



DIPLOMATERVEZÉSI FELADAT

Csorvási Gábor
szigorló villamosmérnök hallgató részére

Pályatervezési és pályakövető szabályozási algoritmusok fejlesztése robotautóhoz

A mobil robotok jövőbeli hatékony alkalmazásának feltétele, hogy minél önállóbb módon képesek legyenek navigálni a környezetükben előforduló akadályok között. A sikeres navigáció alapvető eleme a megfelelő pályatervezési és pályakövetési algoritmusok alkalmazása. A feladat célja ilyen algoritmusok fejlesztése és implementálása kormányzott (autószerű) mechanikájú robothoz. A feladat része az algoritmusok tesztelése egyrészt a V-REP robotszimulációs platformon, másrészt egy, a tanszéken rendelkezésre álló elektromos autómodellen.

A pályatervező algoritmus bemenetei a robot kezdeti és elérni kívánt pozíciója és orientációja, valamint a környezet geometriai leírása. Az algoritmus kimenete pedig egy, az autószerű robottal bezárható, ütközésmentes pálya, melynek megadása történhet primitív pályaelemek segítségével (pl. egyenes haladás, köríven fordulás), illetve pozíciók és orientációk mintavételes időfüggvényével (trajektória megadásával) is.

A pályakövető algoritmus bemenete a fenti formában megadott pálya (ill. trajektória), kimenete pedig a robot sebesség-, illetve kormányszög-alapjele.

A hallgató feladatának a következőkre kell kiterjednie:

- Ismerje meg a V-REP robot-szimulációs környezetet!
- Készítse el a robotautó modelljét a szimulátorban!
- Ismerje meg a robotikában alkalmazott pályatervezési megközelítéseket!
- Implementáljon egy globális pályatervező algoritmust, amely már a tervezési fázisban figyelembe veszi a robot kinematikai korlátozásait!
- Implementáljon egy pályakövető algoritmust, amely lehetővé teszi a megtervezett pálya stabil lekövetését!
- Tesztelje az algoritmusokat a V-REP környezetben és a valós roboton is, értékelje a tapasztalatokat, és ha szükséges, adjon javaslatokat az algoritmusok javítására!

Tanszéki konzulens: Kiss Domokos

Budapest, 2014. március 14.

Dr. Vajk István
egyetemi tanár
tanszékvezető



Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Automatizálási és Alkalmazott Informatikai Tanszék

Pályatervezési és pályakövető szabályozási algoritmusok fejlesztése robotautóhoz

DIPLOMATERV

Készítette
Csorvási Gábor

Konzulens
Kiss Domokos

2014. december 12.

Tartalomjegyzék

Kivonat	4
Abstract	6
1. Bevezető	7
1.1. Problémafelvetés	7
1.2. Pályatervezés elmélete	7
1.2.1. Alapvető fogalmak	7
1.2.2. Pályatervezők osztályozása	9
2. Útvonaltervezés C*CS pályákkal	11
2.1. Reeds-Shepp lokális pályák	11
2.2. C*CS lokális pályák	12
2.3. C*CS approximációs módszer	12
2.3.1. Globális tervező	13
2.3.2. Lokális tervező alkalmazása	13
2.4. $c\bar{c}S$	16
2.5. Eredmények	17
2.5.1. Mérések	17
2.5.2. Fejlesztési lehetőségek	17
2.6. Új globális tervező	17
2.6.1. RRT	18
2.6.2. RTR	18
3. Pálya időparamétere	20
3.1. Időparaméterezés	20
3.2. Jelölések	21
3.3. Korlátozások	22
3.4. Geometriai sebességprofil	25
3.5. Újramintavételezés	27
4. Pályakövető szabályozás	34
4.1. Pályakövetés	34
4.1.1. Sebességszabályozás	34

4.1.2. Referenciapont-választás	35
4.1.3. Túlhaladás problémája	37
4.2. Virtuális vonalkövető szabályozás	37
4.3. Eredmények	39
5. Algoritmusok megvalósítása	40
5.1. Szimuláció – V-REP	40
5.1.1. Sserver program	41
5.1.2. Kliens program	41
5.1.3. Implementáció	42
5.2. Szimulációs eredmények	43
6. Megvalósítás valós roboton	45
6.1. Felépítés	45
6.2. Nehézségek	46
6.2.1. Helymeghatározás	46
6.2.2. Odometria	47
6.2.3. Kotyogás	47
6.3. Jövőbeli tervezek	48
7. Összegzés	49
Köszönetnyilvánítás	50
Irodalomjegyzék	52
Függelék	53
F.1. Megjegyzés a CD melléklettel kapcsolatban	53

HALLGATÓI NYILATKOZAT

Alulírott *Csorvási Gábor*, szigorló hallgató kijelentem, hogy ezt a diplomatervet 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 átvettettem, egyértelműen, a forrás megadásával megjelöltem.

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

Budapest, 2014. december 12.

Csorvási Gábor
hallgató

Kivonat

A mobil robotok manapság egyre inkább feltörekvőben vannak. Már nem csak az ipar fedeli fel őket, hanem lassan a minden nap életünk részévé válnak. Azonban még rengeteg elméleti és gyakorlati kérdés vár megoldásra, hogy az ilyen robotokkal rendszeresen találkozzunk. A mobil robotika egyik legalapvetőbb kérdése az akadályok jelenlétében történő mozgástervezés és mozgásvégrehajtás. A dolgozatban ezt a kérdéskört járom körül, foglalkozok a globális és lokális geometriai pályatervezéssel, pályamenti sebességprofil kialakításával, valamint pályakövető szabályozással. Ezeket egy kerekeken guruló robotmodellre alkalmazom, mind szimulált, mind valós környezetben.

A dolgozatban bemutatom a leggyakrabban használt pályatervezési algoritmusokat, és az ezekhez kapcsolódó előnyöket és problémákat. Külön kitérek az általam vizsgált robotmodellnél felmerülő kinematikai korlátozásokra, és ezek hatásaira a pályatervezésben. Egy approximációs pályatervezési megközelítést mutatok be a dolgozatomban, amely egy globális és egy lokális tervező algoritmus együttes használatán alapszik.

Az általam alkalmazott C*CS lokális tervező, és approximációs módszer, az irodalomban jól ismert, az autószerű robotok esetén alkalmazott Reeds-Shepp pályákon alapszik. Az algoritmus körívek (C) és egyenes szakaszok (S) kombinációjából állít elő autókkal is lekövethető pályákat. Ehhez szükség van egy globális tervező által előállított pályára. A dolgozatomban bemutatok egy egyszerű celladekompozíciós eljárást, amely ezt elvégzi, majd ennek segítségével szemléltetem a C*CS algoritmus működését. Végül bemutatok egy új globális tervező algoritmust (RTR), mely a differenciális robotok számára készült, de kiküszöböli a celladekompozíciós eljárás hibáit.

A megtervezett geometriai pálya még nem tartalmaz információt a mozgás időparamétereire (a robot sebességére, gyorsulására vagy szögsebességére) nézve. Ebből kifolyólag ismertetem azon algoritmust is, amely a pályamenti sebességprofil meghatározására lett kitalálva. Ez a profil a robot maximális sebessége, maximális gyorsulása, maximális szögsebessége és a robot kerekeinek maximális gyorsulása alapján számolható ki. Az így kialakuló pályát ezután újramintavételezzük, hogy időben egyenletes mintavételű pálya álljon rendelkezésre a pályakövető szabályozás számára.

A pályakövető algoritmus a robot pályamenti sebességét és az orientációját egymástól függetlenül szabályozza. A szétcsatolt rendszer sebesség és kormányszög beavatkozó jelei, a robot kinematikai egyenletei alapján közvetlenül képzik a beavatkozó jeleket. A sebesség-szabályozási kört a robotban található PI szabályozó valósítja meg. Az orientációszabályozás egy mozgás közbeni korrekciót hajt végre, amelynek alapját a robot későbbi

előírt pozíciói képezik.

Az algoritmusokat a V-REP robotszimulációs környezetben implementáltam és teszteltem, majd működésüket valós roboton is vizsgáltam. Végül bemutatom a felhasznált autószerű robotot, és annak működését, kitérek az elkészítés közben felmerült nehézségekre, és a jövőbeli tervekre is.

Abstract

The research and application of mobile robots is nowadays increasingly widespread. Beyond industrial applications they are getting popular in personal usage as well. However, there are a lot of theoretical and practical issues to be resolved. One of the most fundamental aspects of mobile robotics is motion planning and control in environment populated with obstacles. In this paper we discuss global and local geometric path planning, velocity profile generation and motion control along the path. We simulated the investigated methods and tested with a car-like robot.

In this paper we present the most commonly used path planning algorithms, and their benefits and disadvantages. We discuss the kinematic constraints for the tested robot model and the consequences of these constraints. We present an approximation method for path planning, which is based on a global and a local planner algorithm.

We use the C*CS local planner method, which is based on the well-known Reeds-Shepp method. The algorithm uses arcs (C) and straight segments (S) to provide a path for car-like robots. To reach this, a global planner method is presented. After the global path is generated the C*CS local planner is used to approximate the global path. At the end of this section a new global planner method is presented, to correct the flaws of the previously shown method.

The path generated previously does not contain any information about the robot's velocity, acceleration and angular velocity. Therefore, we present an algorithm developed to determine the velocity profile. The profile is based on parameters such as the robot's maximum velocity, maximum acceleration, maximum angular velocity along the path and the maximum acceleration of the robot's wheels. After velocity profile generation the geometric path needs to be re-sampled to obtain a path having uniform time-sampling for the motion controller.

The path following algorithm controls the robot translational velocity and orientation in two independent control loops. The velocity and steering angle control signals of the decoupled system passed directly to the robot, based on the kinematic equations of the robot. The velocity control loops implemented on the robot by a PI controller. The orientation controller is based on orientation error between the current robot configuration and a future path point.

We have implemented and tested the algorithms in V-REP robot simulation framework and with a real robot as well. Finally we present the structure of the used real robot, and the difficulties during the development and implementation, and the future plans.

1. fejezet

Bevezető

1.1. Problémafelvetés

A helyváltoztatásra képes, úgynevezett mobil robotok esetében alapvető szituáció, hogy a robotnak a feladata végrehajtásához el kell jutnia egy célpontba. Ehhez önmagának kell az adott környezetben megterveznie a pályát, és emberi beavatkozás nélkül kell sikeresen eljutnia a kívánt célpontba. A probléma nagyságrendileg nehezebb amikor a robot környezetében akadályok is találhatóak.

Céлом azon megközelítések és módszerek áttekintése, amelyek megoldást nyújtanak az autonóm pályatervezés kérdésére. Néhány módszert részletesen is ismertetek, ezeket szimulátoron és valós robotokon is implementáltam, illetve teszteltem. A pályatervezéshez szorosan kapcsoló téma a mozgásirányítás, amivel szintén foglalkoznunk kell, hogy valós környezetben ténylegesen használtható eljárásokat kapunk.

1.2. Pályatervezés elmélete

Az elmúlt időszakban a pályatervezéssel kapcsolatban igen sok kutatás foglalkozott [1]. Ahhoz, hogy ezeket az algoritmusokat ismertessük, be kell vezetnünk néhány alapvető fogalmat.

