a robot végberendezése semmiképpen sem tud elérni, ezek a pontok a könnyebb kezelhetőség miatt megjelölésre kerülnek.

Ha egy pont a körön kívülre esik, akkor a cellába -99 szám kerül beírásra. Ezzel hagyományos értelemben véve a mátrix már nem nevezhető bitmapnek (mert már nem csak két értéket tartalmazhat egy cella), azonban az egyszerűség kedvéért ezt az elnevezést megtartom.

$$\text{Ha } \sqrt{x_p^2 + y_p^2} > (a_1 + a_2), \text{ akkor } \text{bitmap}(x_{row}, y_{col}) = -99$$

Az elérhető pontok a korábbi jelölésrend szerint 1 értékűek, az akadályok pedig 0.

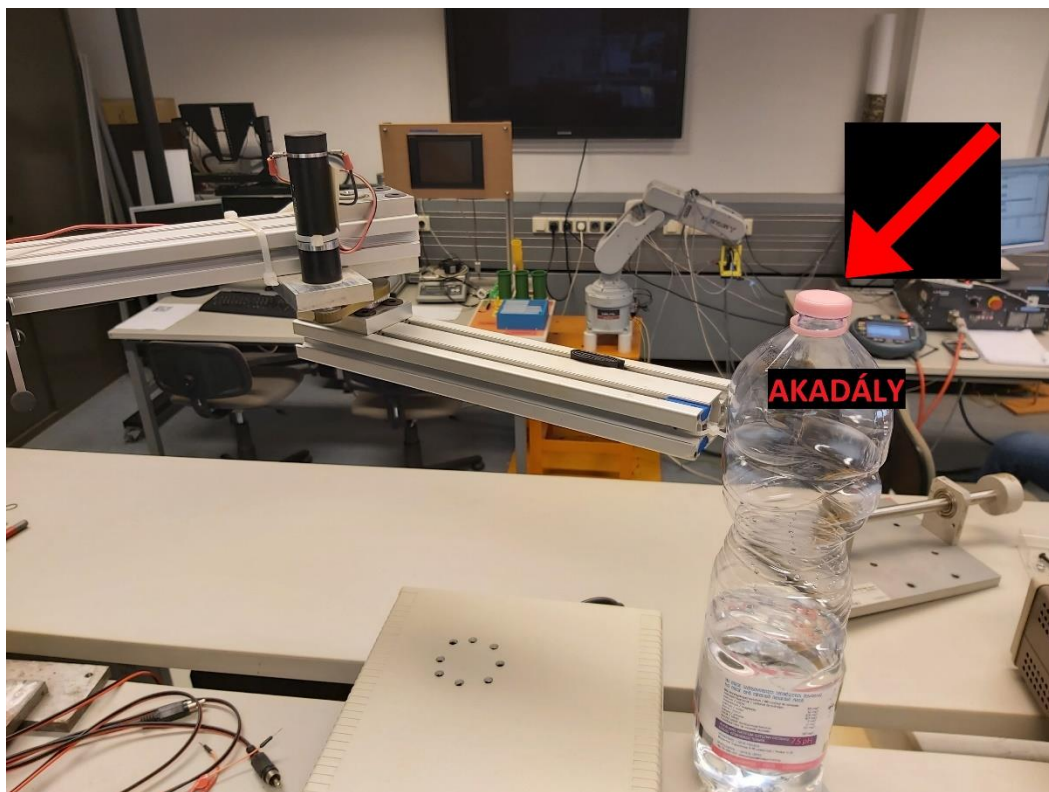
$$\text{bitmap_xy} = \begin{pmatrix} 1 & \dots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \dots & 1 \end{pmatrix}, \text{ ahol az akadály pontjai 0, a nem elérhető pontok pedig -99.}$$

5.2 Akadályok felvétele

Az akadályok felvételére több lehetőség is adott, a legkomplexebb akadályok megadásához akár a tér pontjai egyesével is állíthatóak. Ez a módszer bár sok szabadságot jelent az akadályok alakjával kapcsolatban, nyilvánvalóan hosszadalmas és körülményes

feladat. Egyszerűbb geometriák manuálisan is megadhatóak, pl. téglalapok és körök (középponttal és sugárral).

A legkonvencionálisabb megadása az akadályoknak, az iparban is elterjedt ún. betanítás. Betanítás során feltételezem, hogy az akadályok henger alakúak lehetnek. A betanítás menete egyszerű: a későbbiekben bemutatásra kerülő Simulink modellt elindítva, a szabályozást kikapcsolva, a robotkar végberendezését (ami jelen esetben egy kábelkötegelő) az objektum 3 különböző pontjához kell mozgatni kézzel (5.2. ábra), ezekben a pontokban az enkóder által mért csukló szögek leolvashatóak a Simulinkben. Az így leolvasott radián értékeket a *trajectory* függvényben a paraméterek közé kell bevezetni.



5.2. ábra – Akadály betanítása

A szögekből először koordinátákat számolok a már ismertetett képlet alapján (ld. 3.1. fejezet), mint az korábban bemutatásra került ez a megfeleltetés egyértelmű. Az így megkapott három XY pár segítségével egyértelműen kiszámítható a kör középpontja és sugara.

Egy kör általános egyenlete: [5]

$Ax^2 + Ay^2 + Bx + Cy + D = 0$, ahol (x,y) a kör középpontja.

Ha ismert három pont a körön, akkor az alábbi determináns írható fel: [5]

$$\begin{vmatrix} x^2 + y^2 & x & y & 1 \\ x_1^2 + y_1^2 & x_1 & y_1 & 1 \\ x_2^2 + y_2^2 & x_2 & y_2 & 1 \\ x_3^2 + y_3^2 & x_3 & y_3 & 1 \end{vmatrix} = 0$$

Az együttható mátrixok megadhatóak a következő aldeterminánsok megoldásával: [5]

$$A = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} \quad B = \begin{vmatrix} x_1^2 + y_1^2 & y_1 & 1 \\ x_2^2 + y_2^2 & y_2 & 1 \\ x_3^2 + y_3^2 & y_3 & 1 \end{vmatrix}$$

$$C = \begin{vmatrix} x_1^2 + y_1^2 & x_1 & 1 \\ x_2^2 + y_2^2 & x_2 & 1 \\ x_3^2 + y_3^2 & x_3 & 1 \end{vmatrix} \quad D = - \begin{vmatrix} x_1^2 + y_1^2 & x_1 & y_1 \\ x_2^2 + y_2^2 & x_2 & y_2 \\ x_3^2 + y_3^2 & x_3 & y_3 \end{vmatrix}$$

Ezen együtthatók segítségével, kiszámítható a középpont koordinátái és a sugár: [5]

$$x = -\frac{B}{2A} \quad y = -\frac{C}{2A} \quad r = \sqrt{\frac{B^2 + C^2 - 4AD}{4A^2}}$$

Miután rendelkezésre állnak a kör adatai, a kör összes belső pontját és a körön lévő pontokat a bitmapen 0 értékre állítom. Ha a kör (azaz az akadály) egy része az elérhető tartományon kívülre esik, ott a korábban ismertetett -99 értékű cellát hagyom meg.

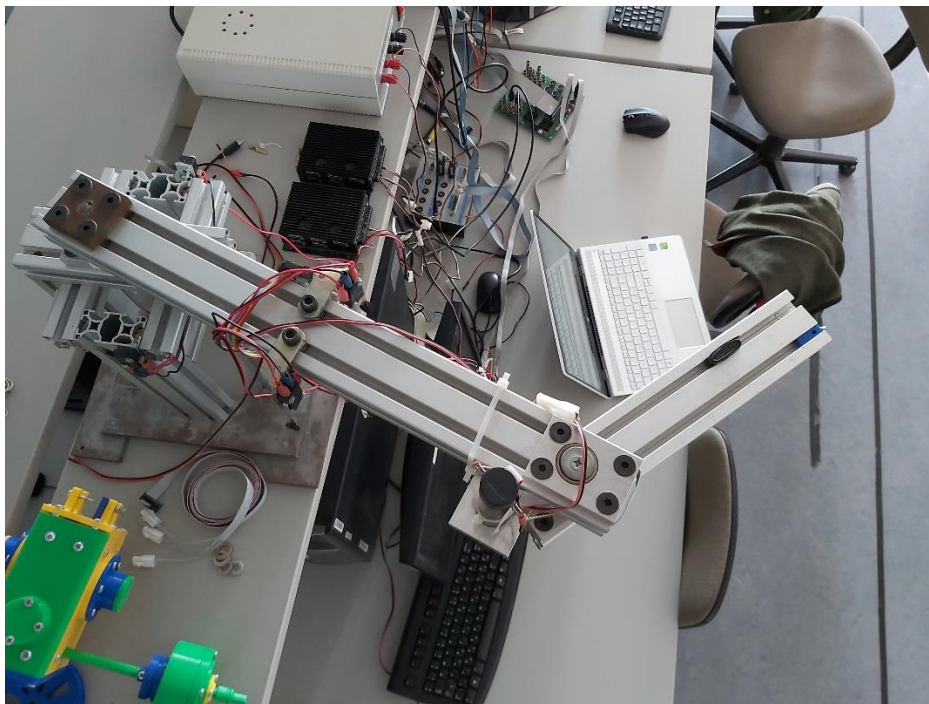
5.3 Transzformáció a konfigurációs térbe

A fenti lépések során kreált mátrix segítségével rendelkezésre állnak az akadályok helyzetei. A következő lépés az akadályok átranszformálása a csuklóváltozók (q_1, q_2) térébe. Az XY tér minden pontja, amelyen akadály található (azaz a bitmapben 0 érték szerepel), kerül leképezésre a konfigurációs térben.

A Descartes-koordinátarendszer egyes pontjai a korábban prezentáltak (ld. 3.2. fejezet) szerint átválthatóak csuklóváltozó értékekbe. Ahogy az kifejtésre került, egy ponthoz egy vagy két szögpar is tartozhat, ezért az XY teret reprezentáló bitmap egyes elemei két q_1, q_2 cellához is tartozhatnak a csuklóváltozók bitmapjében. Az akadály összes pontjához 1 vagy 2 szögpar kiszámításra kerül, az előállt szögparok a bitmap rendszerbe történő átváltást követően – ha azok nem esnek a végállásokon túlra – a konfigurációs tér bitmapjében rögzítésre kerülnek.

6 Megvalósítás fizikai hardveren

A pályatervező algoritmus a szimulációban sikeresen tesztelésre került, a végső feladat ezt az implementációt a fizikai robotkaron is tesztelni, valós akadállyal kipróbálni. A robotkar elrendezéséről, beavatkozó szerveiről és mérőberendezéseiről a korábbiakban már volt szó (ld. 1.3.1. fejezet), ebben a fejezetben röviden bemutatásra kerül a PC-n futó Simulink fejlesztőkörnyezet és a robotkar közti kommunikációt megvalósító hardver- és szoftverkomponens. Ezek után ismertetésre kerül a robotkar mozgatásához szükséges szabályozó tervezésének és hangolásának lépései, illetve a folyamat nehézségei és elért eredményei. Miután a robotkar besabályozásra került, az előző fejezetben (5.2. fejezet) tárgyalt módszerrel betanításra kerülnek az akadályok, végül pedig a robotkar különböző pozíciók közt fog mozogni ütközésmentesen.

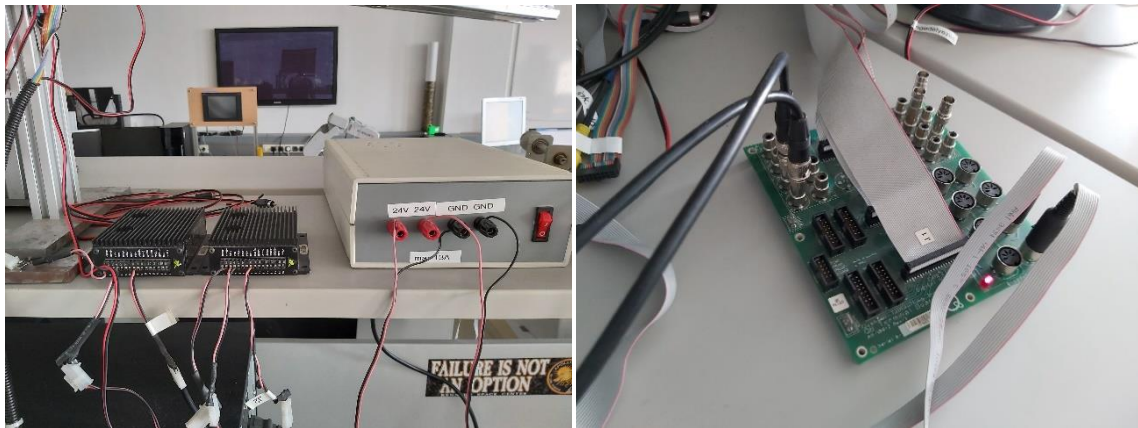


6.1. ábra – Mérési elrendezés (felülnézeti kép)

6.1 Robotkar és PC kommunikáció

A laborban található személyi számítógépen a Matlab Simulink 2012b verziója található meg, a szabályozást is ezen a platformon végeztem el. A beavatkozó szervek és a szögelfordulásmérők tulajdonságait a BME-VIK Villamosmérnöki Msc képzés Intelligens robotok és járművek laboratórium segédletből gyűjtöttem. [1]

A teljesítményelektronika (6.2. ábra) és a DC hajtások a Maxon cég eszközei. A teljesítményelektronika áramszabályozást is megvalósít, az áram a nyomatékkal arányos. Az alkalmazott Qunaser Q8 adatgyűjtő kártya az inkrementális jeladóból érkező impulzussorozatot közvetlenül képes fogadni. A jeladók két csatornásak, egy körbeforduláshoz 400 számláló impulzus tartozik, azonban a motor tengelye és a csukló tengelye közti áttétel miatt a tényleges felbontás 22800 egy teljes körül forduláshoz, ez igen pontos mérési adatok szolgáltatását teszi lehetővé. A teljesítményelektronika számára csuklónként egy (-10V) – (+10V) tartományban kiadott feszültségérték felel meg az áram alapjelnek, amelyet az adatgyűjtő kártya DA átalakítói állítanak elő. [1] A fejlesztés során valós idejű implementációra a Quarc rendszert használtam (ld. 1.1. fejezet).



6.2. ábra -- Teljesítményelektronika (b) kommunikációs portok (j)

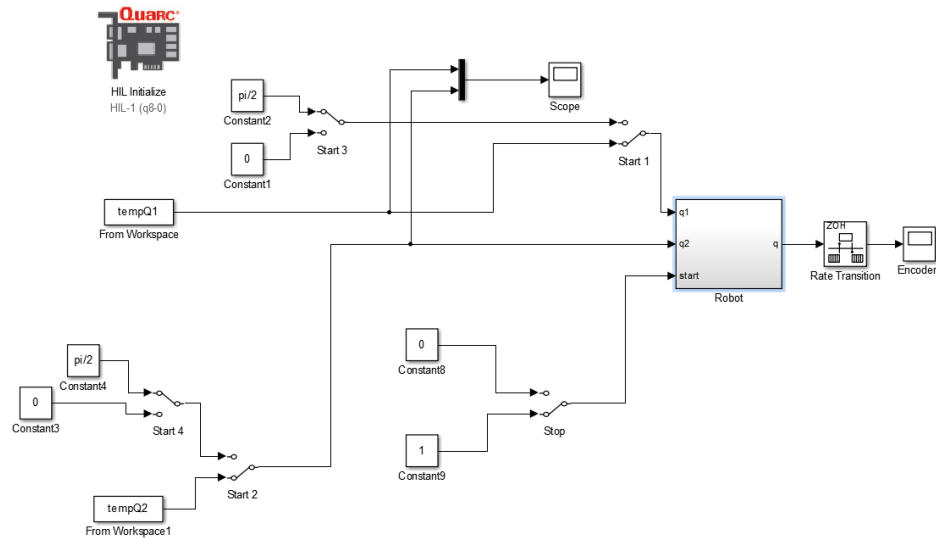
A Quarc felhasználása a Simulinkben belül rendkívül kényelmes, a szoftverben a megfelelő kommunikációs portok konfigurálása után mindössze csatlakozni kell a robothoz, és a program futtatható valós időben a célhardveren.

6.2 Simulink modell a fizikai robotkarhoz

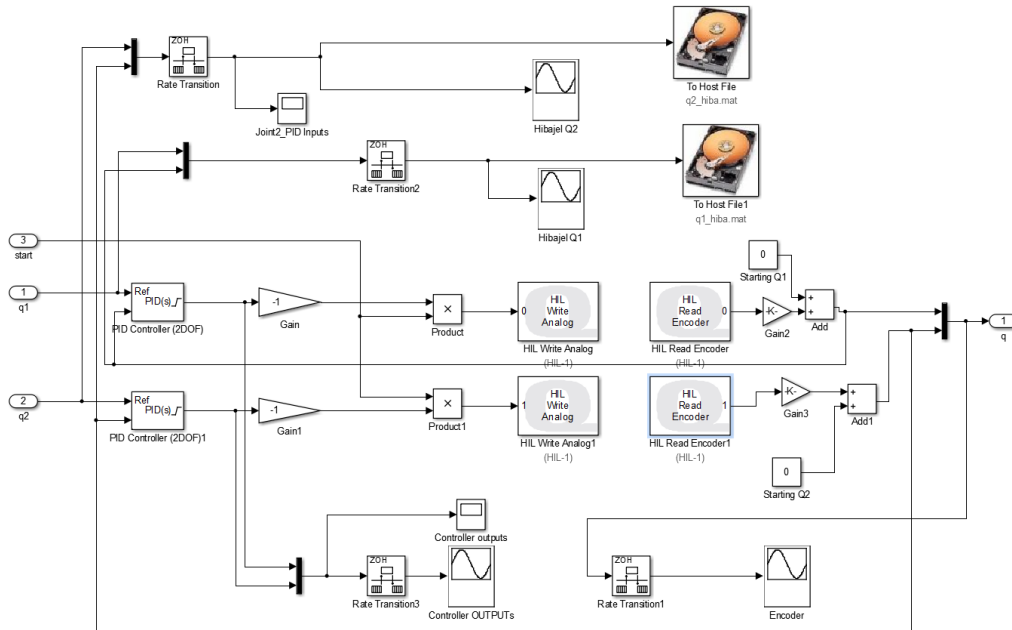
A pályatervező algoritmus futtatása után, az adott időpillanathoz tartozó csuklópozíciókat két külön változóba mentem (itt a deriváltak már nem szerepelnek), amely változókat a Simulink a *From workspace* blokkal tud elérni.

A program *Robot* nevű alrendszerében található (6.4. ábra) a csuklókhöz tartozó egy-egy szabályozó blokk (*PID Controller (2DOF)*). A beavatkozó jelet a robot felé a *HL Write Analog* blokkok küldik, az enkóderek jelének olvasását pedig a *HL Read Encoder* blokkok végzik. Utóbbi jelét át kell skálázni radiánba, a szabályozóba ezek az átskálázott jelek vannak visszacsatolva. A felszerelt szenzorok nem abszolút enkóderek,

ebből kifolyólag futtatáskor az indítási pozíciót fogják 0-nak tekinteni. Azért, hogy a robotnak lehessen kezdeti szöget állítani, az átskalázás utáni összeadó blokkok egyik bementét módosítva manuálisan lehetőség nyílik ezt megtenni. A különböző jelek megjelenítésére beépített megjelenítő segítségével van lehetőség: az alapjelen kívül, az enkóder által mért érték megjelenítése (*Hibajel Q1 és Hibajel Q2*), valamint a szabályozó kimeneti jele a legfontosabbak.



6.3. ábra – Simulink program a fizikai robothoz



6.4. ábra – Simulink „Robot” alrendszer

A korábban említett (ld. 5.2. fejezet) betanítás során a szöghelyzeteket a külső rendszerben (6.3. ábra) az *Encoders* nevű oszcilloszkópról lehet leolvasni.

6.3 PID szabályozás

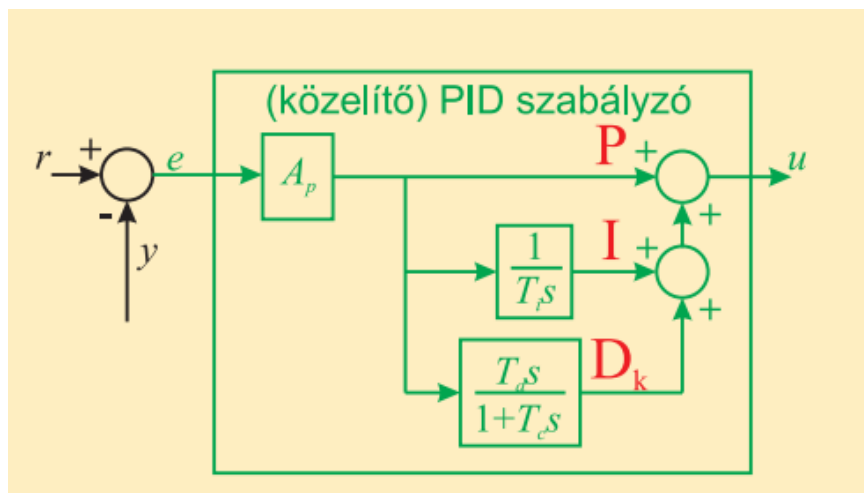
A PID szabályozó elméleti háttérét a BME-VIK Villamosmérnöki BSc képzés Szabályozástechnika tárgy előadás anyagaiból sajátítottam el. [6]

A robotkar pályakövetésének biztosításához PID szabályozót választottam. A PID az egyik leggyakrabban használt eszköz zárt rendszerek szabályozásához. A szabályozó három hatás lineáris kombinációjából tevődik össze: egy hibajellel arányos (proporcionális) tag, egy hibaintegrállal arányos (integráló) tag és egy hibaderiválttal arányos (differenciáló) tag. A szabályozó bemenete egy „e” hibajel, ami az ún. referencia jel (alapjel), és a szabályozandó jellemző, azaz az aktuálisan mért érték különbsége. Ezen bemenet függvényében a szabályozó kimeneti (végrehajtó) jelet állít elő, az aktuális hiba (P), a múltbeli hiba (I) és a hiba változásának (D) függvényében. A D-hatás valóságban nem realizálható, ezért egy ún. közelítő D-hatást kell alkalmazni. A szabályozó átviteli függvénye:

$$W_{PID} = A_p \left(1 + \frac{1}{sT_i} + \frac{sT_d}{1+sT_c} \right)$$
, ahol A_p az erősítés, T_d a deriválási idő, T_i az integrálási idő, és T_c a közelítő D-hatást megvalósító időállandó. [6]

A maximális beavatkozó jel, ugrás alakú alapjel esetén: $u_{max} = A_p \left(1 + \frac{T_d}{T_c} \right)$

A jól hangolt szabályozó gyors és pontos alapjel követést biztosít, valamint nincs maradó hibája.



6.5. ábra – PID szabályozó hatásvázlata [6]

6.4 A szabályozó hangolása

A szabályozást több tényező is nehezíti, a legfontosabbakról lesz röviden szó. Az első és a második csuklók egymástól nyilvánvalóan nem függetlenek és az egyik csukló mozgása kilengést okoz a másik csuklóban. Másik nehézség, hogy a DC motorok bizonyos kivezérelt áramszint alatt nem mozdulnak el, mert a súrlódás a mechanikus alkatrészekben nagyobb, mint a motor nyomatéka. A felépítésből adódóan, a robotot alkotó szegmensek nem homogének, a gépépítő profilokra több szenzor, végálláskapcsoló és kábel van szerelve, melyek súlya nem elhanyagolható, valamint a kábelek mozgás során könnyen felcsavarodhatnak, beakadhatnak, így különböző erővel húzzák vissza a robotkart.

A robotkarok ipari felhasználásából kiindulva, a szabályozásnak gyorsnak és pontosnak, maradó hiba nélkülinek kell lennie. Szintén fontos kritérium, hogy a túllövés mértéke minimális, illetve ideális esetben 0 legyen. Mivel a robotkar egy bizonyos nyomatékszint alatt nem mozdul meg, nagy az ún. „elintegrálódás” veszélye, amikor az I hatás folyamatosan növekedik, mert a kiadott beavatkozó jel nem elégséges az alapjel követésére. Ezt a problémát alacsony I és magas P tényezővel küszöböltem ki, ezáltal a kezdeti beavatkozó jel is kellően nagy lesz, viszont az I hatás miatt maradó hiba nem lesz a rendszerben. A folyamat gyorsításához elengedhetetlen a D hatás, különösen az első csuklónál fontos, hiszen az lomhább és nagyobb tehetetlenséggel mozog, ezért itt nagyobbra is választottam az értéket.

Mivel a rendszernek átviteli függvénye nem volt adott és automata hangolást nem lehet végezni, a hangolást sok-sok különböző mérés során finomított paraméterekkel végeztem. Habár a modellezett robotkar jelentősen eltér a valóságos robotkartól, iránymutatás céljából módosítottam a korábbi (4.4.2. fejezetben bemutatott) Simulink modellt és megvalósítottam benne a PID szabályozást (6.9. ábra), hogy ezáltal az iránymutatás által könnyebb legyen a valós rendszert paraméterezni.

A szabályozó blokkok beállításához, először a csuklók módosításait kell elvégezni: itt mindössze annyi a teendő, hogy a szolgáltatott bemenet a nyomaték legyen, a mérés pedig a szögelfordulás (6.6. ábra), ami a szabályozó visszacsatolása is egyben.

Actuation	
Torque	Provided by Input
Motion	Automatically Computed
Sensing	
Position	<input checked="" type="checkbox"/>
Velocity	<input type="checkbox"/>
Acceleration	<input type="checkbox"/>
Actuator Torque	<input type="checkbox"/>
Lower-Limit Torque	<input type="checkbox"/>
Upper-Limit Torque	<input type="checkbox"/>

6.6. ábra – Csuklók be- és kimenetei

Az előre definiált PID blokkok nagy mértékű személyre szabhatóságot adnak, mint például a kimenet szaturálását, ami a valós rendszeren különösen hasznos. E mellett különböző együtthatókat és súlyozási paramétereket is állítani lehet, melyek közül mindössze a P hatás b értékét módosítottam 0.1-ről 1-re, így kezdeti szögállás értékek is megadhatóak a rendszernek. A beállított paraméterek lent láthatóak (6.7. ábra, 6.8. ábra).

▼ Compensator formula

$$P(b \cdot r - y) + I \frac{1}{s}(r - y) + D \frac{N}{1 + N \frac{1}{s}}(c \cdot r - y)$$

Main Initialization Output Saturation Data Types State Attributes

Controller parameters

Source: internal

Proportional (P): 20

Integral (I): 0.7

Derivative (D): 25

☒ Use filtered derivative

Filter coefficient (N): 100

Setpoint weight (b): 1

Setpoint weight (c): 1

6.7. ábra – Első csukló PID szabályozó blokkja (szimuláció)

▼ Compensator formula

$$P(b \cdot r - y) + I \frac{1}{s}(r - y) + D \frac{N}{1 + N \frac{1}{s}}(c \cdot r - y)$$

Main Initialization Output Saturation Data Types State Attributes

Controller parameters

Source: internal

Proportional (P): 20

Integral (I): 3

Derivative (D): 2

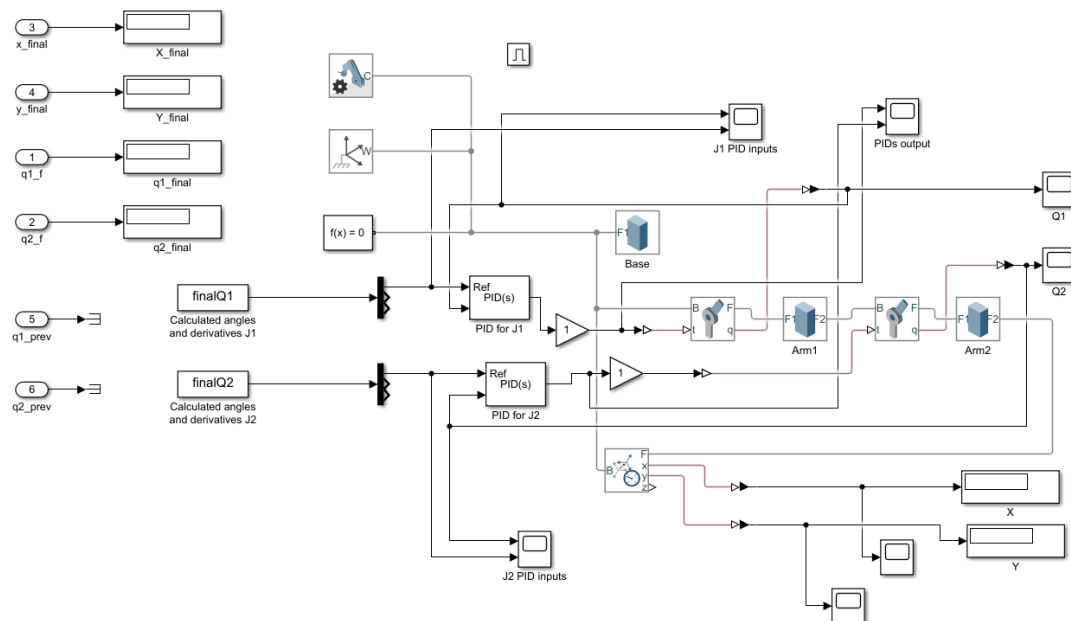
☒ Use filtered derivative

Filter coefficient (N): 100

Setpoint weight (b): 1

Setpoint weight (c): 1

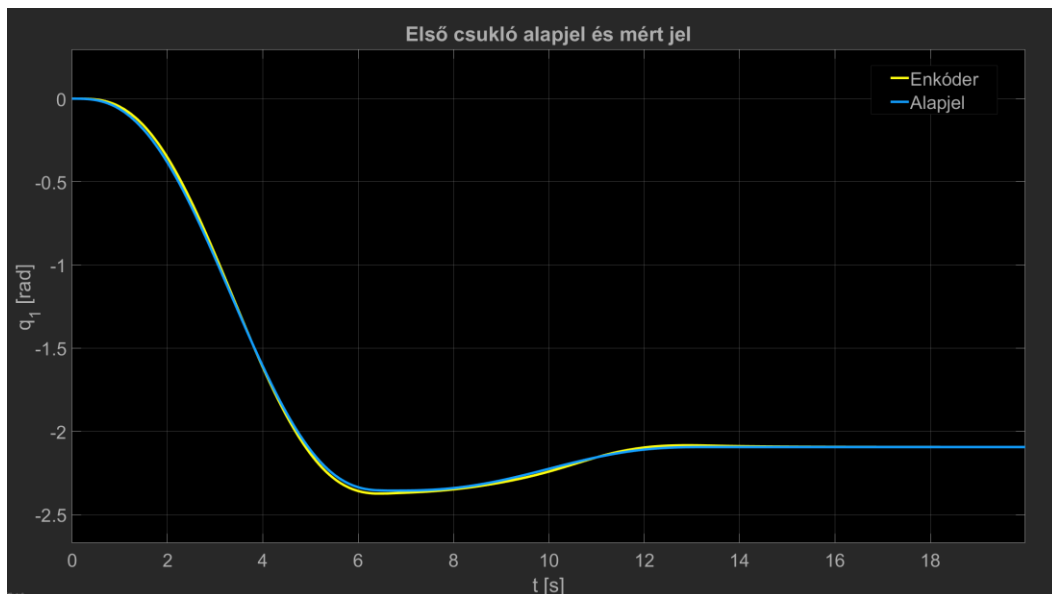
6.8. ábra – Második csukló PID szabályozó blokkja (szimuláció)