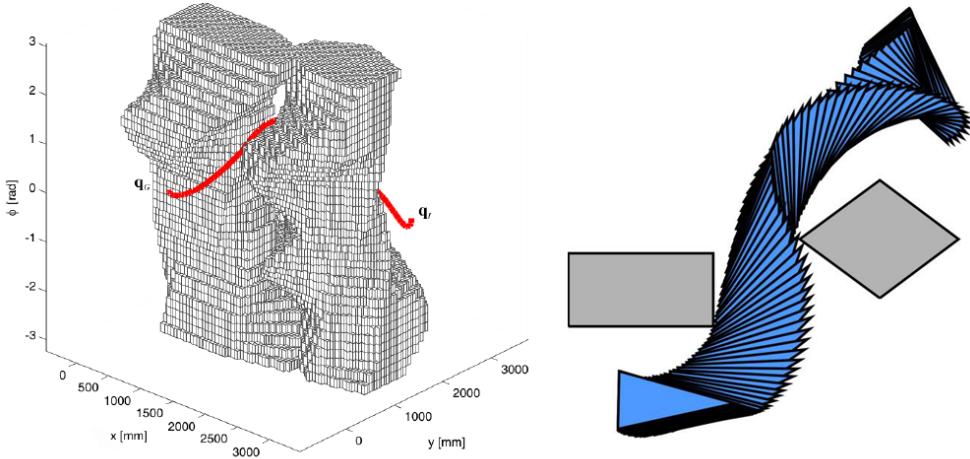
1.2.1. Alapvető fogalmak

A pályatervezés során a robot pillanatnyi állapotát a *konfigurációjával* írhatjuk le. Síkban mozgó robotok esetében a konfiguráció a következőket tartalmazza [2]:

$$q = (x, y, \theta), \quad (1.1)$$

ahol q a robot konfigurációja, x, y határozza meg a robot pozíóját a síkon és θ határozza meg a robot orientációját.

Egy lehetséges környezetben a robot összes állapotát, a *konfigurációs tér* adja meg, amit C -vel jelölünk. A konfigurációs tér azon részhalmazát, amely esetében a robot a környezetben található akadályokkal nem ütközik, *szabad (konfigurációs) térnek* nevezünk



1.1. ábra. Konfigurációs tér szemléltetése egy adott útvonal során. A konfigurációs térben a piros vonal a robot útját a célpontja felé [3].

(C_{free}). E halmaz komplementere azokat a konfigurációkat tartalmazza, amelyek esetén a robot ütközne az akadályokkal ($C_{obs} = C \setminus C_{free}$). A konfigurációs teret az 1.1. ábrán szemléltetjük.

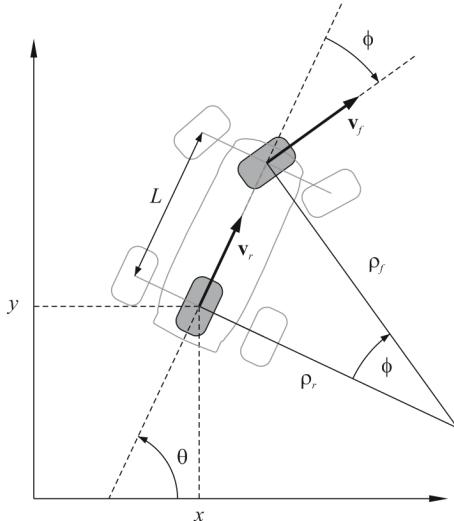
A korlátozások ismerete alapvető fontosságú a pályatervezés és mozgásirányítás során. A környezetben elhelyezkedő akadályokat *globális korlátozásoknak* tekintjük, a robothoz kapcsolódó korlátozásokat pedig *lokális korlátozásoknak* [2]. A lokális korlátozásokat a robot konfigurációs változóinak differenciál-egyenletével írhatjuk le, ezért gyakran nevezik őket *differenciális korlátozásoknak* is. Differenciális korlátozások vonatkozhatnak sebesség (kinematikai) és gyorsulás mennyiségre is (dinamikai korlát). Dolgozatomban csak kinematikai korlátozásokkal foglalkozni, dinamikai korlátokkal nem.

Egy autó esetén mindenki számára egyértelmű, hogy csak bizonyos íveken tudunk mozogni, egy adott konfigurációból nem tudunk a konfigurációs tér bármely irányába elmozdulni, habár a szabad tér bármely konfigurációjába eljuthatunk. Autónál emiatt nem olyan egyszerű például a párhuzamosan parkolás. Azokat a robotokat, amelyek ehhez hasonló korlátozásokkal rendelkeznek, *anholonom rendszereknek* nevezzük. Az anholonom korlátozásról akkor beszélünk, ha a korlátozás olyan differenciál egyenettel írható le, amely nem integrálható.

Az általam vizsgált robottípus az *autószerű robot*, egy anholonom rendszer. Viszont léteznek olyan robotok, amelyek nem rendelkeznek anholonom korlátozásokkal (holonom rendszerek), ilyenek például az omnidirekcionális robotok. Egy omnidirekcionális robot képes bármilyen konfigurációból a tér bármely irányába elmozdulni.

Robotmodell

Az általam vizsgált robot típust, az autószerű robotot mindenki jól ismeri és elterjedtsége megkérőjelezhetetlen, de a kinematikai leírása már kevésbé ismert, ellenben az 1.2. ábra segítségével könnyedén levezethető:



1.2. ábra. Autószerű robot modellje.

$$\begin{aligned}\dot{x} &= v_r \cos \theta \\ \dot{y} &= v_r \sin \theta \\ \dot{\theta} &= \frac{v_r}{L} \tan \phi,\end{aligned}\tag{1.2}$$

ahol L az első és hátsó tengelyek távolsága, ϕ a kormányszög, v pedig a hátsó tengely középpontjának tangenciális sebessége, amelyet a robot referencia pontjának nevezünk.

1.2.2. Pályatervezők osztályozása

Mielőtt belekezdenék az általam megvizsgált pályatervező algoritmusok részletesebb ismeretébe, tekintsük át az irodalomban használatos módszereket.

Geometriai tervezés szerinti csoportosítás

A pályatervezők geometriai módszerei szerint alapvetően két csoportot különböztetünk meg: a *globális tervezők* és a *reaktív tervezők* csoportját [2].

A globális tervezők esetében a konfigurációs tér egészét figyelembe vesszük a tervezéskor, míg a reaktív tervezők csupán a robot környezetében lévő szűkebb tér ismeretére építenek. A globális tervezők előnye, hogy képesek akár optimális megoldást is találni, míg a reaktív tervezők egy lokális minimumhelyen ragadhatnak, nem garantálható, hogy a robot eljut a célponthoz. A globális tervezés hátránya azonban a lényegesen nagyobb futási idő, ezért gyakran változó vagy ismeretlen környezet esetén előnyösebb lehet a reaktív tervezők használata.

A reaktív tervezők esetében a robot alakját körrel szokták közelíteni, ezzel is egyszerűsítve a tervezés folyamatát. Ezzel szemben globális algoritmusok a robot pontos alakját figyelembe vesszik, aminek nagy jelentősége van szűk folyosókat tartalmazó pálya esetén.

Az általam bemutatott algoritmusok is figyelembe veszik a robot pontos alakját.

A globális tervezők esetén megkülönböztetünk mintavételes és kombinatorikus módszereket [1]. A mintavételes módszerek a konfigurációs teret véletlenszerűen mintavételezik, és ez alapján próbálnak utat keresni a célpontba. Ellenben a kombinatorikus módszerek a környezet pontos geometriai modellje alapján terveznek utat. Ezen algoritmusok előnye, hogy ha nem létezik megoldás, akkor ezt véges időn belül képesek eldönten, ám a mintavételes tervezők ezt nem tudják véges időn belül megtenni.

Irányított rendszer szerinti csoportosítás

A robotok, mint irányított rendszerek esetén megkülönböztetjük az anholonom és holonom rendszereket a pályatervezők csoportosítása esetén is. Anholonom rendszerek esetén önmagában a robot állapotváltoztatása sem triviális feladat. Azokat az eljárásokat, amelyek képesek egy anholonom rendszert egy kezdő konfigurációból egy cél konfigurációba eljuttatni az akadályok figyelembe vétele nélkül, *lokális tervezőknek* hívjuk [1].

Gyakran már a globális tervező figyelembe veszi a robot korlátozásait és ennek megfelelő geometriai primitíveket használ, de a globális tervező által megtervezett pályát közelíthetjük egy lokális tervezővel is, ha az anholonom robotunk közvetlenül nem tudná lekövetni a globális pályát. Ezt az eljárást, approximációs módszernek nevezik. Egy ilyen algoritmusra látunk példát a következő fejezetben.

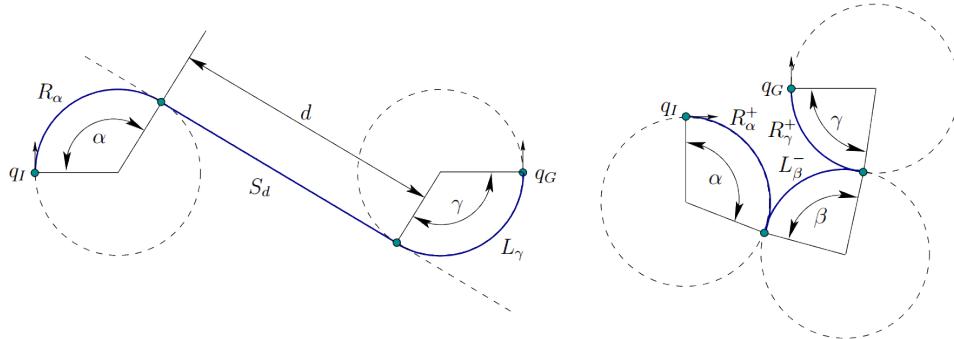
2. fejezet

Útvonaltervezés C*CS pályákkal

A C*CS és a $c\bar{c}S$ algoritmus Kiss Domokos munkája [4]. Az algoritmusok elsődlegesen autószerű robotok számára terveznek pályát, de az így tervezett pálya egy differenciális robot számára is végrehajtható. Feladatom az algoritmus implementálása volt C++ nyelven, majd annak tesztelése szimulációs, illetve valós környezetben. A fejezetet az algoritmus ismertetésével kezdem, majd kitérek az implementációs problémákra, és az elért eredményekre is.

2.1. Reeds-Shepp lokális pályák

Az anholonom rendszerek irányítása akadályuktól mentes környezetben is egy igen bonyolult feladat. Sok esetben nem adható meg általános algoritmus, csak néhány speciális rendszer esetén. Szerencsére ilyen rendszerek közé tartoznak a differenciális robotok, az autószerű robotok, amelyek csak előre mozoghatnak (Dubins autó), és azok amelyek előre és hátra is képesek mozogni.



2.1. ábra. Dubins és Reeds-Shepp megoldások [1]

Az utóbbi típusú robotokat hívjuk Reeds-Shepp autóknak, melyeknél bizonyított, hogy bármely kezdő- és célkonfiguráció közt a legrövidebb utat megtalálhatjuk 48 lehetséges megoldás közül, amelyből kettő látható a 2.1 ábrán. Ezek a megoldások maximum öt egyenes vagy körív kombinációjából állhatnak, és a pályák maximum két csúcsot tartalmazhatnak, azaz ennyiszer lehet irányt változtatni a végrehajtás közben [5]. A megoldások száma egyéb

megkötések árán tovább csökkenthető.

Mint látható akadályuktól mentes környezetben találhatunk optimális útvonalat, de ennek hátránya, hogy minden minimális sugarú pályákat feltételez, mely egy valós esetben nem életszerű, illetve a pályák lehetnek igen bonyolultak is. Azonban ha elvetjük az optimalitás igényét - amit egyébként is meg kell tennünk, ha egy globális tervező részeként alkalmazzuk a módszert - akkor a lehetséges megoldásokon jelentős mértékben egyszerűsíthetünk.