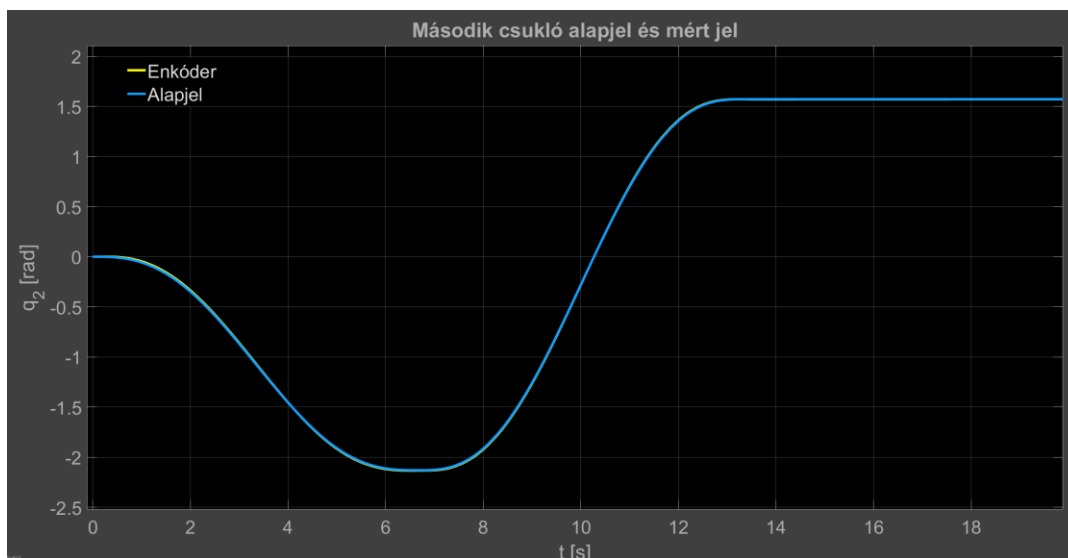
6.9. ábra – Módosított Simulink modell PID szabályozóval (szimuláció)

A szimulációban történő hangolás nagy előnye, hogy a rosszul paraméterezett szabályozó akkor sem okoz kárt a környezetében, ha a robotkar a rossz vezérlés miatt össze-vissza pörög. A szimuláció a valós rendszerben meglévő zavaró hatások közül többet nem tartalmaz, pl. a már említett súrlódást. A hangolás kifejezetten jól sikerült, egy polinomiális pálya alapjelét igen nagy pontossággal képes lekövetni a modellezett robotkar a szimulációban.

A lenti ábrán látszik, hogy a második csukló (6.11. ábra) szabályozása ilyen típusú pályákra közel tökéletes, a „mért” csuklászögek a teljes pályán követik az alapjelet, a szabályozás pontos, túllövés nincs. Ahogy az várható volt, az első csukló alapjel követése kevésbé lett jó a szimulációban (6.10. ábra), viszont bőven az elfogadható hibahatáron belül mozog. A mérési idődiagrammon leolvasható, hogy egy nagyobb alul lövés a 6. másodpercben történt, de ez egy tized radiánnál kisebb volt.



6.10. ábra – Első csukló alap- és mért jelei (szimuláció)



6.11. ábra – Második csukló alap- és mért jelei (szimuláció)

A szimuláció segítségével sikerült a valós rendszerre tervezett szabályozó paramétereinek hozzávetőleges arányát megtudni, így megkezdhető a valós robotkar felparaméterezése. A PID szabályozók kimenete nem a vezérlésre kiadott feszültségérték, az még átskálázásra kerül a szabályozó blokkon kívül és attól függetlenül.

A fizikai rendszer Simulink vezérlése korábban bemutatásra került (6.4. ábra), az abban felhasznált PID blokk funkcionalitása megegyezik a szimuláció során felhasznált blokkal (6.9. ábra). A valós rendszeren végzett tesztek során körültekintően jártam el,

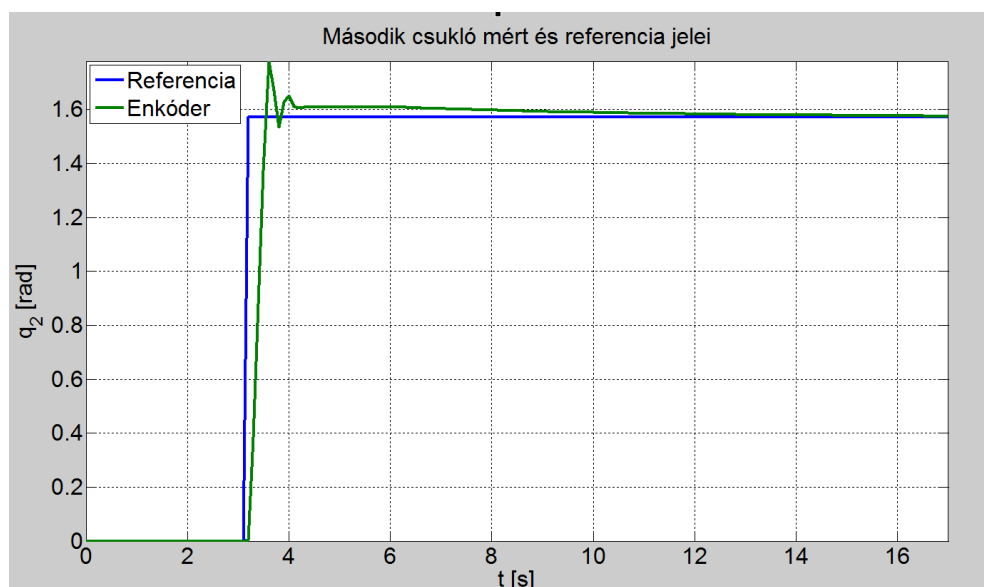
külön-külön tesztelve a két csuklót, ezáltal elkerülve, hogy a robotkar kárt tegyen magában és a környezetében.

	P	I	D
Első csukló (q_1)	39.5	0.5	1.2
Második csukló (q_2)	9.5	1.8	0.2

6.12. ábra – Végleges szabályozó paraméterek

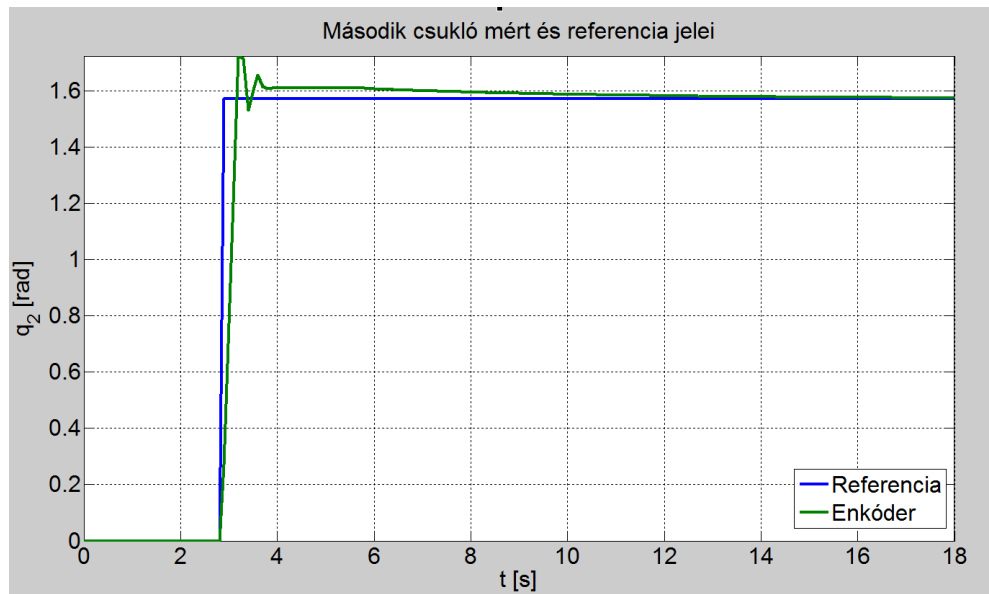
A szabályozás nagyon sikeresnek mondható a végleges paraméterekkel (6.12. ábra). Érdekesség, hogy az integráló tag kis módosítása, a többi érték változatlanul hagyása mellett is nagyon jelentősen befolyásolja a rendszer viselkedését. A beszabályozott robotkar egységugrás alapjelre is tesztelve lett, ami valós felhasználás szempontjából nem, azonban a szabályozás képességeinek felmérése szempontjából nagy jelentőségű. A tesztek a két csuklóra külön-külön végeztem, mindkettőt két külön esetben vizsgálva: egyikben a nem tesztelt csukló 0 radián értéken történő szabályozása mellett, másik esetben a szabályozást kikapcsolva.

Először a második csukló alapjelét vizsgáltam: az alapjelet hirtelen 0-ról $\frac{\pi}{2}$ -re módosítottam, az első csuklót eközben szabályozottan 0 radiánon tartottam. A grafikonon látható (6.13. ábra), hogy az enkóder által mért csuklópozíció eltérése az alapjeltől 2,5 másodperccel később 2%-nál kevesebb. Az ugrás pillanatától számítva 7 másodperc elteltével a hiba már 1% alatt van. A felfutás gyors, a végértékre való beállítás lassú.



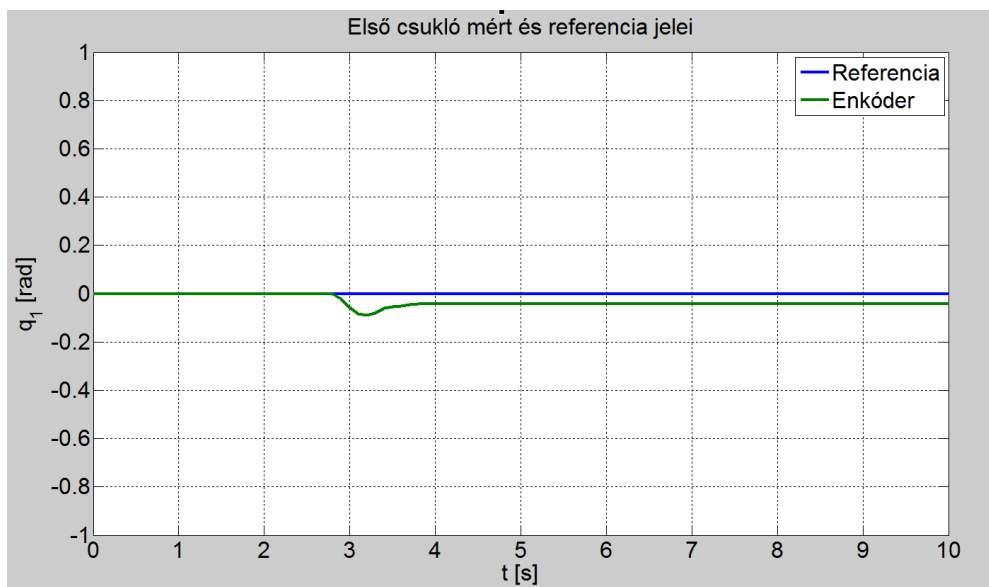
6.13. ábra – Második csukló egységugrás alapjel esetén (első csukló 0 radiánon tartva)

A túllövés mértéke 9.5%, a hullámszás kevesebb, mint negyed másodpercig tart. Amennyiben az első csukló szabályozását kikapcsoltam és a tesztet megismételtam (6.14. ábra) látható, hogy a túllövés csúcsa időben kissé ellaposodik, illetve a végértékre beállítás kissé lomhább, de nagy különbségek nem tapasztalhatóak.



6.14. ábra – Második csukló egységugrás alapjel esetén (első csukló nem szabályozva)

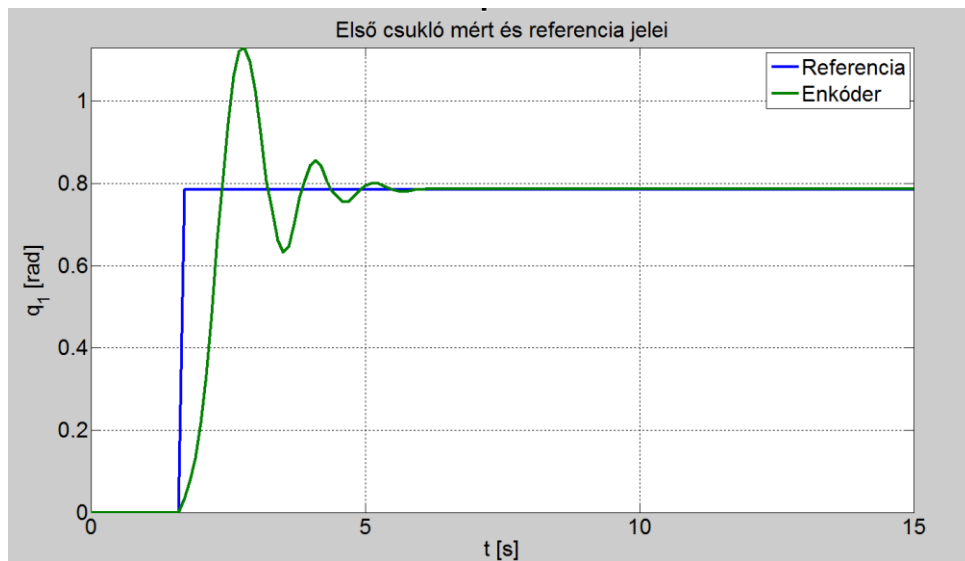
Ebben az esetben az első csukló a második csukló mozgásának hatására kissé elmozdul a kiindulási pozíciójából (6.15. ábra), hiszen a szabályozója ki van kapcsolva.