2.2. C*CS lokális pályák

A lokális tervezők bármely kezdő- és célkonfiguráció páros esetén megoldást kell nyújtanak, de megfelelő koordináta-rendszer választásával egyszerűsíthetünk a számításokon. Tegyük fel, hogy egy ilyen választás mellett adódott $q_I = (x_I, y_I, \theta_I)$ kezdő és $q_G = (0, 0, 0)$ célkonfiguráció. Ha eltekintünk a minimális fordulási sugár korlátozásától, és feltezzük, hogy $\theta_I \neq 0$, akkor könnyen belátható, hogy egy kör és egy egyenes segítségével elérhető a célkonfiguráció. Először egy érintő körön elfordulunk a $\tilde{q}_G = (\tilde{x}_G, 0, 0)$ köztes célkonfigurációba, majd egy egyenes mentén végighaladunk a célig. Az ehhez tartozó kör sugarát a következő egyenlet segítségével számíthatjuk:

$$\rho_{I,\tilde{G}} = \frac{y_I}{1 - \cos \theta_I} \quad (2.1)$$

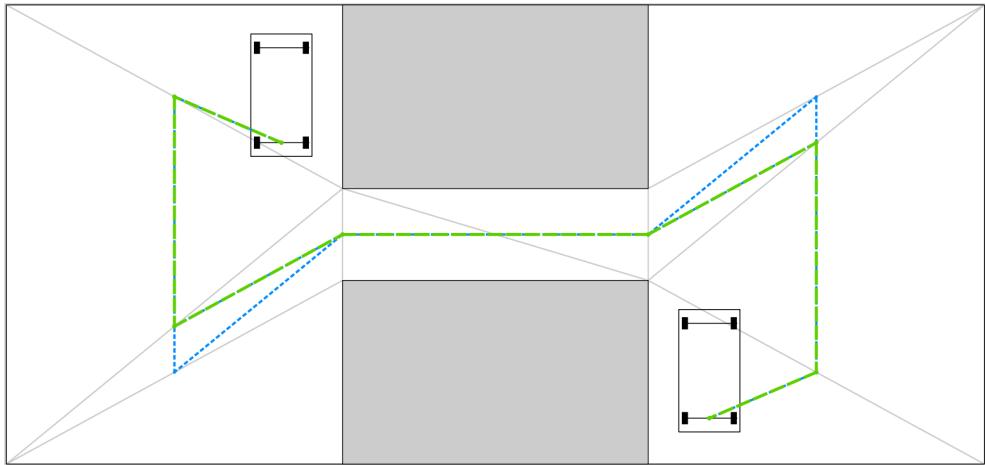
Ha a kiadódó sugár kisebb, mint a minimálisan megengedett ($|\rho_{I,\tilde{G}}| < \rho_{min}$), esetleg $\theta_I = 0$, akkor egy egyenes vagy egy kör segítségével egy köztes kezdőkonfigurációba ($\tilde{q}_I = (\tilde{x}_I, \tilde{y}_I, \tilde{\theta}_I)$) kell eljutnunk, ahol biztosított, hogy $\tilde{\theta}_I \neq 0$ és, hogy $\rho_{\tilde{I},\tilde{G}} \geq \rho_{min}$. Megjegyzendő, hogy az első szakasz nem lehet egyenes, ha $\theta_I = 0$, $\theta_I = \pi$ vagy $|y_I| < 2\rho_{min}$. Bebizonyítható [4], hogy \tilde{q}_I köztes konfigurációt végtelen sokféleképpen megválaszthatjuk.

Hogy egyszerűsítsük a jelöléseket, a továbbiakban az egyenes szakaszokra S a körívekre pedig a C betűk segítségével hivatkozunk. Ezt felhasználva belátható, hogy a célpontba egy SCS , vagy egy CCS pálya segítségével eljuthatunk. Könnyen belátható, hogy ha egy körív (C) sugarával a végtelenbe tartunk, akkor a szakasz az egyeneshez tart. Az olyan speciális köríveket, amelyek sugarára végtelen is lehet, C^* -gal jelöljük. Innen a módszer neve a C^*CS .

2.3. C*CS approximációs módszer

Az általam használt algoritmus egy approximációs módszert alkot, mely egy előzetes globális pályát rekurzív módon felbont kisebb szakaszokra, majd ezekre próbál illeszteni egy-egy fentebb bemutatott C*CS pályát.¹ A végeredményül elkészült, az algoritmus által visszaadott pálya autószerű robotokkal könnyedén lekövethető, mivel elsődlegesen ezek számára lett kialakítva. Ennek ellenére a megoldást természetesen egy differenciális robot is képes lekövetni, mivel az nem rendelkezik korlátozással a forduló kör sugarát illetően.

¹Bár a lokális tervező algoritmus neve a C*CS, de a végeredményben kialakult pálya összességében is körök és egyenesek kombinációjából áll, így ez a név ráragadt az approximációs módszerre is.



2.2. ábra. Előzetes pálya tervezése, folytonos vonal jelzi a felbontást, pontozott vonal a gráfot, szaggatott pedig az elkészült előzetes pályát

2.3.1. Globális tervező

A szükséges előzetes pálya bármilyen globális tervező eredménye lehet. Elsődleges célja egy mankó nyújtása a későbbi tervező számára. A végső megoldásnak nem feltétele, hogy az előzetes pálya akár egyetlen pontját is tartalmazza.

Mi erre a céllra egy celladekompozíción alapuló algoritmust használtunk. Ez az eljárás a környezetet háromszögekre bontja [6], majd ezeknek a háromszögeknek az oldalfelező pontjait összekötve gráfot alkot. Ebbe beszűrja a kezdő- és célkonfigurációt, majd ezeket összeköti a legközelebb álló néhány ponttal. Az éleket a pontok egymástól való távolságával súlyozzuk, majd ebben a gráfban a Dijkstra-algoritmus [7] segítségével megkeressük a legrövidebb utat. Erre egy példát a 2.2 ábrán láthatunk.

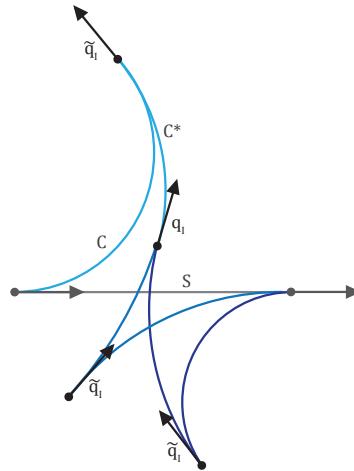
Ennek a megoldásnak az előnye, hogy a szabad terület közepén alkot pályát, így ha az autó ezt követi, akkor bármilyen irányú manőverezésre lesz lehetősége, ha a pálya ezt megengedi. További előnye, hogy ez egy kombinatorikus eljárás, így véges időn belül képes megmondani, hogy létezik-e megoldás. Az eljárás egyik fő hibája, hogy a háromszögelés miatt csak sokszögekkel leírható akadályokkal képes dolgozni, és még ebben a formájában nem veszi figyelembe az autó kiterjedését. Ezen könnyen lehet segíteni, ha figyelembe vesszük az oldalfelező pontok közötti szakaszok távolságát az akadályuktól, és ha a pálya- és az akadályél túl közel vannak egymáshoz, akkor töröljük az élt a gráfból. Sajnos az eljárás egyéb negatívumokkal is bír, amiről a fejezet végén még szót ejtek.

2.3.2. Lokális tervező alkalmazása

Ha a globális tervező tudott visszaadni megoldást, akkor az algoritmus tovább folytatódik a következőképpen: Az előzetes pálya két konfigurációját kiválasztjuk, és a fentebb említett C*CS pályákat keresünk köztük. Az eljárás először a pálya két végpontja között keres útvonalat, ami egyszerű esetekben akár rögtön megoldásra is vezethet, felgyorsítva az algoritmus működését. Ha ez a keresés nem járt sikерrel, akkor az előzetes pályát megfelezi az algoritmus, és az első konfiguráció, valamint az új célkonfiguráció között keres megoldást. Ezt

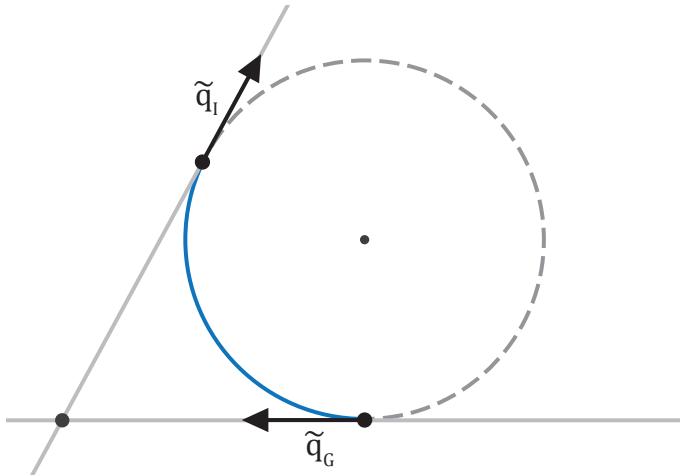
egészen addig ismétli, míg van köztes konfigurációs pont. Ha elfogyott, további pontokat illeszt a pályába.

Az előzőekben láthattuk, hogy a C*CS végtelen sok megoldást nyújt. Lokális esetben ez nem feltétlen hasznos, de akadályok jelenlétében ez megváltozik, mivel így sokkal nagyobb valószínűséggel találhatunk végrehajtható pályát. Természetesen az összes megoldást nincs lehetőségünk kipróbálni, így ezt a problémát valamilyen mintavételező eljárással kell megoldanunk.



2.3. ábra. $q_{\bar{i}}$ megválasztása

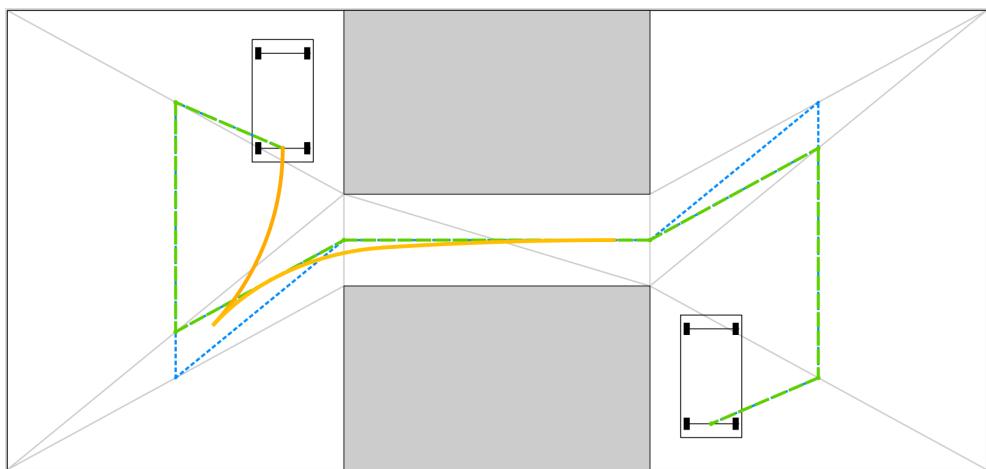
A végtelen sok megoldást a \tilde{q}_I kiválasztásának szabadsága okozza. Erre egy példa láttható a 2.3 ábrán. Ezért az algoritmus összegyűji azokat a konfigurációkat, melyeket a q_I konfigurációból ütközés nélkül elérhetünk. Ehhez a környezetet fel kell osszuk egységnyi távolságokra, mivel így véges sok lehetőséget kapunk. A kiszámítás ideje természetesen függ a választott távolságegységtől és a környezet méretétől. Az eredmények azt mutatják, hogy a teljes algoritmus futásának ez a leghosszabb része, ami nem meglepő, mivel a körívek kiszámítása komplex művelet, és ezt egy adott pont esetén a robot testének minden csícsára ki kell számoljuk, hogy ütközést tudunk detektálni. A művelet hatékonyságán több módon lehet javítani, például nagyobb távolságegység megválasztásával. Másik javítási lehetőség, ha előre elkészítünk egy foglaltsági mátrixot, ami megmondja az adott pont akadályon beül van-e, így ezekre a pontokra nem kell a számítást elvégezni. Mivel az így kapott körívek egy adott kezdőkonfigurációhoz tartoznak, további gyorsításra ad lehetőséget, ha az approximációs lépésekben inkább a célkonfiguráció pontját mozgatjuk, így nem kell újra és újra kiszámolni a köríven elérhető sokaságot.



2.4. ábra. Középső körív számítása érintő körrel

Az algoritmus további részében a 2.2 pontban említett módon, a hátralévő körív, és egyenes szakasz kiszámítása a feladat. Egy irányított kör esetén ez két lehetséges pályát jelent, amint az látható a 2.4 ábrán. Ezek után nem elég csak a körívek végrehajthatóságát ellenőriznünk, hanem meg kell nézzük ezt a hátralévő egyenes szakaszokra is. Ugyan a globális pálya tervezésekor ellenőriztük ezeket, de az érintő körök keresésekor nem volt feltétel, hogy az érintési pontok ezeken a szakaszokon belül helyezkedjenek el.

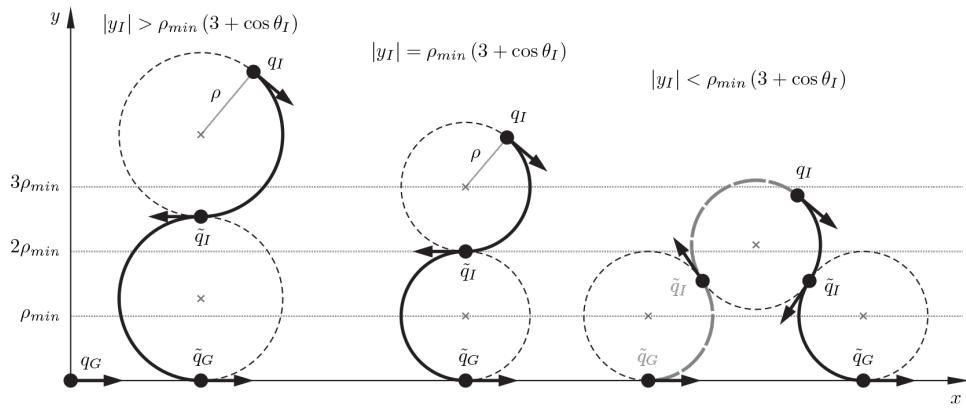
Végül az így keletkező végrehajtható pályák sokasága közül ki kell választanunk egyet. Ezt többféleképpen megtehetjük. Talán a legkézenfekvőbb a legrövidebb megoldás kikeresése és beillesztése az előzetes pályába. Itt érdemes megemlíteni, hogy a végeredmény akkor fog igazán hasonlítani a valósághoz, ha lecsökkentjük a tolatások számát, mivel az emberek nagy többsége nem szeret tolatva közlekedni. Hogy ezt megtehessük, az algoritmus opcionálisan elfogad egy súlytényezőt, mellyel a tolató szakaszok „hosszát” tudjuk megnövelni.



2.5. ábra. A C^*CS algoritmus működés közben

2.4. $c\bar{c}S$

Ha az előzőekben látottak nem vezetnek megoldásra, tehát nincs olyan C*CS pálya, mely végrehajtható lenne, akkor egy kisebb szakaszt kell választanunk a globális pályából. Ezt viszont nem tehetjük meg végtelenséggig, mivel a globális pálya általában egyenes szakaszok együtteséből áll. Ezek a szakaszok határozzák meg az S szakasz egyenesét, így tovább bontani nincs értelme, mert nem változtatja meg a tervezett pályát. Ezért az új konfigurációkat a töréspontokban kell elhelyezni, viszont önmagában ez még nem biztosítja, hogy találunk megoldást. Valahogy biztosítanunk kell azt, hogy az algoritmusunk konvergáljon a megoldás felé, amihez olyan lokális tervezőre van szükségünk, mely teljesíti a topológiai feltételt [4]. Ha ezt biztosítani tudjuk, akkor az approximációs algoritmusunk teljes lesz. A teljesség itt azt jelenti, hogy az algoritmus minden esetben megoldással tér vissza, mikor a globális tervező érvényes pályát szolgáltatott. Tehát a körívekkel való közelítés nem csökkenti a megoldás létezésének esélyét.



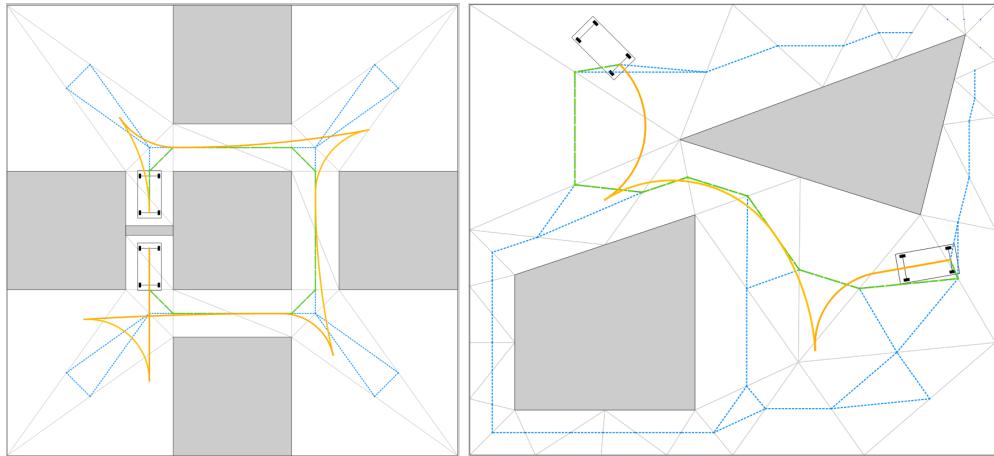
2.6. ábra. A $c\bar{c}S$ algoritmus megoldásai különböző y_I esetén[4]

Olyan esetekben, amikor a globális pályát tovább kellene bontanunk, átváltunk a $c\bar{c}S$ algoritmusra, mely a topológiai feltételt teljesíti. Ez az algoritmus a C*CS algoritmus egy módosított változata, mely csak egy megoldást ad egy konfiguráció párra. Az eljárás lényege, hogy az első két kör sugara megegyező, de ellentétes előjelű, tehát a másik irányba kell forgassuk a kormányt. A komplementer jelölés jelzi az előjel változását. Ahogy a 2.6 ábrán látható, ha q_I túl közel van a q_G egyeneséhez, akkor a körök egymás mellett elcsúsznak, és két megoldást is adnak.

Bár a $c\bar{c}S$ egy megoldást ad, a végrehajtásakor több lehetséges megoldást is „eldob”. A módszer implementációját úgy készítettem el, hogy minden ilyen megoldást ellenőrizzen, ha esetleg a legrövidebb nem lenne végrehajtható, akkor válasszon másikat. Jogosan felmerülhet a kérdés, hogy ha ez az algoritmus minden esetben nyújt megoldást, akkor miért nem ezt használjuk a C*CS helyett? Bár valóban a $c\bar{c}S$ minden esetben nyújt megoldást, a C*CS több lehetséges megoldás közül választ, így a gyakorlatban természetesebb pályákat ad eredményül.

2.5. Eredmények

A feladatom megvalósításához rendelkezésre állt a C*CS algoritmus egy MATLAB scriptben megírt változata, ellenben ez csak demonstrációs céllal szolgált, a feladatot lassan hajtotta végre, valószínűleg az interpretált működés következtében, ezért vált szükségessé egy C++ implementáció.



2.7. ábra. A *C*CS* algoritmus megoldása különféle környezetekben

2.5.1. Mérések

Pontos méréseket nem végeztünk, de nagyságrendileg százszoros gyorsulást sikerült elérnünk, és a legbonyolultabb környezetben is egy másodpercen belül sikerült megoldást találnia az algoritmusnak². Ez egy igen nagy előrelépés, így valószínű, hogy egy kisebb teljesítményű beágyazott számítógépen is elfogadható időn belül végez.

2.5.2. Fejlesztési lehetőségek

A fejlesztés során odafigyeltem, hogy hol lehetne gyorsítani, módosítani a működésen. Ahol ez egyszerűen megvalósítható volt, ott ezeket megtettem, de maradtak további fejlesztési lehetőségek is a programban, például számos helyen lehetne a futást párhuzamosítani.

Az esetek nagy többségében a celladekompozíciós eljárással tervezett globális pálya jó eredménnyel szolgál, de néhány speciális esetben – ilyen két szűk merőleges folyosó találkozása, nem található megoldás. Részben hasonló probléma mikor a célkonfiguráció és az utolsó szakasz iránya pont ellentétes – ilyen probléma merül fel a párhuzamos parkolás esetén is. Ezek elkerülésére egy másik globális tervezőt kell használnunk.

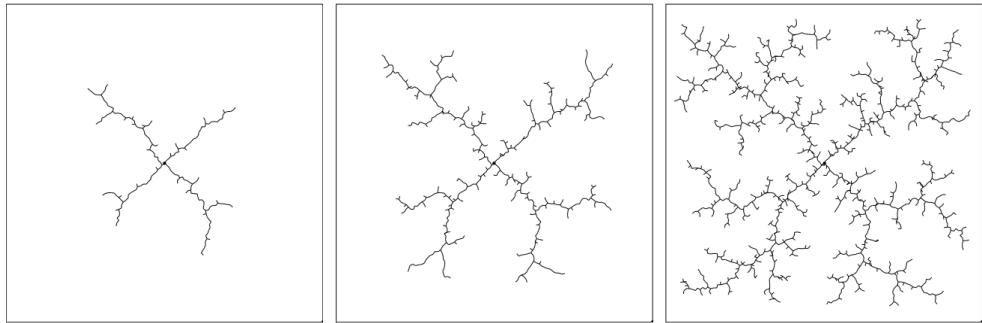
2.6. Új globális tervező

A fejlesztés során a fentebb felsorolt problémák miatt a globális tervezőt lecseréltem az RTR nevű algoritmusra. Ez is Kiss Domokos munkája [8], az implementációját Nagy Ákos végezte el [9].

²Intel Core 2 Duo E8400 @ 3.0GHz, 4GB RAM

2.6.1. RRT

A globális tervezők sok esetben topologikus gráfokat (speciális esetben fákat) használnak a konfigurációs tér struktúrájának leírásához [10]. A szakirodalomban egyik leggyakrabban használt ilyen algoritmus a *Rapidly Exploring Random Trees* [11]. Ennek a lényege, hogy a kezdeti konfigurációból egy fát építünk a szabadon bejárható konfigurációs térben. A fa csomópontjaiban konfigurációk találhatóak, és a fa terjesztését úgy irányítjuk, hogy a kívánt célkonfiguráció felé tartson. Ha a fa ténylegesen eléri a célkonfigurációt, akkor az utat a kezdeti konfigurációból a célkonfigurációba már könnyedén megkaphatjuk.