6.15. ábra – Első csukló szabályozó nélkül, második csuklót egységugrásra tesztelve

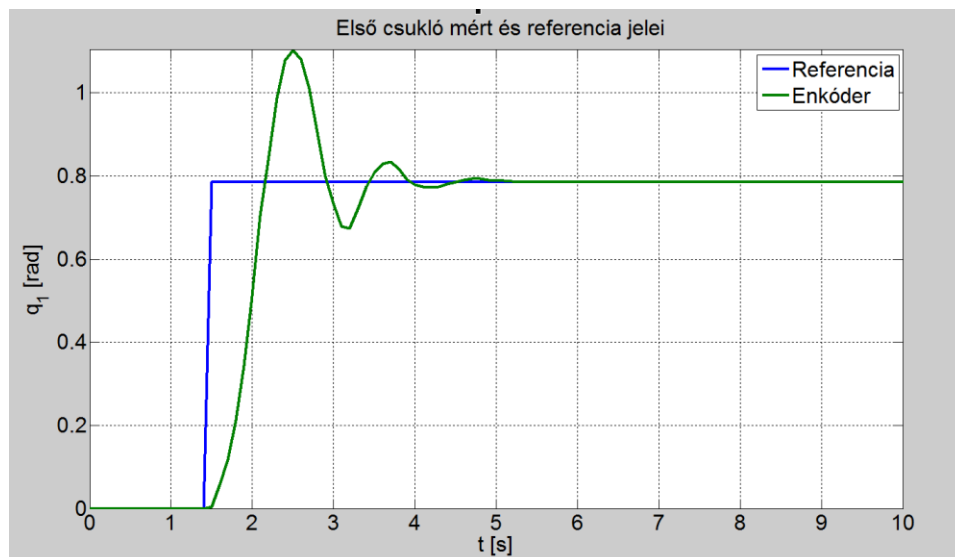
A teszteket az első csuklóra ugyanúgy elvégeztem, ebben az esetben az alapjelet 0-ról $\frac{\pi}{4}$ -re változtatva. A túllövés mértéke 29% (6.16. ábra), azonban érdekes módon a beállítás

nem lassabb, 3 másodperc alatt a hiba 2%-nál kisebb, 4 másodperc után pedig 1% alatti. A túllövés mértéke nagyon jelentős, ugrás jellegű jelekkel a robotkart a felhasználás során vezérelni nem lehet, szerencsére polinomiális pálya esetén az alapjelkövetés kitűnő.



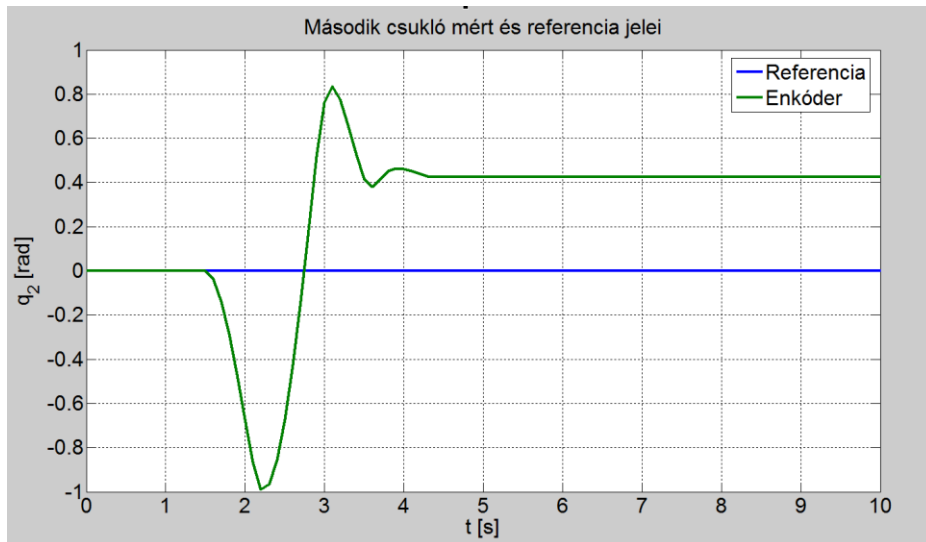
6.16. ábra – Első csukló egységugrás alapjel esetén (második csukló 0 radiánon tartva)

A tesztet megismételve úgy, hogy a második csukló szabályozását kikapcsoltam látható (6.17. ábra), hogy a beállítás főbb paramétereit elhanyagolhatóan változtak.



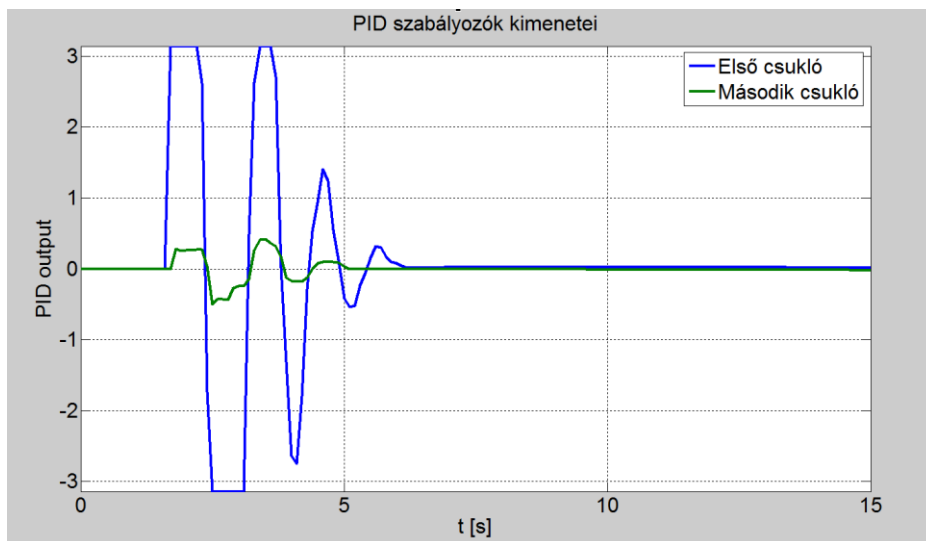
6.17. ábra – Első csukló egységugrás alapjel esetén (második csukló nem szabályozva)

Amennyiben a második csukló nem volt szabályozva az első csukló mozgása közben, a mozgás során a második szegmens – tehetetlenségéből adódóan – erőteljesen kilengett, ahogy ez az alábbi grafikonon is látszik (6.18. ábra).



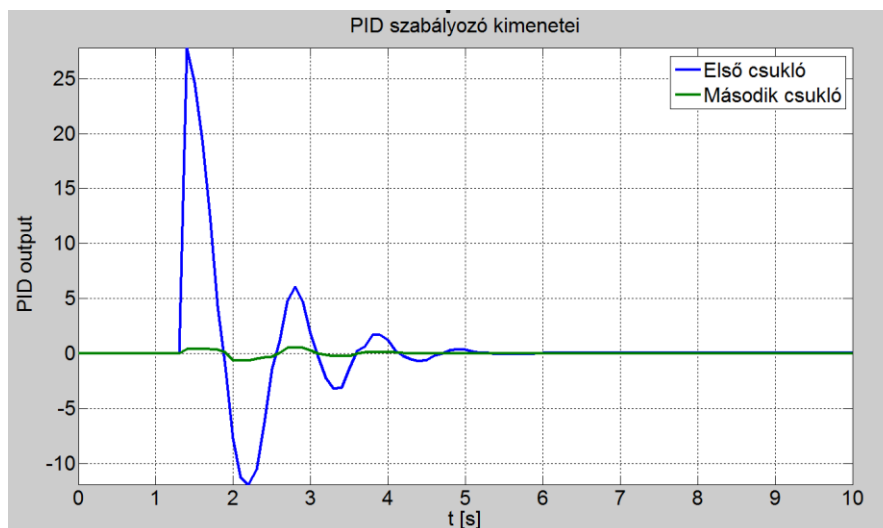
6.18. ábra – Második csukló szabályozó nélkül, első csuklót egységugrásra tesztelve

Érdemes megjegyezni, hogy a szabályozó kimenetei szaturálva vannak, aminek értéke 3.14. Az egységugrás teszteket szaturáció nélkül is elvégeztem (ezek megtekinthetők a függelékben: 8.6.1. fejezet), azonban a csuklók beállása gyakorlatilag nem változott, ezzel szemben a kiadott beavatkozó jel (és ebből kifolyólag a nyomaték) az eredetihez képest (6.19. ábra) akár nyolcszorosára is megnőtt (6.20. ábra).



6.19. ábra – A szabályozók kimenete (kimenet szaturálva, első csukló egységugrás alapjelre, második csukló 0 radiánon tartva)

Ebből kifolyólag a PID szabályozók kimeneti szaturációját megtartottam, amit a beállított 3.14 értéken felül nem volt érdemes választani. Fontos megjegyezni, hogy a PID kimenetek nem közvetlen beavatkozó jelek és nem is V mértékegységűek, ezek a jelek (azonos módon) átskálázásra kerülnek. Megjelenítésük az egymáshoz való viszonyukat és a mérési elrendezésekben tapasztalt különbségeket hivatott szemléltetni.

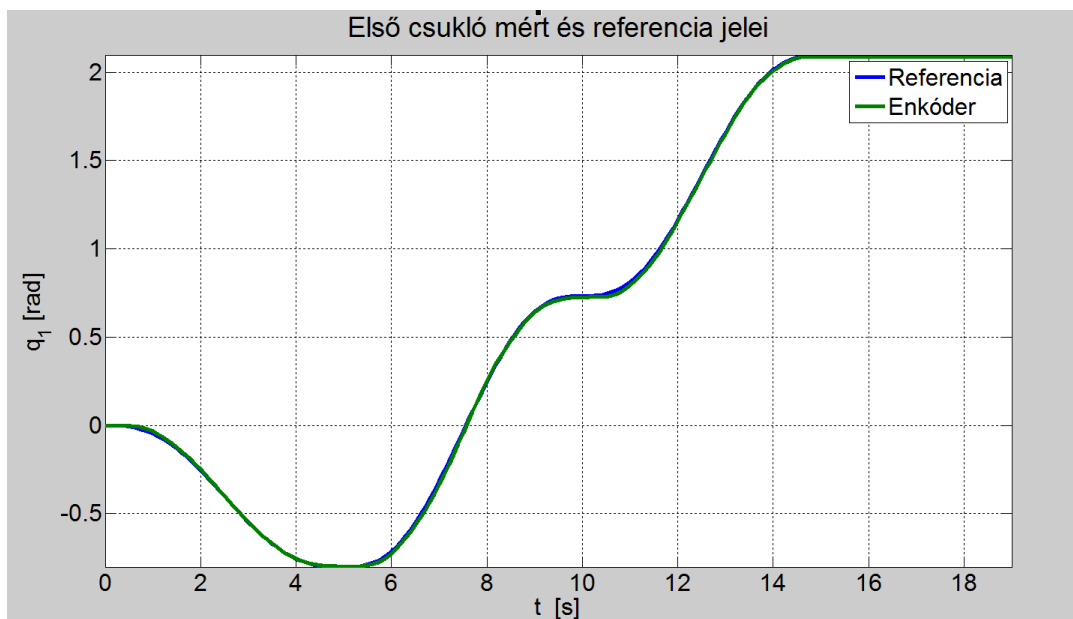


6.20. ábra – A szabályozók kimenete (kimenet nincs szaturálva, első csukló egységugrás alapjelre, második csukló 0 radiánon tartva)

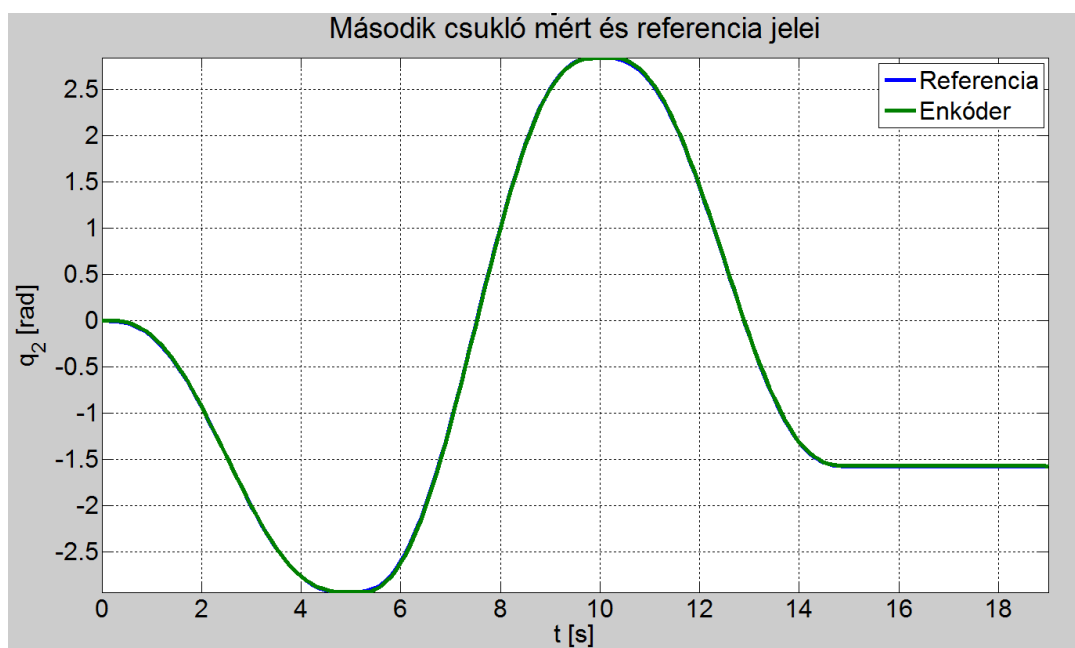
Az alkalmazás szempontjából lényegesebb adatok, hogy a polinomiális pályát miképpen tudja lekövetni a robot. Ebből a szempontból a szabályozás rendkívül sikeresnek mondható, a robotkar mozgása az elvártnak megfelelő, gyorsan és pontosan követi az elméleti pályát.

A lenti grafikonokon láthatóak az első (6.21. ábra) és a második (6.22. ábra) csuklók alapjelei és az enkóder által mért pozíciók. A két grafikon ugyanazon futás során lett rögzítve, azaz a szegmensek egyszerre mozogtak, sokszor ellentétes irányba. Mindkét csukló a 0 kezdő helyzetből indult, az első csukló $\frac{2\pi}{3}$ -ba lett mozgatva, míg a második csukló $-\frac{\pi}{2}$ -be. A szabályozás szemléltetése céljából összetett mozgásokat választottam, az akadályok a konfigurációs térben lettek megadva.

A második csukló (6.22. ábra) szögelfordulásmerője által szolgáltatott adatok teljes átfedésben vannak a matematikailag számolt pálya pontjaival. Az első csukló (6.21. ábra) a váltásoknál kissé lomhább, a nagyobb tehetetlensége miatt lassabban reagál a változásokra, azonban agresszívabb szabályozással túllövés jelenhetett volna meg a mozgásoknál, amit az akadályok kikerülése érdekében mindenképp el akartam kerülni.

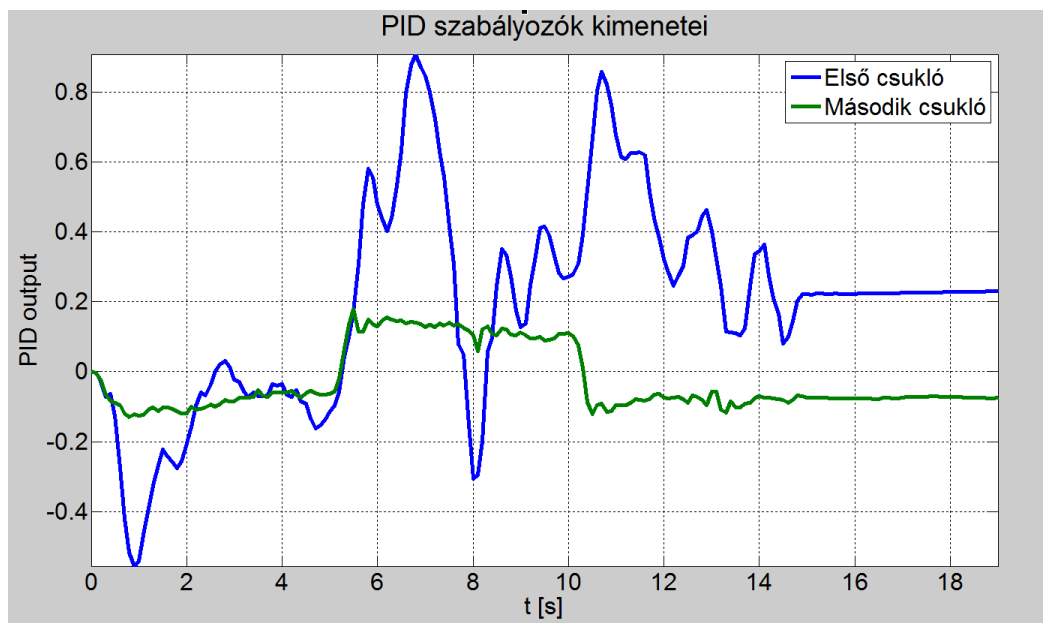


6.21. ábra – Első csukló alap- és mért jelei (valós rendszeren)



6.22. ábra – Második csukló alap- és mért jelei (valós rendszeren)

A PID szabályozók kimeneteit megvizsgálva (6.23. ábra) látható, hogy az első csuklóra adott beavatkozó jel háromszor-négyszer nagyobb intervallumban mozog, mint a második csukló jele. Az első csukló jele szintén erőteljesebb ugrásokat tartalmaz, nagyobb tüskékkel és hirtelen váltásokkal. A kimenet bőven a 3.14-es szaturációs értéken belül mozog.



6.23. ábra – A két szabályozó által szolgáltatott kimeneti jelek (relatív, mértékegység nélküli számok)

A korábbiakhoz hasonlóan itt is fennáll, hogy a grafikonon *PID output* függőleges tengely mértékegység nélküli szám, nem pedig V. Az ott látható jelek átskálázásra kerülnek ugyanakkora skálázási tényezőkkel, a grafikon kizárólag annak a szemléltetésére szolgál, hogy a két csukló beavatkozó jele hogyan viszonyul egymáshoz. További elrendezések mérési eredményei megtalálhatóak a függelékben (ld. 8.6. fejezet).

7 Akadálykerülő mozgás a valós robotkarral

Végezetül, minthogy a szabályozás megbízhatóan működik, az ütközésmentes pálya tesztelése maradt hátra a valós robotkaron, ténylegesen jelen lévő és betanított akadállyal. Ebben a fejezetben ezen tesztek eredménye lesz ismertetve.

7.1 Tesztelés

Az akadály (jelen esetben egy műanyag flakon) betanítása a korábban bemutatott módon történik (ld. 5.2. fejezet). Ezt követően a *main.m* scriptben beírhatóak a kezdeti és véghelyzeti szögek (lehetne koordinátákat is megadni), majd futtatható a script. A *probe.m* scriptet futtatva a kalkulált pálya a Simulink rendszer számára elérhető változókat feltölti, ezek után a Simulinket fordítani kell. Ha ez megtörtént a robothoz lehet csatlakozni (fontos, hogy a robotkar a kezdeti szögnek megadott pozícióban álljon), majd a futtatás gombra kattintva megindul a teszt.

A videón dokumentált kezdeti-, és a mozgás végeztével elért szögek:

	Kezdeti szög	Elérni kívánt szög
Első csukló (q_1)	0	$\frac{2\pi}{3}$ (= 120°)
Második csukló (q_2)	0	$-\frac{\pi}{2}$ (= -90°)

A mozgásról, 3 különböző szögből, 3 különböző mérés során, de ugyanazon a pályán, az alábbi videók készültek (internetkapcsolat szükséges, YouTube):

Előlről nézve: <https://youtu.be/FjE2pI4V9Lo>

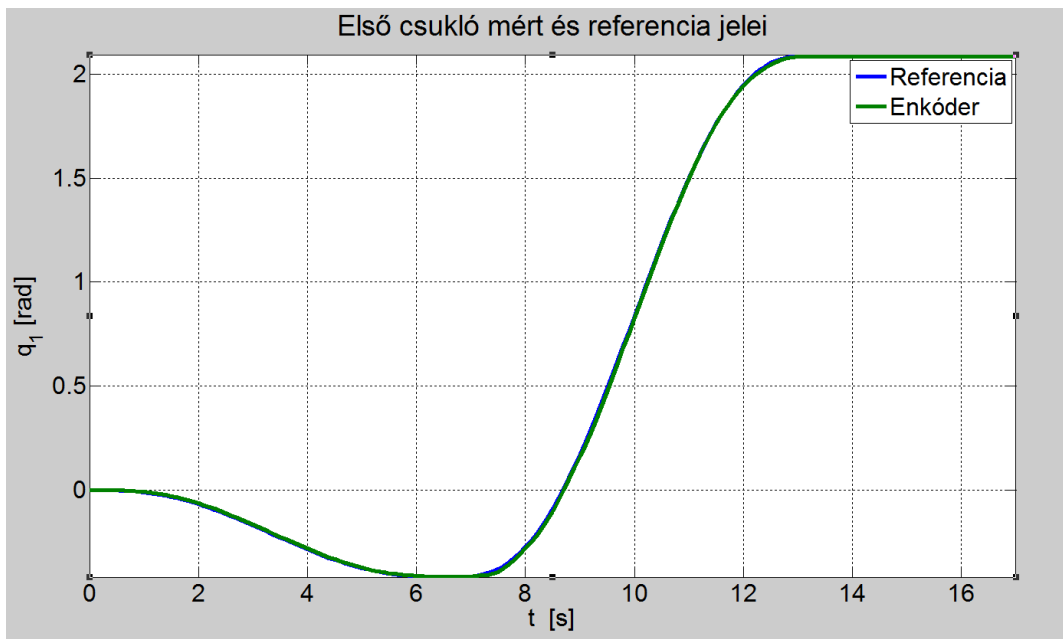
Felülnézetből: <https://youtu.be/UDC0zLCx2eE>

Hátulról nézve: https://youtu.be/rsxeDaZ_gIQ

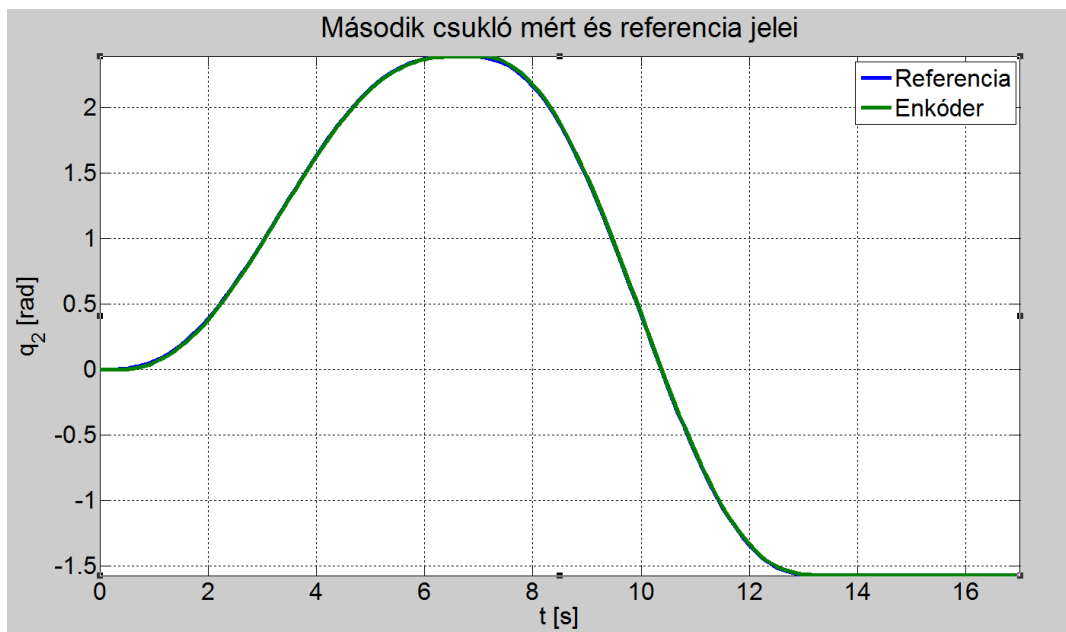
A videókon látható, hogy a második csukló közel végállásba fordul, miközben az első csukló az ellenkező irányba halad, majd mikor ez megtörtént, már túl tud haladni az akadályon, majd beállnak az elvárt pozícióba.

7.2 Eredmények analízálása

A pályatervező algoritmus az előző alfejezetben (7.1. fejezet) bemutatott esetben rendkívül optimális pályát talált, a mozgás 2 részből áll, azaz mindössze egy köztes pont beiktatásával ki tudta kerülni az akadályt. Fontos megemlíteni a korábban kifejtett gondolatot (ld. 4.4.3. fejezet), miszerint az újbóli futtatások során más és más fa épül fel, illetve a bejárás is véletlen sétáláson alapul, ennek következtében két egymást követő futtatás nem biztosan ugyanazt a pályát adja meg.



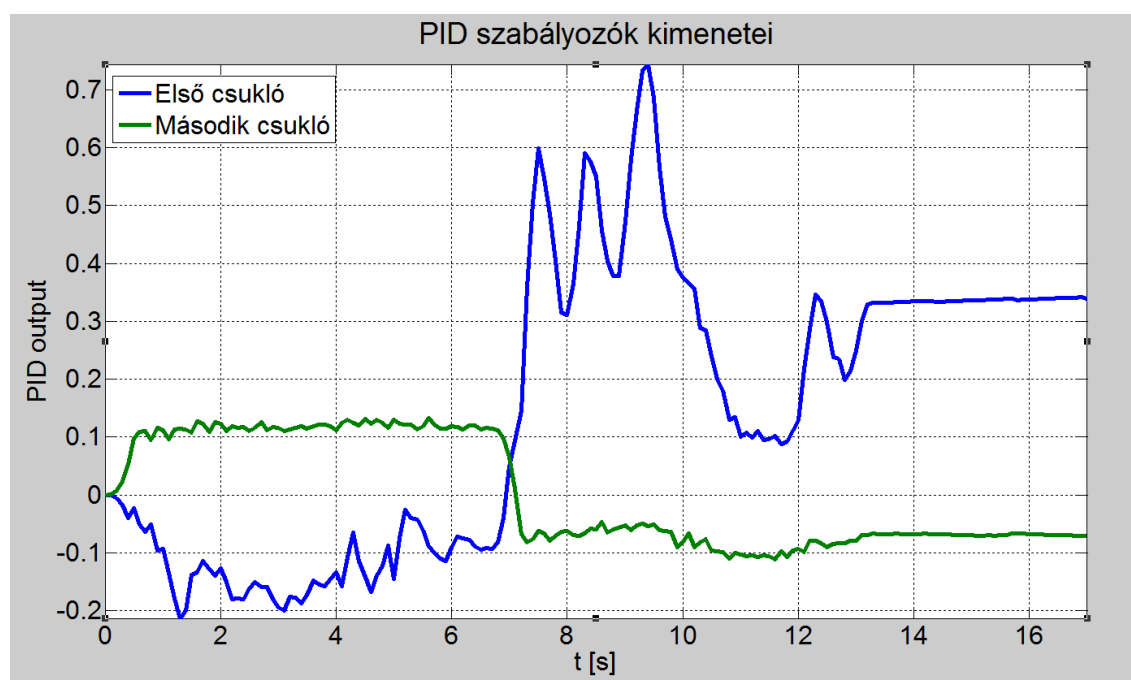
7.1. ábra – Első csukló alap- és mért jelei (akadálykerülés)



7.2. ábra – Második csukló alap- és mért jelei (akadálykerülés)

A fenti grafikonokon megfigyelhető (7.1. ábra és 7.2. ábra), ami a videókon (különösen a felülnézetin) is jól látszott, hogy az akadály helyzetéből adódóan, a második csukló nem tud teljesen nyitva lenni az első csukló kb. 0.1 ($\sim 5^\circ$) és 0.52 ($\sim 30^\circ$) közötti állásaiban, ezért, hogy az akadályt kikerülje, a második csuklónak legalább derékszögben vagy annál kisebb szögben kell állnia, amíg az első csukló ezen a szakaszon túlhalad.

A mozgás elején az első csukló kissé negatív irányba mozog, annak érdekében, hogy a második csukló be tudjon fordulni az akadály előtt. Természetesen többféle útvonal is létezik, az algoritmus a legkisebb utat választja ki a megtaláltak közül, a gráfbejárást követően. A módszerben benne van a bizonytalanság, de ugyanazon fát bejárva a megtalált út jellemzően ugyanaz.



7.3. ábra – A két szabályozó által szolgáltatott kimeneti jelek (relatív, mértékegység nélküli számok)

További videók különböző elrendezésekben a függelékben találhatóak (ld. 8.10. fejezet), ugyanúgy a függelékben további grafikon tekinthető meg az akadálykerülésről (ld. 8.8. fejezet).

8 Összegzés

A feladat szerint a fizikai robotkart lemodelleztem és hozzá mozgási egyenleteket számoltam. Ezek után megvalósítottam az akadálykerülő algoritmust. A valós rendszer szabályozása után az elkészített algoritmust és mozgási egyenleteket teszteltem a valós rendszeren, az akadályokat XY koordináta-rendszerben adtam meg, melyek a konfigurációs térbe kerültek átranzformálásra.

8.1 Eredmények értékelése

Az elkészített modell sokat segített a valós rendszer vizualizálásában, alapvető adottságaik kellően megegyeznek, a fejlesztés minden szakaszát jelentősen megkönnyítette az elkészített szimuláció. Az ütközésmentes algoritmus megbízhatóan felépít egy térképet, annak bejárása egyszerűbb akadályok esetén optimálisan történik, összetettebb akadályok és ún. csapdák esetén megmutatkoznak a sztochasztikus alapokon nyugvó módszer hátrányai.

A mérési eredmények és a tesztek alapján a robotkar besabályozása várakozáson felül sikerült, összetett, gyors mozgásokhoz is megfelelő, ugrásokra is tűréshatáron belül reagál, ötödfokú polinommal leírt pályát viszont túllövés nélkül és minimális hibával követ.