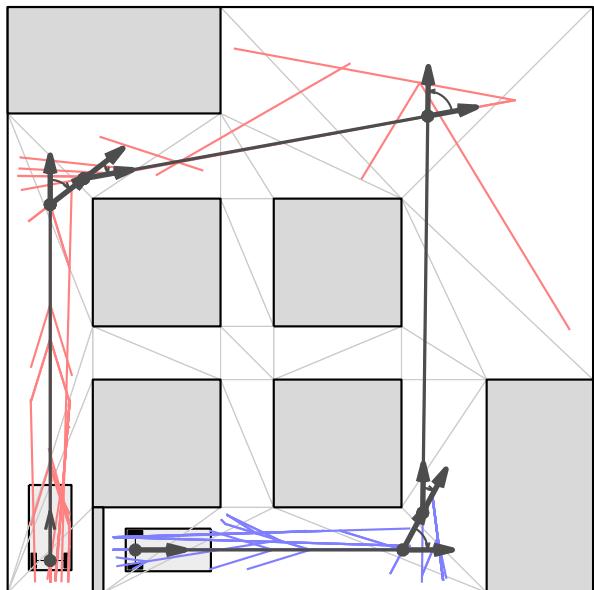


2.8. ábra. Az RRT algoritmus három különböző iterációnál [11].

2.6.2. RTR

Az *Rotate-Translate-Rotate* algoritmus a fentebb látott RRT algoritmus egy módosított változata. Elsődlegesen differenciális robotok számára tervez pályát, de jó alapot szolgáltat a C*CS algoritmusnak is. Nevét a benne használt mozgási primitívekről kapta, azaz a R , mint fordulás, a T pedig az egyenesen haladást jelöli. Az RRT algoritmustól eltérően itt az algoritmus két fát épít, egyet a kezdő, egyet pedig a célkonfigurációból. Másik fontos különbség, hogy ütközés esetén a pályát minden irányba továbbterjeszti, így növelte a megoldás megtalálásának lehetőségét.

Ez az algoritmus azért előnyös, mert a celladekompozíciós eljárás hibát kiküszöböli. Tehát két szűk folyosó találkozásánál csak akkor ad megoldást, ha van hely az elfordulásra. Mivel a keresés során két fát épít, így a párhuzamos parkolás esetén is segíti a megoldás megtalálását.



2.9. ábra. Az RTR algoritmus; piros a kezdő, kék a célkonfigurációból indított fa. [9]

3. fejezet

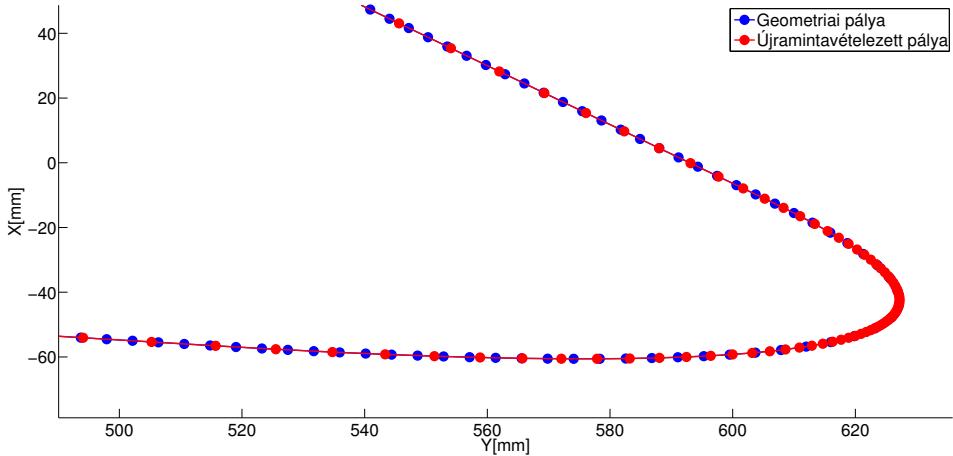
Pálya időparaméterezése

A pályatervező által elkészített ütközésmentes pálya nem tartalmaz semmilyen idővel kapcsolatos információt. Ebben a fejezetben a pálya pontjaihoz sebesség értékeket rendelünk hozzá. Ezt a többlet információt a pályakövető algoritmus használja fel, hogy mozgás során a robot kinematikai korlátai ne okozzanak problémát. Tehát az időparaméterezés elősorban a robot korlátait használja fel, de arra is alkalmas, hogy meghatározzuk a pálya bejárásának idejét.

A módszert Nagy Ákos dolgozta ki differenciális robotokhoz, majd módosítottam autószerű robotok számára [9]. Az elveket végül úgy alakítottuk ki közösen, hogy az ne függjen a robot típusától, így az alkalmazható legyen bármilyen kerekeken mozgó robottípusra.

3.1. Időparaméterezés

Az időparaméterezés két fő lépésből áll. Elsőként a kapott geometriai pályához sebesség értékeket rendelünk hozzá, majd ezután újramintavételezzük a pályát. Az újramintavételezés után a pálya időben egyenletes lesz, tehát az egymást követő pályapontok között azonos idő telik el. A mintavételezés idejét a pályakövető algoritmus mintavételi ideje határozza meg. A geometriai pályát általában távolságban egyenletesen mintavételezzük, de ez nem kötelező feltétel az időparaméterezéshez.



3.1. ábra. A pálya időparaméterezése.

A szakirodalomban nem sok időparaméterezéssel kapcsolatos munka található. Egy hasonló megközelítést Christoph Sprunk munkájában olvashatunk [12]. A legfontosabb eltérés, hogy Sprunk külön korlátozza a robot tangenciális és centripetális gyorsulását, míg mi a robot kerekeinek eredő gyorsulását korlátozzuk. Ez a megoldás a valóságot jobban közelíti, hiszen attól, hogy a gyorsulás két komponense a korlátok alatt marad, nem biztos, hogy az eredő gyorsulás sem haladja meg a korlátot.

Az időparaméterezés során nem használjuk ki az előző fejezetekben bemutatott pályatervező által tervezett pálya speciális tulajdonságait, a célunk egy olyan algoritmus készítése, amely tetszőleges geometriai pályából képes sebesség információval ellátott, időben egyenletes mintavételű pályát készíteni. Emiatt nem építhetünk a pályatervező által használt geometriai elemekre (körív, egyenes), és ezek speciális tulajdonságaira.

Általános esetben nem tudjuk analitikusan meghatározni a pálya görbületét, így görbület becslést kell alkalmaznunk [13]. Természetesen abban az esetben, ha a pályatervező rendelkezik már a pálya görbületével, az időparaméterező algoritmus azt fogja használni a becslés helyett, mivel így pontosabb eredményt érhetünk el, és gyorsítja is a végrehajtást.

3.2. Jelölések

Ebben a fejezetben a (3.1) táblázatban megadott jelöléseket fogjuk használni. Azokban az esetekben, ahol fontos megkülönböztetni a geometriai pályát és az (újra)mintavételezett pályát, ott a felső indexben található g betű a geometriai pályát jelöli, az s betű pedig a mintavételezett pályát. A pálya pontjait 1-től számozzuk.

$$\begin{aligned}
\Delta t(k) &: \text{A } k \text{ és a } k+1 \text{ pontok között eltelt idő} \\
t(k) &: \text{A } k. \text{ pontban az addig eltelt idő} \\
\Delta s(k) &: \text{A } k \text{ és a } k+1 \text{ pontok közti távolság} \\
s(k) &: \text{A } k. \text{ pontban az addig megtett távolság} \\
v(k) &: \text{A } k. \text{ pontban a robot sebességének nagysága} \\
\omega(k) &: \text{A } k. \text{ pontban a robot szögsebességének nagysága} \\
a_t(k) &: \text{A } k. \text{ pontban a robot tangenciális gyorsulásának nagysága} \\
c(k) &: \text{A } k. \text{ pontban a görbület nagysága} \\
N &: \text{A pálya pontjainak száma} \tag{3.1}
\end{aligned}$$

Azokat az eseteket külön jelöljük, amikor a robot egyik kerekére vonatkozó mennyiségekről beszélünk. Első indexben azt, hogy első (f) vagy hátsó (r) kerék, utána, hogy bal (l), jobb (r), kerékről van szó. Ezenkívül a kereknél megkülönböztetjük, hogy tangenciális (a_t), centripetalis (a_c) vagy eredő (a) gyorsulásról beszélünk.

Fontos megjegyezni, hogy a $\Delta s(k)$ távolságot úgy kell értelmezni, hogy a $k.$ és $k+1.$ pont között egy körív található, és az ezen mért távolság lesz $\Delta s(k)$. A körívet a $c(k)$ görbület határozza meg. Ha nem köríveket használnánk, hanem egyenesen kötnénk össze a pályapontokat, akkor a görbületnek szükségszerűen nullának kellene lennie.

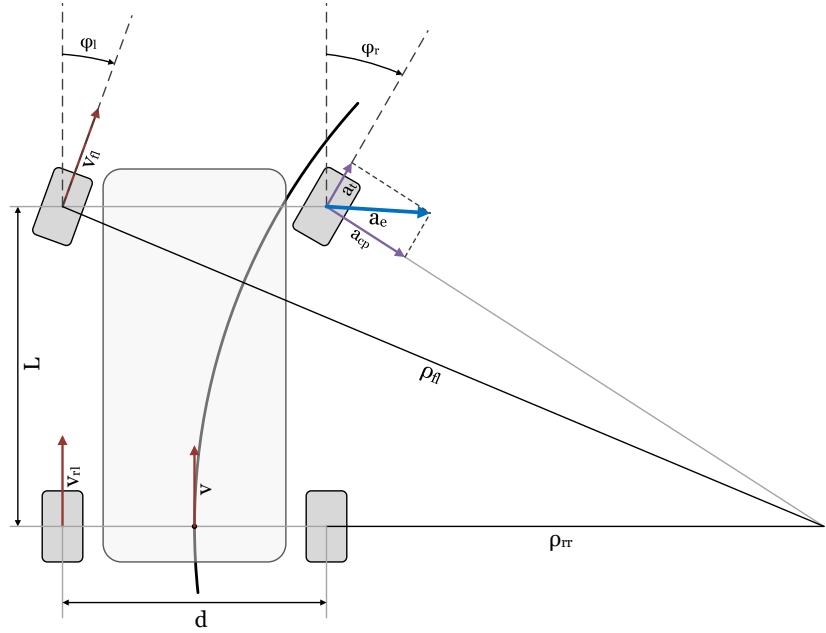
3.3. Korlátozások

A robot mozgását általános esetben a 3.2. ábra mutatja be. Az időparaméterezés során figyelembe vesszük a robot pályamenti sebességét és szögsebességét valamint a robot kerekeinek tangenciális és eredő gyorsulását.