8.2 Továbbifejlesztési lehetőségek

Az ütközésmentesség tökéletességének vizsgálatával ezen szakdolgozat nem foglalkozik. Az akadályok megadása kizárólag a végberendezés pozíciójára érvényes, ebből kifolyólag az nincs vizsgálva, hogy a szegmensek ütköznek-e akadályba. Természetesen ilyen akadály felvehető: mikor a végberendezés „elférne” az akadály mögött, azonban a szegmens a síkbeli mozgásból adódóan értelemyszerűen nekiütközik annak. Ezek alapján a probléma megoldható azzal, hogy az akadályokat „kiterjesztem”, azaz a betanított akadályokat XY síkba megnövesztem, így az általuk „kitakart” területet is akadályként reprezentálom. Ezzel a megoldással a legnagyobb probléma az, hogy egyébként elérhető területek esnek ki, hiszen akadályok mögötti tartomány bizonyos szöghelyzetekkel általában elérhetőek. Másik, előremutató és elegánsabb megoldás, hogy a teljes szegmens láncolatra vizsgálom az ütközést, ezáltal az akadályok „mögötti” területek ugyanúgy elérhetőek maradnak. Ez 6 szabadságfokú robotkaroknál különösen

hasznos, hiszen ott a robotkar végberendezése nem csak síkban mozog, ezért egy pozíció sokkal több irányból megközelíthető.

Egy közvetlen fejlesztési lehetőség a különböző egységek és funkciók beágyazása egy felhasználóbarát platform alá, amelynek irányítása hozzáértést nem igényel és intuitív. A felületen lehetőség lenne akadályok egyszerű, egy gombos betanítására, ezen felül lehetőség lenne pontok felvételére, melyek között a robotkar mozoghatna.

Hosszabb távú fejlesztés az előzőkhez kapcsolódóan, a robotkarra egy Z irányba elmozdulni képes szerszám szerelése, a robot különböző szekvenciák sorozatát teljesítené, mely kérések egy PLC felől érkeznének és a robotkar egy gyártásfolyamatba dolgozna bele. Ezen kívül komplexebb akadályok megadása, ezeknek akár kamerás felismerése, a mozgási pontok kamerás pozícionálása.

A feladatkör kibővítését egy hat szabadságfokú robotkarra való áttérés jelentené, így a pályatervezés és mozgás egy térbeli feladat lenne, jelentősen bonyolultabb megvalósításokkal és matematikával.

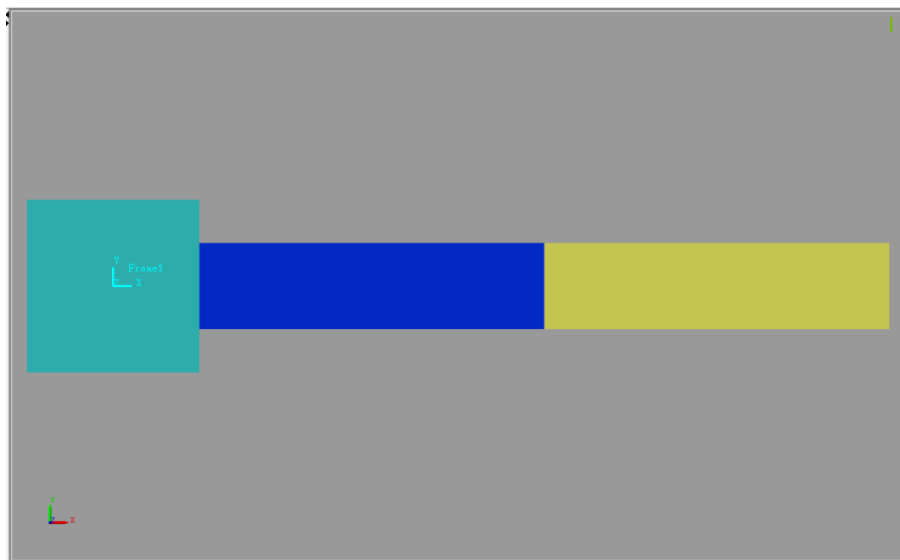
Irodalomjegyzék

- [1] Dr. Kiss Bálint: Robotkar inverz geometriája (és irányítása). Mérési útmutató, Intelligens robotok és járművek laboratórium, 2016
- [2] Gincsainé Dr. Szádeczky-Kardoss Emese: Robotkar inverz geometriája mérés. Mérési útmutató távoktatáshoz, 2020
- [3] Harmati István: Ütközésmentes pályatervezési algoritmusok mobilis robotokhoz. Segédlet az „Autonóm robotok és járművek” tárgyhoz, 2008
- [4] Prof. Peter Corke, QUT Robot Academy: Inverse Kinematics for a 2-Joint Robot Arm Using Geometry, <https://robotacademy.net.au/lesson/inverse-kinematics-for-a-2-joint-robot-arm-using-geometry/> (revision 16:24, 11 November 2020)
- [5] AmBrSoft Calculators+: Circle defined by 3 Points, <http://www.ambrsoft.com/trigocalc/circle3d.htm> (revision 17:52, 15 November 2020)
- [6] Dr. habil. Harmati István: VIIIAB05 – Szabályozástechnika, Stabilitás, Soros kompenzátorok, Bsc képzés, előadás diasor 2020. február 28.
- [7] MathWorks®: Matlab® és Simulink® termékek https://www.mathworks.com/products.html?s_tid=gn_ps (rev. 17:19, 03 December 2020)
- [8] Quanser: Quarc valós idejű irányítórendszer gyártói honlapja <https://www.quanser.com/products/quarc-real-time-control-software/> (rev. 17:23, 03 December 2020)
- [9] Fanuc: Scara robotok, termékismertető, a gyártó honlapja <https://www.fanuc.eu/hu/hu/robotok/robotasz%C5%B1r%C5%91-lap/scara-series> (rev. 17:08, 08 December 2020)

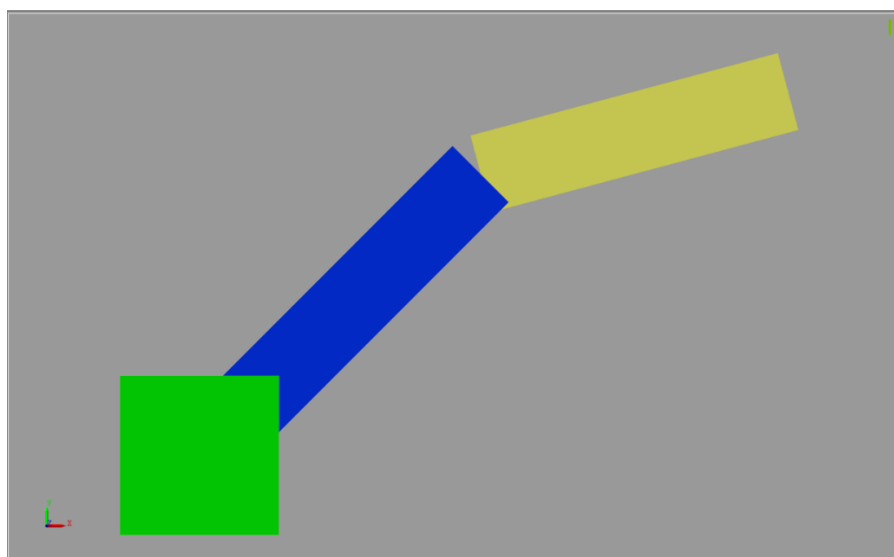
Függelék

8.3 Simscape modell felülnézeti képei

A fizikai robotkar hossza kissé eltér a szimulációban megvalósítottól, előbbi első szegmense 45 cm hosszú, második szegmense pedig 30 cm, ezzel szemben a modell 50 és 40 centis szegmenseket használ. Másik jelentős különbség, hogy míg a szimulációban a forgástengelyek a szegmensek végeinél helyezkednek el, addig a valós roboton mindkét szegmens túlnyúlik ezen.



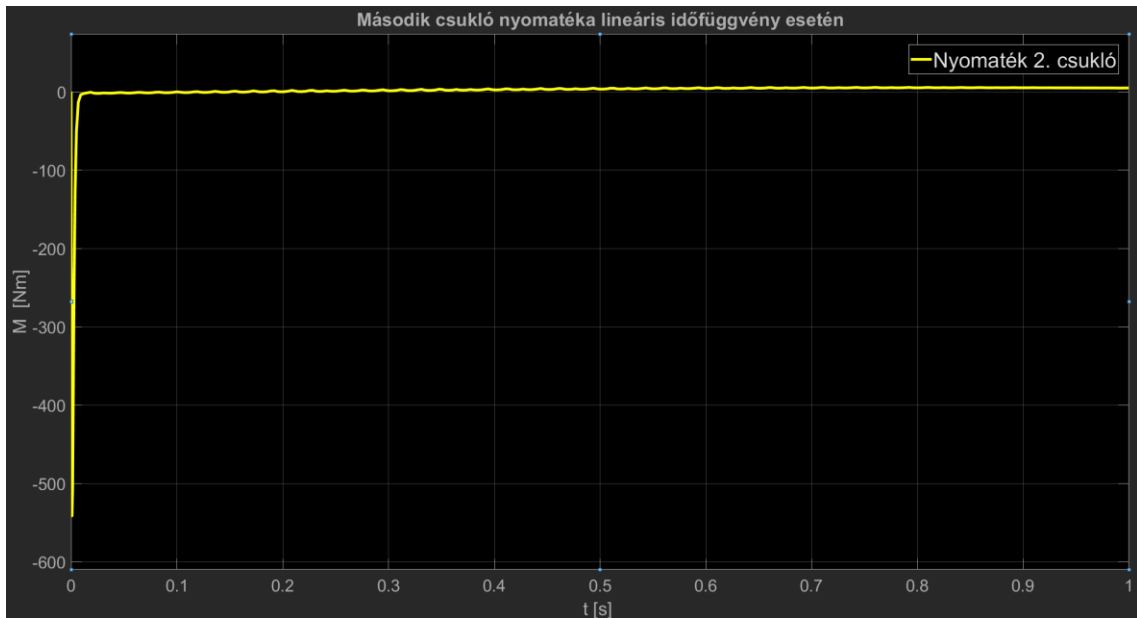
8.1. ábra – A modell felülnézeti képe ($q_1, q_2 = 0$)



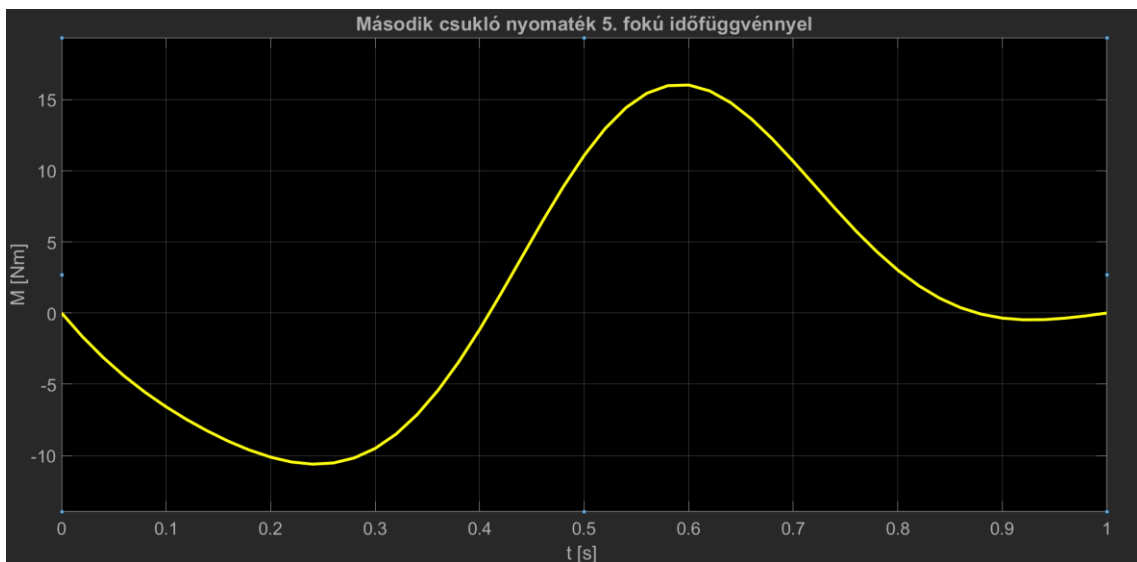
8.2. ábra – A modell felülnézeti képe ($q_1 = 45^\circ$ és $q_2 = -30^\circ$)

8.4 Második csukló nyomaték görbéi

Az első csuklóhoz hasonlóan, lineáris pályát számolva a kezdeti pillanatban hatalmas beavatkozó nyomaték kell. Ezt magasabb fokszámú polinomokkal ki lehet küszöbölni.



8.3. ábra – Második csukló nyomaték görbe 1. fokú időfüggvénnyel



8.4. ábra – Második csukló nyomaték görbe 5. fokú időfüggvénnyel

8.5 Polinomiális pálya mátrixa

Láthatóak az első és második csuklóhoz tartozó jelek és azoknak 1. és 2. deriváltjai, melyek kiszámítása és elvi háttere a 3.3.1. fejezetben kerültek bemutatásra. A mátrix első oszlopa az idő, mely jelen esetben minden sorral 10^{-3} másodperccel halad előre; a mozgási egyenletbe ezt az időt behelyettesítve megkapjuk a második oszlopot, ami az aktuális csuklópozíció; a harmadik és negyedik oszlopok a mozgási egyenlet első és második deriváltjai. A 10 000 sort tartalmazó mátrix első 30 sora látható, mely a 4.4.3. fejezetben bemutatott teszthez tartozik.

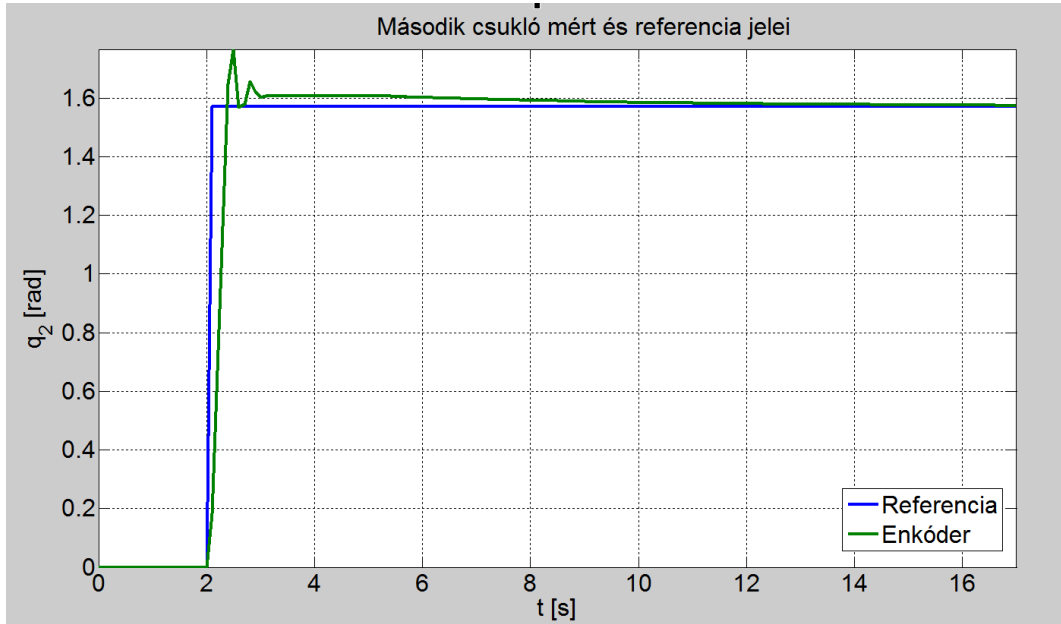
10000x4 double					
	1	2	3	4	5
1	0	2.3562	0	0	
2	1.0000e-03	2.3562	-1.6742e-06	-0.0033	
3	0.0020	2.3562	-6.6913e-06	-0.0067	
4	0.0030	2.3562	-1.5043e-05	-0.0100	
5	0.0040	2.3562	-2.6723e-05	-0.0133	
6	0.0050	2.3562	-4.1721e-05	-0.0167	
7	0.0060	2.3562	-6.0029e-05	-0.0200	
8	0.0070	2.3562	-8.1641e-05	-0.0233	
9	0.0080	2.3562	-1.0655e-04	-0.0266	
10	0.0090	2.3562	-1.3474e-04	-0.0298	
11	0.0100	2.3562	-1.6621e-04	-0.0331	
12	0.0110	2.3562	-2.0096e-04	-0.0364	
13	0.0120	2.3562	-2.3896e-04	-0.0396	
14	0.0130	2.3562	-2.8022e-04	-0.0429	
15	0.0140	2.3562	-3.2473e-04	-0.0461	
16	0.0150	2.3562	-3.7248e-04	-0.0494	
17	0.0160	2.3562	-4.2346e-04	-0.0526	
18	0.0170	2.3562	-4.7766e-04	-0.0558	
19	0.0180	2.3562	-5.3508e-04	-0.0590	
20	0.0190	2.3562	-5.9570e-04	-0.0622	
21	0.0200	2.3562	-6.5953e-04	-0.0654	
22	0.0210	2.3562	-7.2654e-04	-0.0686	
23	0.0220	2.3562	-7.9674e-04	-0.0718	
24	0.0230	2.3562	-8.7011e-04	-0.0750	
25	0.0240	2.3562	-9.4666e-04	-0.0781	
26	0.0250	2.3562	-0.0010	-0.0813	
27	0.0260	2.3562	-0.0011	-0.0844	
28	0.0270	2.3562	-0.0012	-0.0876	
29	0.0280	2.3562	-0.0013	-0.0907	
30	0.0290	2.3562	-0.0014	-0.0938	

10000x4 double				
	1	2	3	4
1	0	1.5708	0	0
2	1.0000e-03	1.5708	-7.5338e-06	-0.0151
3	0.0020	1.5708	-3.0111e-05	-0.0301
4	0.0030	1.5708	-6.7696e-05	-0.0451
5	0.0040	1.5708	-1.2025e-04	-0.0600
6	0.0050	1.5708	-1.8774e-04	-0.0749
7	0.0060	1.5708	-2.7013e-04	-0.0898
8	0.0070	1.5708	-3.6739e-04	-0.1047
9	0.0080	1.5708	-4.7947e-04	-0.1195
10	0.0090	1.5708	-6.0634e-04	-0.1343
11	0.0100	1.5708	-7.4796e-04	-0.1490
12	0.0110	1.5708	-9.0431e-04	-0.1637
13	0.0120	1.5708	-0.0011	-0.1784
14	0.0130	1.5708	-0.0013	-0.1930
15	0.0140	1.5708	-0.0015	-0.2076
16	0.0150	1.5708	-0.0017	-0.2221
17	0.0160	1.5708	-0.0019	-0.2367
18	0.0170	1.5708	-0.0021	-0.2511
19	0.0180	1.5708	-0.0024	-0.2656
20	0.0190	1.5708	-0.0027	-0.2800
21	0.0200	1.5708	-0.0030	-0.2944
22	0.0210	1.5708	-0.0033	-0.3087
23	0.0220	1.5708	-0.0036	-0.3230
24	0.0230	1.5708	-0.0039	-0.3373
25	0.0240	1.5708	-0.0043	-0.3516
26	0.0250	1.5708	-0.0046	-0.3658
27	0.0260	1.5708	-0.0050	-0.3799
28	0.0270	1.5707	-0.0054	-0.3941
29	0.0280	1.5707	-0.0058	-0.4081
30	0.0290	1.5707	-0.0062	-0.4222

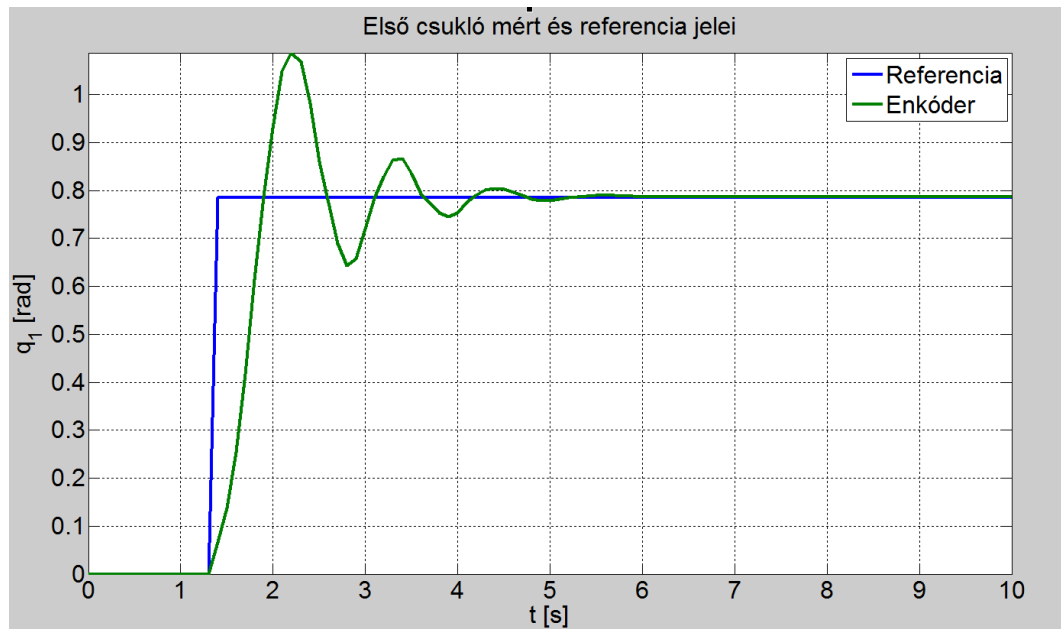
8.5. ábra – finalQ1 (b) és finalQ2 (j)

8.6 Fizikai robotkar szabályozásához kapcsolódó ábrák

8.6.1 Ugrás alapú jelek, szaturáció nélkül



8.6. ábra – Második csukló egységugrás alapjel esetén (nincs szaturáció, első csukló 0 radián értéken tartva)



8.7. ábra – Első csukló egységugrás alapjel esetén (nincs szaturáció, második csukló 0 radián értéken tartva)

8.6.2 Végleges szabályozó paraméterek

Function Block Parameters: PID Controller (2DOF)

PID Controller (2DOF)

This block implements continuous- and discrete-time PID control algorithms with setpoint weighting and includes advanced features such as anti-windup, external reset, and signal tracking. You can tune the PID gains automatically using the "Tune..." button (requires Simulink Control Design).

Controller: PID Form: Parallel

Time domain:

☒ Continuous-time
☐ Discrete-time

Main PID Advanced Data Types State Attributes

Controller parameters

Proportional (P): 39.5 [Compensator formula](#)

Integral (I): 0.5

Derivative (D): 1.2

Filter coefficient (N): 100

Setpoint weight (b): 1

Setpoint weight (c): 1

Tune...

Initial conditions

Source: internal

Integrator: 0

Filter: 0

External reset: none

☐ Ignore reset when linearizing
☒ Enable zero-crossing detection

OK Cancel Help Apply

8.8. ábra – Első csukló PID paraméterei (valós robotkar)

Function Block Parameters: PID Controller (2DOF)

PID Controller (2DOF)

This block implements continuous- and discrete-time PID control algorithms with setpoint weighting and includes advanced features such as anti-windup, external reset, and signal tracking. You can tune the PID gains automatically using the "Tune..." button (requires Simulink Control Design).

Controller: PID Form: Parallel

Time domain:

☒ Continuous-time
☐ Discrete-time

Main PID Advanced Data Types State Attributes

Controller parameters

Proportional (P): 9.5 [Compensator formula](#)

Integral (I): 1.8

Derivative (D): 0.2

Filter coefficient (N): 100

Setpoint weight (b): 1

Setpoint weight (c): 1

Tune...

Initial conditions

Source: internal

Integrator: 0

Filter: 0

External reset: none

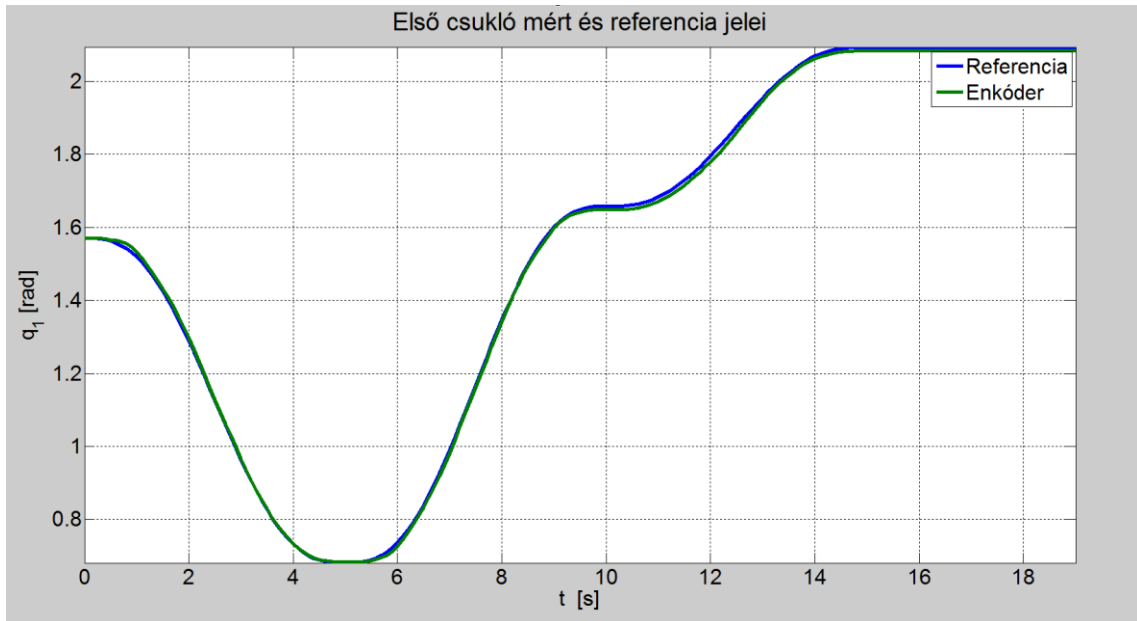
☐ Ignore reset when linearizing
☒ Enable zero-crossing detection

OK Cancel Help Apply

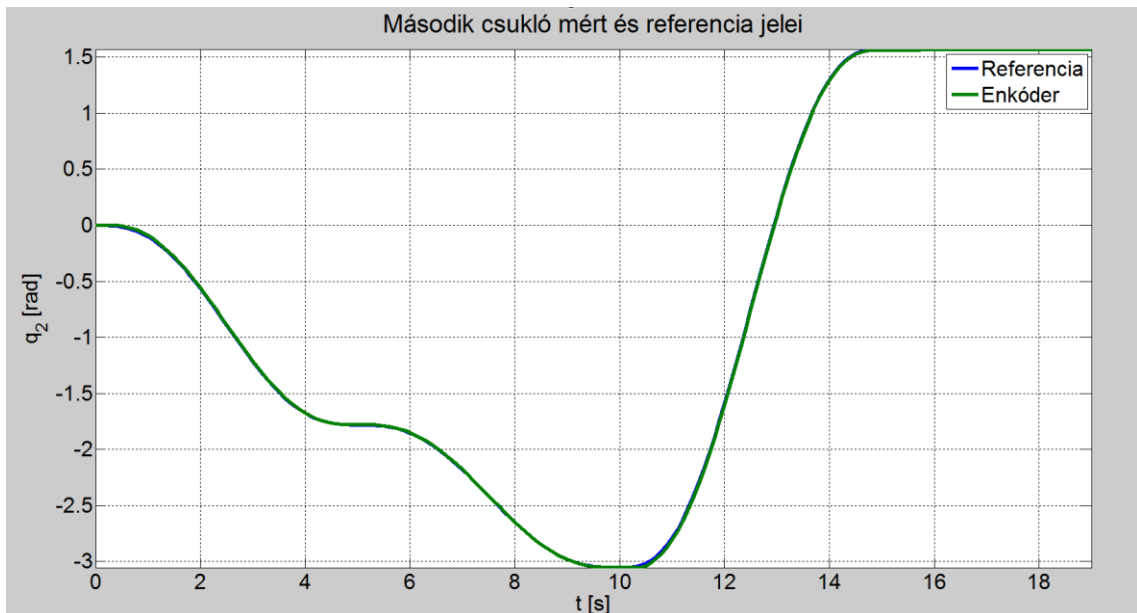
8.9. ábra – Második csukló PID paraméterei (valós robotkar)

8.7 Első csukló nem 0 kezdeti szögből indítva

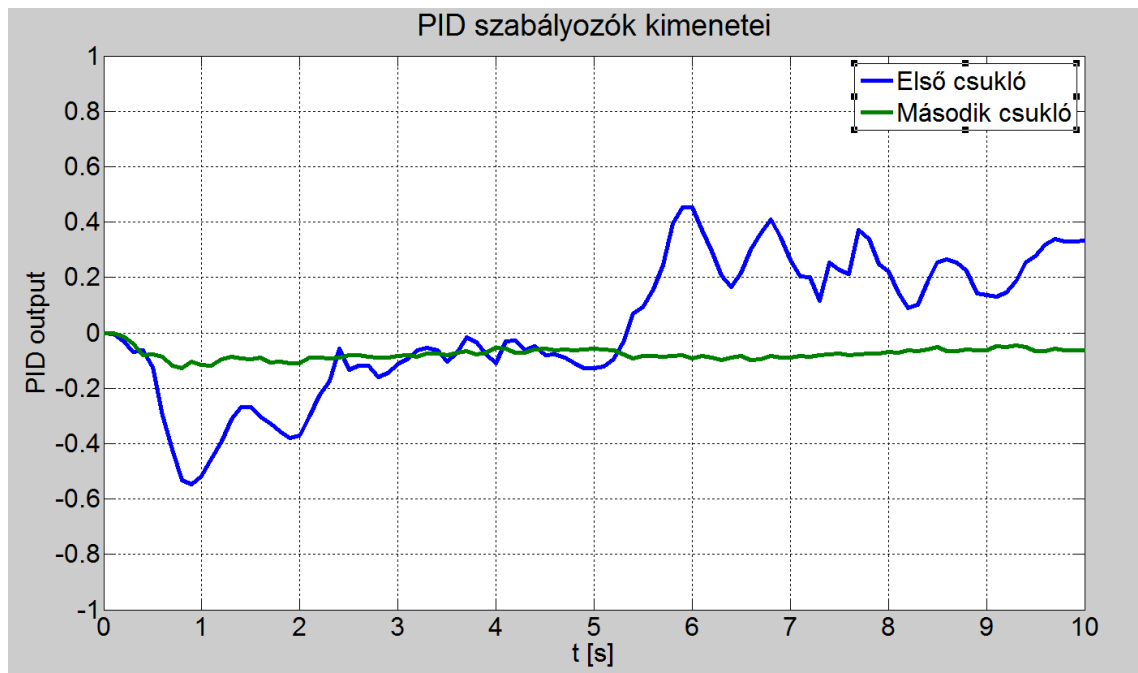
A robotkar mindkét csuklója tetszőleges kezdeti szögből indítható, feltéve, hogy a Simulink rendszerben a kezdeti konstans értékek át lettek írva (6.4. ábra). Ezáltal egy valós felhasználás során bármely két pont közt képes ütközésmentes pályán végig mozogni.