A sebességről létrehozásakor rendelkezésünkre áll a pálya görbülete, azaz az autó referenciaPontja által bejárt kör sugara (ρ). Ebből a hátsó kerekek fordulókörének sugara a következőképpen számolható:

$$\begin{aligned}
\rho_{rl} &= \rho - \frac{d}{2} \\
\rho_{rr} &= \rho + \frac{d}{2}. \tag{3.2}
\end{aligned}$$

Fontos megemlíteni, hogy itt ρ előjeles sugarat jelöl. Ami esetünkben pozitív körülfordulási irányban pozitív (balra fordulás). Az első kerekek esetén nincs ilyen könnyű dolgunk. Ahhoz hogy az első kerekek fordulás közben ne csússzanak meg oldalirányba, a két keréknél különböző szögben kell elfordulnia, ezt nevezzük Ackermann-hajtásnak. Ez azzal áll kapcsolatban, hogy a kerekeknek különböző köríven kell elfordulniuk. Hogy ezt kiszámítunk, először írjuk fel az összefüggést a sugár és a kormányszög között, kerékpár modellt feltételezve. Ehhez használjuk fel az (1.2) egyenletet:



3.2. ábra. Autószerű robot mozgása köríven

$$\rho = \frac{L}{\tan \phi} \quad (3.3)$$

a sugár előjele konzakvens a kormányszög előjelével. Innen az Ackermann-hajtás szerint a kerekek kormányszöge:

$$\begin{aligned} \phi_l &= \arctan \left(\frac{L}{\rho - \frac{d}{2}} \right) \\ \phi_r &= \arctan \left(\frac{L}{\rho + \frac{d}{2}} \right) \end{aligned} \quad (3.4)$$

ezek és a (3.2) egyenletek segítségével számítható az első kerekek által bejárt körök sugarai:

$$\begin{aligned} \rho_{fl} &= \frac{\rho_{rl}}{\cos \phi_l} \\ \rho_{fr} &= \frac{\rho_{rr}}{\cos \phi_r} \end{aligned} \quad (3.5)$$

A különböző keréksebességek arányosak a referencia pont sebességével, ez az arány pedig a sugarak arányaival írható fel:

$$p(k) = \frac{\rho_w(k)}{\rho(k)} = \frac{v_W(k)}{v(k)} \quad (3.6)$$

ahol w jelzi a kerékre vonatkozó mennyiségeket. Látható, hogy ha a maximális keréksebességet keressük, akkor elegendő csak a maximális abszolút értékű fordulókör sugarát megkeresni. Az (3.2) egyenletek alapján következik, hogy ez mindenkor a külső kerék esetén teljesül. Az (3.5) egyenletek alapján pedig belátható, hogy az első kerekek sebessége mindenkor nagyobb lesz a hátsóéknál, mivel $|\cos \phi| \leq 1$. Azaz jobbra kanyarodás esetén a bal első kerék sebessége a legnagyobb és vice versa.

A sebességprofil készítésekor egy adott robot esetében ezekre a mennyiségekre határozunk meg korlátozásokat:

$$v^{max} : \text{A robot pályamenti sebesség korlátja} \quad (3.7)$$

$$\phi^{max} : \text{A robot maximális kormányszöge, mint fizikai korlátozás}$$

$$\omega^{max} : \text{A robot maximális szögsebessége}$$

$$a_w^{max} : \text{A robot egy kerekének maximális gyorsulás korlátja} \quad (3.8)$$

A gyorsulás korlátját elég egyetlen kerékre meghatározni, feltéve, hogy a kerekek tapadási tényezője megegyezik. A kerekek maximális eredő gyorsulását a tapadási súrlódási együttható (μ_{tap}) határozza meg, amelynél a robot kerelei még nem csúsznak meg. A maximális gyorsulás és a tapadási együttható között a következő egyszerű összefüggés áll fent:

$$a_{max} = \mu_{tap_{max}} \cdot g, \quad (3.9)$$

ahol g a nehézségi gyorsulás. Írjuk fel egy kerék gyorsulását:

$$a(k) = \sqrt{a_{wc}(k)^2 + a_{wt}(k)^2} \leq g \cdot \mu_{tap}, \quad (3.10)$$

ahol $a_{wc}(k)$ egy kerék centripetális gyorsulása és $a_{wt}(k)$ a tangenciális gyorsulása. Az (3.10) egyenletben azzal a feltevéssel élünk, hogy a robot kerelei és a talaj között a tapadási súrlódási együttható állandó és nem függ az erő irányától. Az általunk használt robotoknál ez a közelítés megengedhető, mivel a gumikerekek homogénnak tekinthetők. Ha barázdákat tartalmaznának, akkor már nagyobb eltérést okozna ez a közelítés.

Fontos megjegyezni, hogy a kerékgyorsulás korlátokat lassulásnál is alkalmazzuk. Tehát a kerék gyorsulásának abszolút értékét korlátozzák ezek a megkötések. Így azt tételezzük fel, hogy a kerekek viselkedése gyorsulás és lassulás esetében megegyezik. A robot sebességénél

viszont nem engedünk negatív értékeket, a robot végig előre haladhat. A tervező viszont megadhat olyan pályát ahol tolatnia kell a robotnak, vagy egy helyben megfordulnia, de ezt a pályatervező algoritmus kezeli.

3.4. Geometriai sebességprofil

Első lépésként a geometriai pályapontokhoz rendelünk a korlátoknak megfelelő sebességeket és a későbbiekben ezt a sebességprofilt használjuk fel a pálya újramintavételezéséhez.

A pályamenti sebességeket úgy határozzuk meg, hogy a robot pályamenti gyorsulása a lehető legnagyobb legyen. Ezt megtehetjük úgy, hogy a robot kerekeinek tangenciális gyorsulását maximalizáljuk, azonban több hatás miatt nem tudjuk a kerekek gyorsulását folyamatosan növelni.

Egyrészt a robot sebesség és szögsebesség korlátját nem sérthetjük meg. Ebből a két korlátból a pálya minden pontjára kiszámolhatunk egy maximális sebességet függetlenül az előző pályapont sebességétől:

$$v^{max}(k) = \min \left(v^{max}, \frac{\omega^{max}}{c(k)} \right) \quad (3.11)$$

A szögsebesség korlátját autószerű robot esetén számíthatjuk a maximális sebesség és a minimális fordulókör sugár segítségével, természetesen ebben az esetben az egyenlet leegyszerűsödik: $v^{max}(k) = v^{max}$. Valamint a kerekek centripetális gyorsulása nem haladhatja meg az előírt eredő gyorsuláskorlátot, különben a robot kereke megcsúszna. A pálya adott k . pontjában a kerekek centripetális gyorsulását a következőképpen számolhatjuk ki:

$$a_{wc}(k) = v_w(k)^2 \cdot c(k) = v(k)^2 \cdot c(k) \cdot p(k), \quad (3.12)$$

ahol p tehát a referenciapont és a maximális fordulási körrel rendelkező kerék sugarának aránya. Fontos ezen kívül megjegyezni, hogy mivel a robot gyorsulását határozzuk meg a k . pontban, így a $v(k)$ már rendelkezésünkre áll a $k - 1$. pontban számított gyorsulásból. Amennyiben a kiszámolt centripetális gyorsulások már önmagukban is meghaladják az előírt eredő gyorsuláskorlátot, úgy $v(k)$ értékét addig kell csökkenteni, hogy a centripetális gyorsulás az eredő gyorsuláskorlátot már ne haladja meg. Ezután a kerekek tangenciális gyorsulását (3.13) egyenlet alapján határozhatjuk meg.

$$a_{wt}(k) = \sqrt{(a_{max})^2 - a_{wc}(k)^2} \quad (3.13)$$

Miután kiszámoltuk, hogy az adott pályapontnál mekkora legyen a robot kerekeinek tangenciális gyorsulása már könnyedén számolható a robot gyorsulása és sebessége:

$$a_t(k) = \frac{a_{wt}(k)}{p(k)} \quad (3.14)$$

$$v(k+1) = \min \left(v^{max}(k+1), \sqrt{v(k)^2 + 2 \cdot a_t(k) \cdot \Delta s_c(k)} \right) \quad (3.15)$$

Profil visszaterjesztés

Két esetben előfordulhat, hogy az előző pályaponthoz meghatározott sebességértéket módosítani kell. Egyszerűt, ha a centripetalis gyorsulás önmagában meghaladja a megengedhető maximális gyorsulást, másrészt, ha a (3.15) egyenletben megsérüljük a robot gyorsuláskorlátját. Az előbbi eset a kanyar előtti fékezést fogja meghatározni, utóbbi a pálya végén lévőt, hisz ott előírjuk, hogy $v^{max}(N) = 0$. Ha nem tennénk meg a módosítást, akkor a fékezés következetében fellépő gyorsulás (lassítás) abszolút értéke meghaladhatná az megengedettet, hiszen az előző pontokban nem tudtuk, hogy lassítani kell a robotnak. Ezt a módosítást hívjuk a profil visszaterjesztésének, mivel itt addig kell visszafelé haladva ellenőrizni, amíg a kiszámolt értékek már nem sértik meg a korlátokat.

Ahhoz hogy ezt megtegyük, kezdetnek számoljuk ki, hogy a megváltozott sebesség következetében mekkora lesz a leginkább terhelt kerék tangenciális gyorsulása.

$$a_{wt}(k) = \frac{v_w(k+1)^2 - v_w(k)^2}{2 \cdot \Delta s_w(k)} = \frac{v(k+1)^2 - v(k)^2}{2 \cdot \Delta s(k)} \cdot p(k) \quad (3.16)$$

Amennyiben a kapott tangenciális gyorsulás megséríti a gyorsulásra vonatkozó korlátot az előző pont sebességét is csökkentenünk kell. Ehhez használjuk fel a (3.13) és a (3.16) egyenleteket:

$$\begin{aligned} a_{wt}(k) &= \frac{v(k+1)^2 - v(k)^2}{2 \cdot \Delta s(k)} \cdot p(k) = \sqrt{(a_{max})^2 - a_{wc}(k)^2} \\ &= \sqrt{(a_{max})^2 - ((v(k) \cdot p(k))^2 \cdot c(k))^2} \end{aligned} \quad (3.17)$$

ezt kifejezve $v(k)$ -ra a lentebb látható negyedfokú egyenletet kapjuk:

$$\begin{aligned} d(k) &= \frac{p(k)^2}{4 \cdot \Delta s(k)^2} + c(k) \cdot (p(k))^2 \\ e(k) &= -\frac{2 \cdot v(k+1)^2 \cdot p(k)^2}{4 \cdot \Delta s(k)^2} \\ f(k) &= \frac{v(k+1)^4 \cdot p(k)^2}{4 \cdot \Delta s(k)^2} - a_{max}^2 \\ 0 &= v(k)^4 \cdot d(k) + v(k)^2 \cdot e(k) + f(k). \end{aligned} \quad (3.18)$$

A (3.18) egyenlet valós, pozitív megoldásait keressük. Felmerülhet a kérdés, hogy mi garantálja, hogy minden lesz ilyen megoldás. A Viète-formula felírásával belátható, hogy

mindig pozitív megoldása van az egyenletnek, a másodfokú egyenlet diszkriminánsának felírásával pedig, hogy lesz valós megoldás. Amennyiben több pozitív valós megoldása van az egyenletnek, akkor a legnagyobb megoldást választjuk. Végül erre az értékre kell módosítanunk a sebességet, majd visszalépni, hogy ezzel a módosítással nem sértünk-e újabb értéket¹.

A visszaterjesztés során a sebesség és szögsebesség korlátokkal nem kell foglalkoznunk, hiszen minden esetben, mikor módosítjuk a sebességet, csökkentjük az értékét.

3.5. Újramintavételezés

Miután elkészítettük a geometriai pályához tartozó sebességprofilt, létrehozzuk a véleges pályát, amit majd a pályakövető egység bemenetként megkap. Ez a véleges pálya már időben egyenletesen lesz mintavételezve (mintavételezett pálya). Ehhez először számoljuk ki az eltelt időt a geometriai pálya mentén, amelynek alapja, hogy két pályapont között a robot állandó gyorsulással halad.

$$\Delta t^g(k) = \frac{2\Delta s^g(k)}{v^g(k) + v^g(k+1)} \quad (3.19)$$

$$t^g(k+1) = t^g(k) + \Delta t^g(k) \quad (3.20)$$

A következő lépésben meghatározzuk, hogy az újramintavételezett pályánk hány pontból álljon. Ezt könnyedén megtehetjük, hiszen adott számunkra a kívánt mintavételezi idő(t_s). Így a következő képlet adódik a mintavételezett pályához tartozó pontok számának meghatározására:

$$N^s = \lceil t^g(N^g)/t_s \rceil + 1 \quad (3.21)$$

A pontok számába beleérjük a kezdő és végpontot is. A (3.21). egyenletből következik, hogy amennyiben $t(N^g)$ és t_s nem egymás többszörösei, a mintavételezett pálya utolsó pontjához olyan időpont tartozik, amely nagyobb mint $t(N^g)$. A pálya végpontját még a későbbiekben tárgyaljuk, akkor visszatérünk erre az eltérésre is.

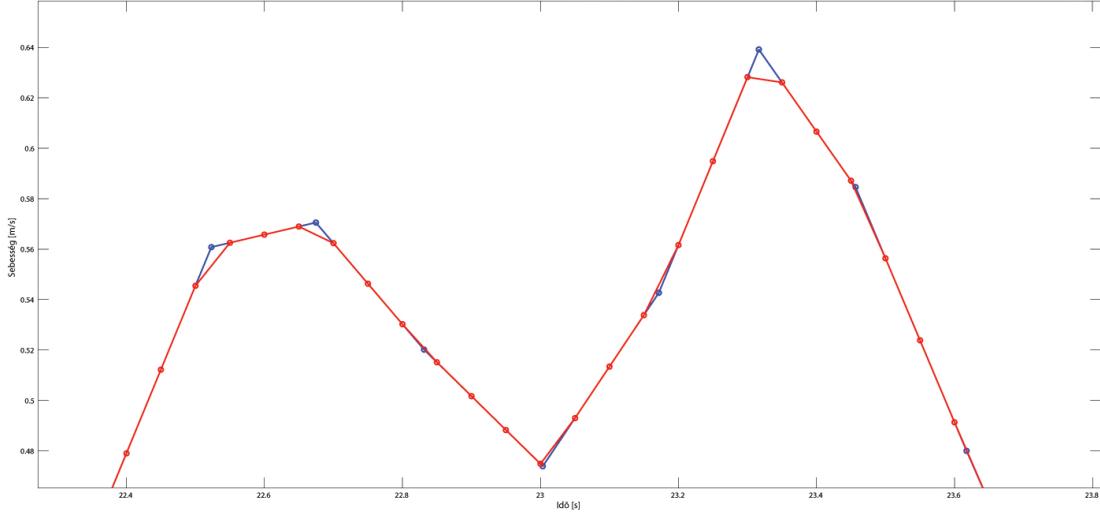
Most pedig határozzuk meg a mintavételezett pálya pontjaiban a sebességet. Ezt a geometriai pálya alapján tesszük, figyelembe véve, hogy a mintavételezett pálya esetén is két pont között állandó gyorsulást feltételezünk. A számítás egy egyszerű lineáris interpolációt valósít meg:

$$v^s(k) = v^g(j) + v^g(j+1) \cdot it(k) \quad (3.22)$$

$$it(k) = \frac{t^s(k) - t^g(j)}{t^g(j+1) - t^g(j)}, \quad (3.23)$$

¹Általános esetben bevezethető egy tangenciális gyorsulás korlát is, de az autószerű robot esetén ezt nem alkalmaztuk.

ahol j jelöli a legkisebb indexet amelyre teljesül, hogy $t^s(k) < t^g(j)$. A lineáris interpoláció miatt teljesül az a feltétel, hogy két pont között állandó gyorsulással mozogjon a robot.



3.3. ábra. A geometriai (kék) és mintavételezett(piros) sebességprofil.

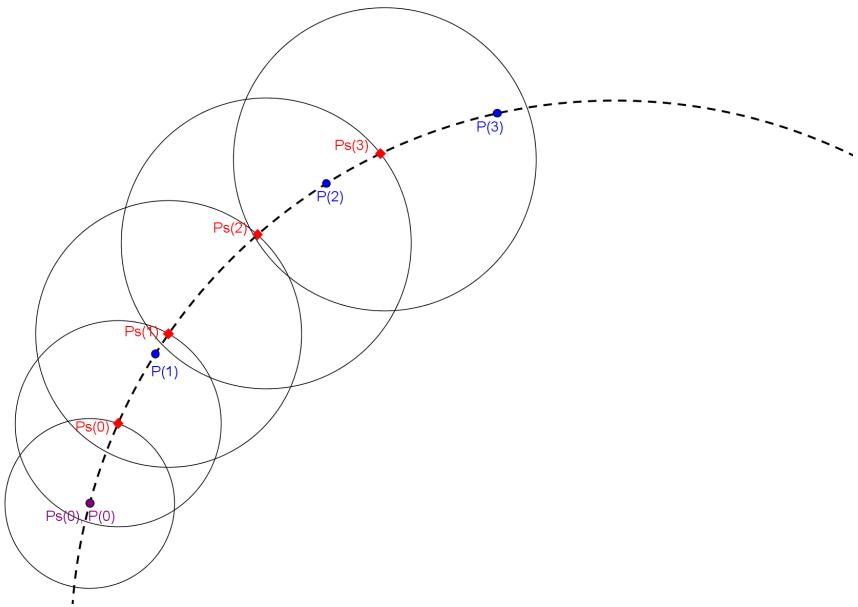
A kiszámított sebességprofil alapján könnyedén adódik a megtett út is:

$$\Delta s^s(k) = \frac{v^s(k) + v^s(k+1)}{2} \cdot t_s \quad (3.24)$$

$$s^{s+1}(k) = s^s(k) + \Delta s^s(k) \quad (3.25)$$

Így már rendelkezésünkre áll a robot kívánt sebessége, a megtett út, valamint az idő a mintavételezett pálya összes pontjában. Már csupán a pályapontjainak koordinátáit kell ezek alapján meghatároznunk.

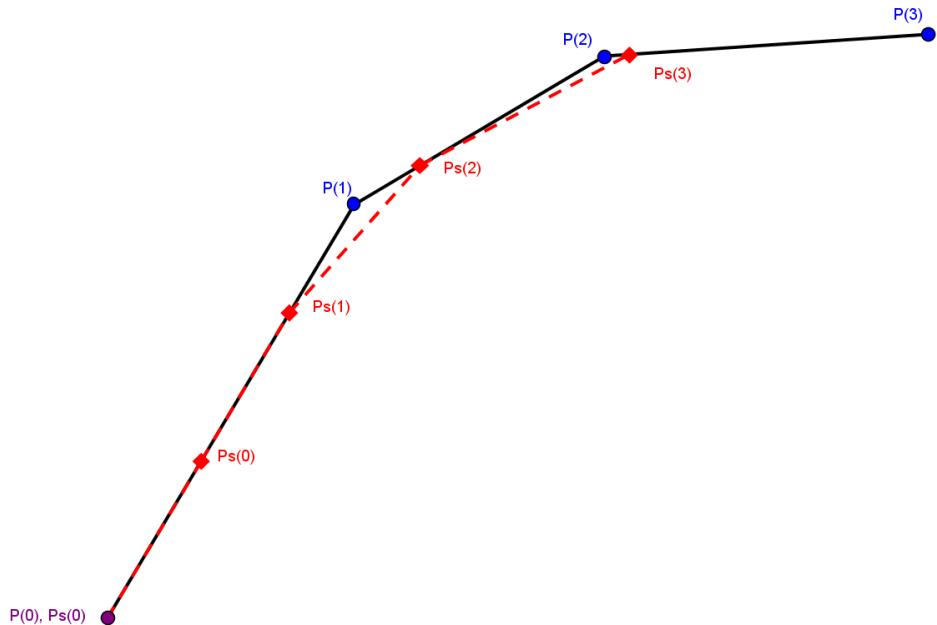
Mivel ismerjük a pályapontok közötti távolságot ($\Delta s^s(k)$), iteratív eljárással az előző pályapont koordinátái alapján az aktuális pontról tudjuk, hogy egy körön helyezkedik el. További feltételünk, hogy a pont az eredeti, geometriai pályán rajta legyen. Ha vesszük a geometriai pálya pontjai közötti görbületből adódó köríveket, akkor az ívek és a kör metszéspontjai közül kell kiválasztanunk a keresett pontot. A kiválasztás egyszerű, ha megjegyezzük, hogy az előző pontnál melyik szakasz alapján találtuk meg a pontot, így csak attól a szakasztól kezdve kell keresni a metszéspontokat. Az algoritmus menete látható a 3.4. ábrán. minden vizsgált szakasznál arra kell figyelni, hogy a metszéspont a szakasz határpontjai között helyezkedjen el. Az első szakasz vizsgálatánál még az is fontos, hogy az előző pont előtti metszéspontot ne vegyük figyelembe. Az ábrán a $Ps(1)$ pontban, ezért nem választhatjuk a másik metszéspontot. A legelső mintavételezett pontot a geometriai pálya első pontjába helyezzük el.



3.4. ábra. A mintavételezett pontok meghatározása. $P(x)$ a geometriai pálya pontjait jelöli, $Ps(y)$ pedig a keletkező mintavételezett pályát.

Mintavételezett pálya végpontja

Az lenne az optimális eset, ha a mintavételezett pálya utolsó pontja egybeesne az eredeti pálya végpontjával, ahogyan a kezdőpontjaik ténylegesen egybeesnek. Alapvetően úgy hozzuk létre a mintavételezett pályát, hogy az a geometriai pálya sebességsprofiljának megfeleljen, ez viszont nem garantálja az előző feltétel teljesülését.



3.5. ábra. A mintavételezett pontok meghatározásánál keletkező hiba.

Három hatás azt eredményezi, hogy nem fog teljesülni ez a feltétel a pálya utolsó pontjára:

1. Ahogy már említettük korábban, nem biztos, hogy a két pályát ugyanannyi idő alatt járja be a robot. Ez maximum t_s időkülönbséget okozhat, és minden esetben távolabbi végpontot eredményez, mint az eredeti végpont.
2. A mintavételezett pálya sebességprofiljának elkészítésekor nem tökéletesen követi az eredeti sebességet a robot a mintavételezésből adódóan. Ez látszik a 3.3. ábrán is. A hiba megegyezik a két görbe alatti terület közötti különbséggel, ami okozhat távolabbi és közelebbi végpontot is.
3. A harmadik hiba a koordináták meghatározásánál keletkezik. Ez a hatás is minden távolabbi végpontot okoz.

A legtöbb esetben célszerű, ha a végpontok egybeesnek, így ezt a mintavételezett pálya meghatározásánál biztosítunk kell. Ha egyszerűen az utolsó pályapontot az eredeti pálya végpontjába tessziük, nem biztos, hogy betartjuk a robot gyorsulás korlátait, így más módszerhez kell folyamodnunk.