8.10. ábra – Első csukló alap- és mért jelei ($\pi/2$ kiindulási szögből)



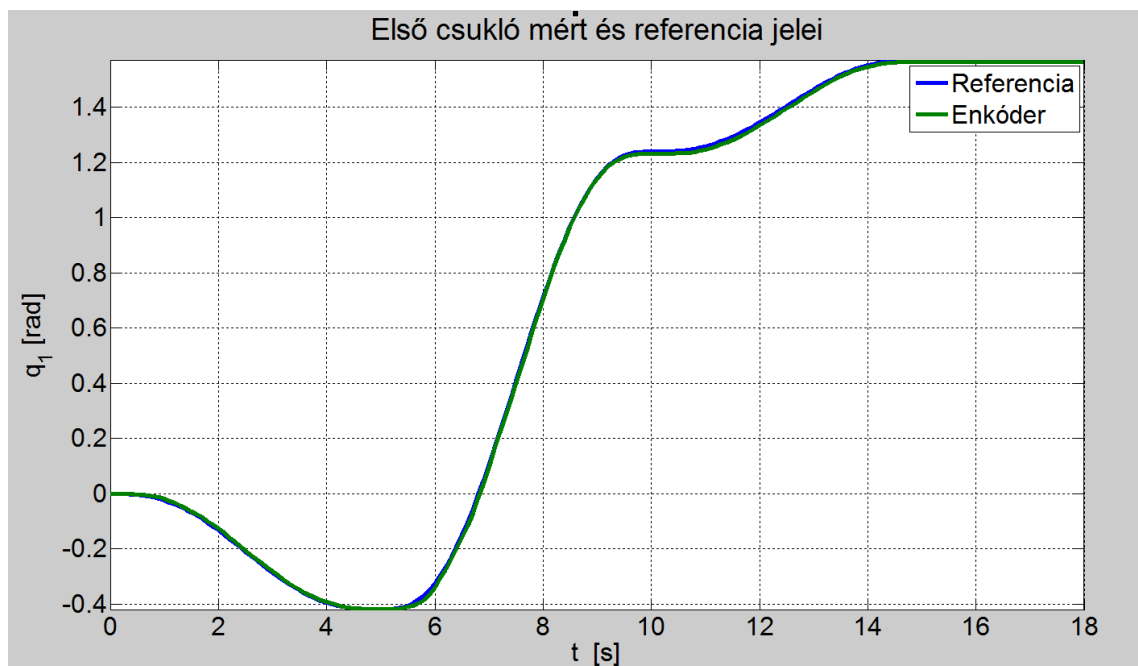
8.11. ábra – Második csukló alap- és mért jelei



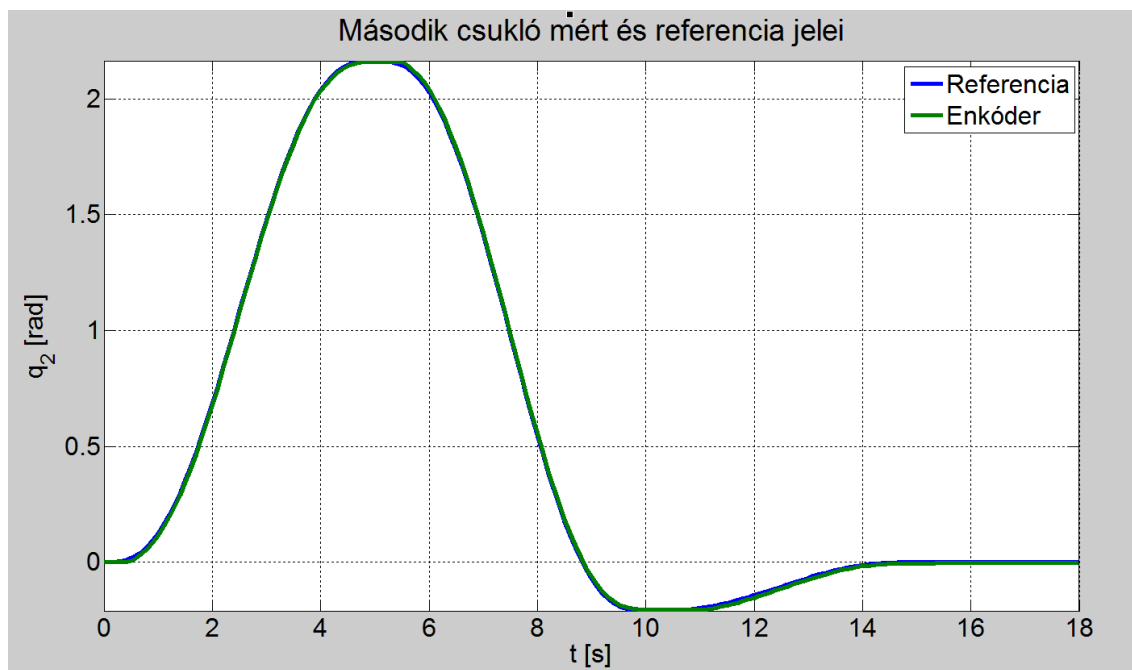
8.12. ábra – Szabályozók kimeneti jelei (relatív)

8.8 Látványos akadálykerülő pályán végig haladás

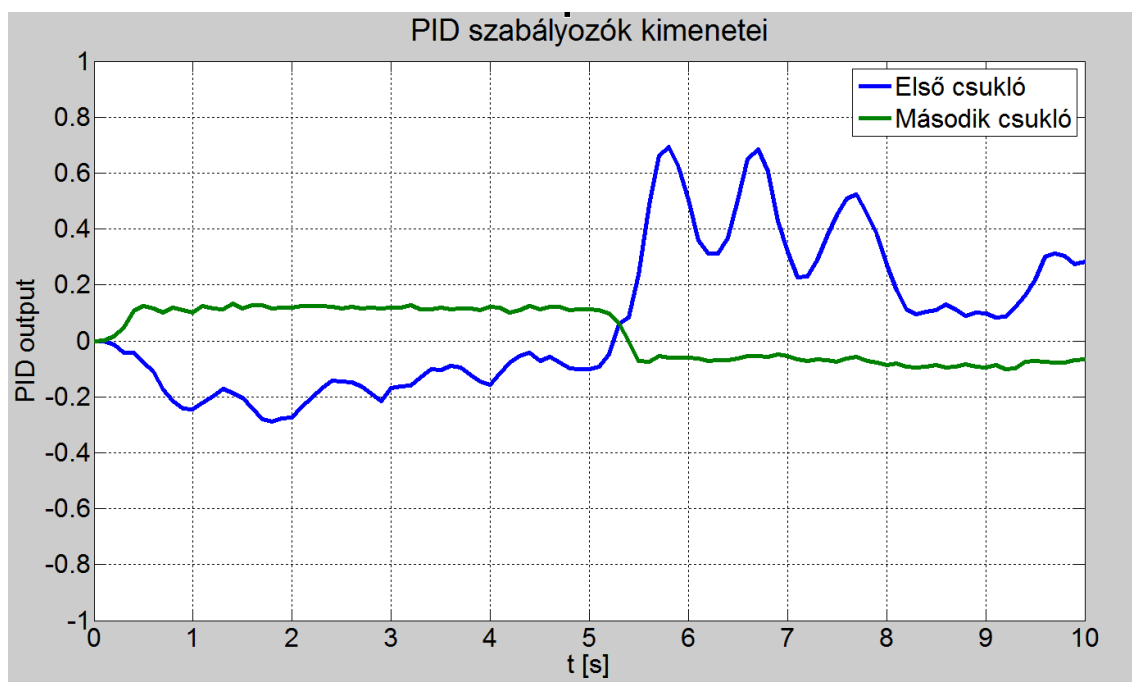
Az alábbi grafikonokon megfigyelhető, ahogy a robotkar kikerüli a definiált akadályt. Mindkét csukló 0 radiánból indul, az első csuklónak $\pi/2$ -be kell eljutnia, a második csukló a kezdeti pozíciójába kell érkezzen a mozgás végére.



8.13. ábra -- Első csukló alap- és mért jelei

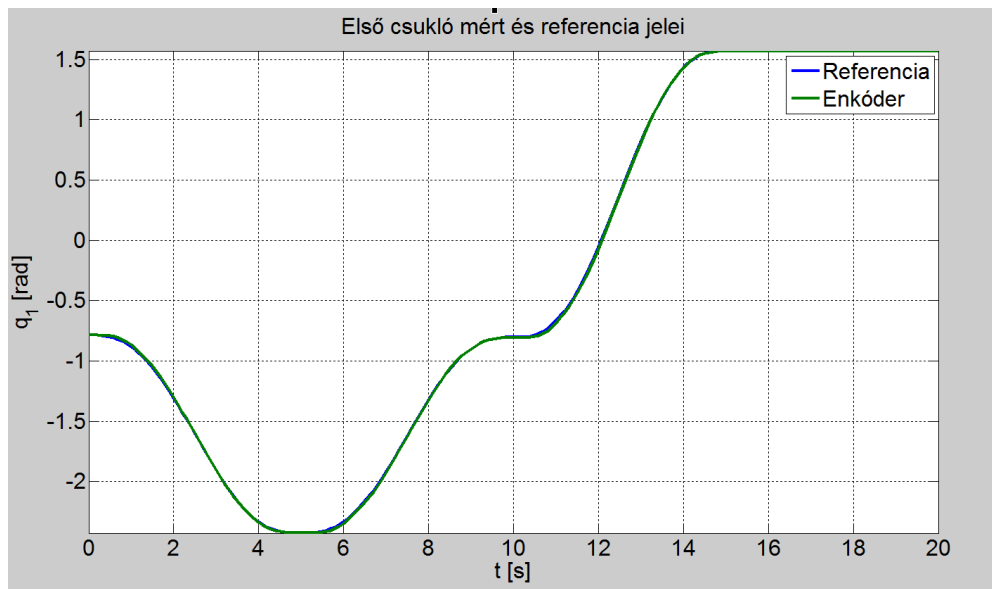


8.14. ábra – Második csukló alap- és mért jelei

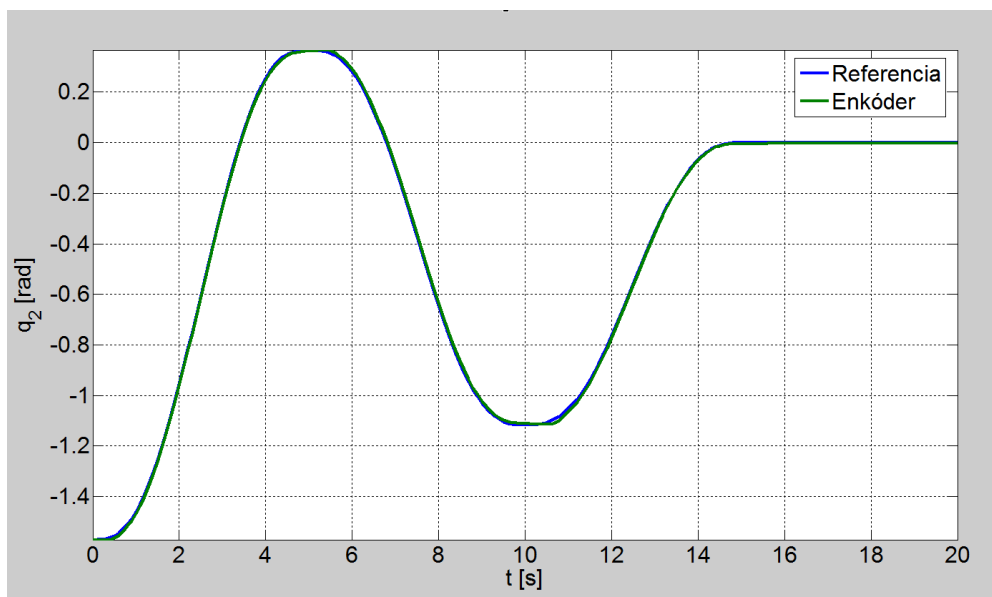


8.15. ábra – Első csukló alap- és mért jelei

8.9 Akadálykerülés nem 0 kezdőszögből



8.16. ábra – Első csukló alap és mért jelei, $-\pi/4$ kezdeti szögből indítva, $\pi/2$ végpozícióba küldve



8.17. ábra – Második csukló alap és mért jelei, $-\pi/2$ kezdeti szögből indítva, 0 végpozícióba küldve

8.10 További videók akadálykerülésről

Videó, ahogy a második csukló 0 kezdeti szögből indulva 0-ba érkezik, de közben befordul és kikerüli az akadályt: <https://youtu.be/oSSINiOayFg>

Videó egy több köztes pontot tartalmazó akadálykerülésről, nem 0 véghelyzetbe: <https://youtu.be/KGXnGgLSRyY>

8.11 Útmutató a szimuláció futtatásához (Matlab 2019b)

A szimuláció a valós robotkar nélkül is futtatható a Matlab Simulink 2019b vagy újabb verziójával, amennyiben az alábbi extra funkciók telepítve vannak:

- *Simscape Multibody*
- *Control System Toolbox*

A robotkar két pont közti mozgatása akadályok, szabályozás és térkép nélkül:

- *robotArm_basic.slx* futtatása (lásd: 3.3.3. fejezet)
- Kezdeti szögek megadása radiánban (*q1 starting* és *q2 starting*)
- Véghelyzet megadása koordinátás alakban (*X final* és *Y final*) [megj.: megadás méterben; együttes hossz ≤ 0.9 és ≥ 0.5 ; negatív előjel szabad]
- „RUN” gomb megnyomása

A robotkar mozgatása akadályok kerülésével:

Szükséges matlab kódok egy mappában megnyitva: *main.m*, *trajectory.m*, *Init.m*, *isFree.m*, *isRoute.m*, *dist.m*, *dist_XY.m*

- Kezdeti és véghelyzeti szögek megadása a *main.m*-ben (lásd: 4.4.1. fejezet)
- Akadályok felvétele a *trajectory.m*-ben (lásd: 4.3.1. fejezet és 5.2. fejezet)

Szabályozás nélküli szimuláció:

- *robotArm_trajectory.slx* futtatása, majd „RUN” gomb

Szabályozással kiegészített szimuláció:

- *robotArmPID_Trajectory.slx* futtatása, majd „RUN” gomb

[(1) megj.: a szimuláción belüli kizárólag ellenőrzés céljából változtatható paramétereket lásd: 4.4.2. fejezet]

[(2) megj.: a *main* futtatása során mindig új fa (vagy erdő) épül fel! Ezáltal változnak a futási paraméterek! Lehetőség van csak az útvonalkeresés újrafuttatására, az *init.m* meghívásának letiltásával a *trajectory.m* függvényben.]

A futási paraméterek beállítása a korábbiakban ismertetett (lásd: 4. fejezet) módon történhet.