Az általunk használt algoritmus lényege, hogy a sebességprofilnak egy részét egy adott sebességgel eltoljuk úgy, hogy a két pálya végpontja pontosan egybeessen. Az eltolás mértékét (Δv_{corr}) a következő képlettel kapjuk meg:

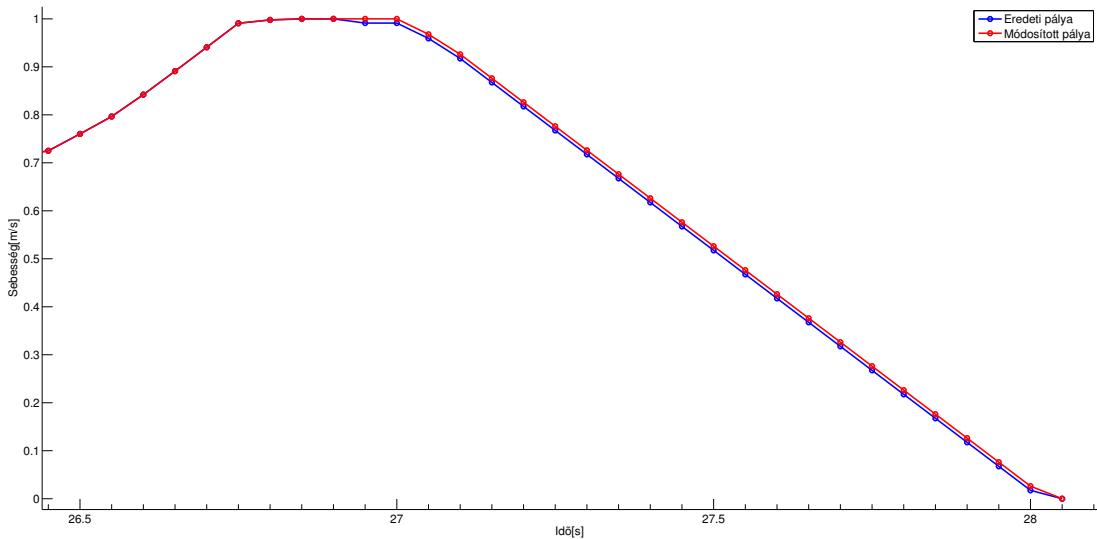
$$\Delta v_{corr} = \frac{\Delta s_{corr}}{t_s \cdot n}, \quad (3.26)$$

ahol Δs_{corr} a mintavételezett és a geometriai pálya végpontjai közötti távolság előjelesen. Ha a mintavételezett pálya utolsó pontja van távolabb, akkor negatív a távolság, különben pozitív. n pedig azoknak a sebességpontoknak a száma, amiket eltolunk.

A (3.26) egyenlet egyszerűen belátható, ha felírjuk az eltolásból adódó területkülönbséget. A Δs_{corr} útkülönbséget azért kell előjelesen megadnunk, hogy minden esetben használható legyen az algoritmus, akkor is, ha a mintavételezett pálya végpontja van távolabb és akkor is ha a geometriai pályáé.

A továbbiakban meghatározzuk azokat a sebességpontokat, amelyeket Δv_{corr} sebességgel eltolunk. Mivel a megváltozott sebességponthoz tartozó koordinátákat újra ki kell számolnunk, így minél kevesebb pontot szeretnénk eltolni a sebességprofilon. Viszont a sebesség és gyorsulás korlátokat be kell tartanunk, így nem tolhatunk el tetszőlegesen kevés pontot.

Vizsgáljuk külön a két alapesetet Δs_{corr} előjele alapján. Kezdjük azzal az esettel amikor Δs_{corr} negatív, tehát a mintavételezett pálya végpontja van távolabb (3.6. ábra). Ekkor a módosítandó szakasz kezdő pontjához tartozó gyorsulásnak pozitívnak kell lennie, hiszen mi csökkenteni fogjuk a soron következő pont sebességét, és ha a gyorsulás pozitív vagy nulla, akkor csökken a robot gyorsulása a szakasz kezdőpontjában. Ha a gyorsulás negatív lenne a kezdőpontban, akkor könnyedén előfordulhat olyan eset, hogy a sebességcsökkentés után megszegjük a gyorsulás korlátot.

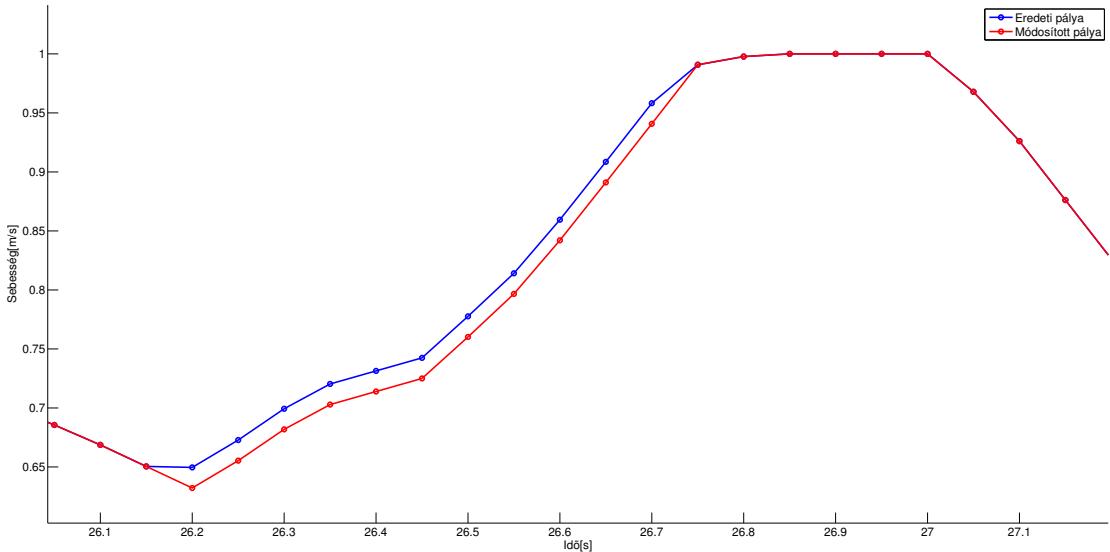


3.6. ábra. A módosított mintavételezett sebességprofil ha Δs_{corr} negatív.

A szakasz végpontjánál pedig negatív gyorsulás szükséges, hiszen a következő pont gyorsulása meg fog nőni a módosítás hatására, és ha pozitív lenne a gyorsulás, a gyorsulásra vonatkozó korlátunkat könnyedén megszegnénk.

Tehát a legegyszerűbb esetben a szakasz kezdőpontja a pálya végén található lassító szakasz eleje, mielőtt lassítani kezd a robot és a végpontja a pálya utolsó előtti pontja. Ennek a szakasznak a pontjait fogjuk a (3.26). egyenletből adódó Δv_{corr} sebességgel csökkenteni, és így a robot pontosan a geometriai pálya végpontjában áll meg.

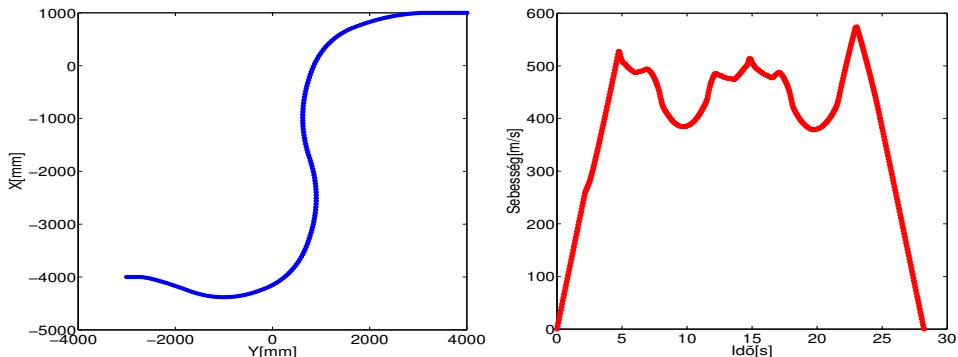
A másik eset, amikor Δs_{corr} pozitív, tehát a mintavételezett pálya végpontja messzebb van a geometriai pálya végpontjához képest. Ekkor – mivel meg fogjuk növelni a szakasz sebességét – pont fordítva kell szakaszt választanunk; a kezdőpontjánál negatív gyorsulás szükséges, a végpontjánál pedig pozitív. Így kerülhető el leginkább a gyorsulás korlát megszegése. Ebben az esetben pedig egy ilyen megfelelő szakasz a pályán található utolsó gyorsító rész.



3.7. ábra. A módosított mintavételezett sebességprofil ha Δs_{corr} pozitív.

Abban az esetben, ha valamiért az előbb leírt triviális szakaszok mégsem jók, másik szakaszt kell választanunk. Első lépésként válasszunk ki egy megfelelő végpontot a keresendő szakaszhoz. Ha Δs_{corr} negatív, akkor megfelelő választás a pálya utolsó előtti pontja, ha pozitív, akkor pedig a pálya utolsó olyan pontja, ahol a gyorsulás pozitív. Ezután keresünk ehhez a kiválasztott végponthoz egy kezdőpontot, de most már vegyük figyelembe a robot korlátozásait és természetesen azt, hogy az útkülönbség az előírt Δs_{corr} legyen. Miután megkaptuk a kezdőpontot is, akkor még ellenőriznünk kell, hogy a végpontnál a robot korlátozásait nem sértjük-e meg. Ezt az első lépésben nem tudtuk megtenni, mivel nem ismertük a végpontot, így Δv_{corr} értékét sem. Ha a végpont megséríti a korlátokat, új végpontot kell keresnünk és ahhoz új kezdőpontot. Ezt addig kell folytatnunk, amíg a robot korlátozásait betartjuk.

Miután a módosított sebességprofil elkészült a szakasz elejétől kezdve újra kell számolnunk a mintavételezett pálya koordinátáit. Ez teljesen ugyanúgy történik, ahogyan már egyszer megkaptuk a mintavételezett pályát. Azért volt fontos, hogy a lehető legkevesebb sebességpontot toljuk el, hogy a koordináták újraszámlálását is kevesebb pontnál kelljen megtenni.



3.8. ábra. A pálya és a mintavételezett sebességprofil autószerű robot esetén

Habár a fenti iteratív eljárás hosszadalmasnak tűnik, vegyük figyelembe, hogy általában kis távolságot kell kompenzálnunk, amihez kis sebességgkülönbség tartozik. Ebből adódóan nagy valószínűséggel a triviális szakasz is megfelelő lesz számunkra.

Szintén fontos megjegyezni, hogy mivel a tárgyalt három hatás elsősorban negatív Δs_{corr} -t eredményezz, így a gyakorlatban ez az eset fordul elő. A gyakorlatot tekintve még említeni kell, hogy a Δs_{corr} nagyságrendje igencsak csekély a pálya teljes hosszához képest, nehezen elképzelhető akárcsak 1%-ot meghaladó arány a teljes pálya hosszához képest.

4. fejezet

Pályakövető szabályozás

Ebben a fejezetben az elkészült, időben egyenletesen mintavételezett pálya követésének problémáját, majd az ehhez készült szabályozási algoritmusokat mutatom be.

4.1. Pályakövetés

Autószerű robot esetén a követendő pályát szakaszokra bontjuk fel. Egy szegmensen belül a robot megállás nélkül halad előre vagy hátra a pálya mentén. Ebből következik, hogy egy szegmens a haladási irányból, és a pálya időben egyenletesen mintavételezett pontjaiból áll. Két szakasz között a robotnak nem szükséges semmilyen speciális feladatot végrehajtani, közvetlenül folytatja a következő szegmens végrehajtásával¹. Ilyenkor van lehetőség az irány módosítására. Ebből következik, hogy a teljes útvonal szegmensekre bontását irányváltoztatások jelzik, az azonos irányú pályaelemeket, mint például amik a C*CS esetén is keletkezhetnek, egy szakaszba egyesíti az algoritmus.

A pályakövető szabályozás alapvetően két szintre oszlik, ahogyan ez a 4.1. ábrán is látható. A felsőbb szinten a pályába kódolt sebesség és pozíció követése a cél, a robot aktuális pozíciója alapján, míg az alsóbb szinten az így számolt sebesség és kormányszög alapjel szabályozása történik. Ez a szabályozás az algoritmusok szintjén is elkülönül, míg az alacsonyabb szintű szabályozások magán a roboton, a vezérlésén (esetleg szimulátorán) végrehajthatók, addig a magasabb szintű szabályozás külön hajtódiagramról végrehajthatók.

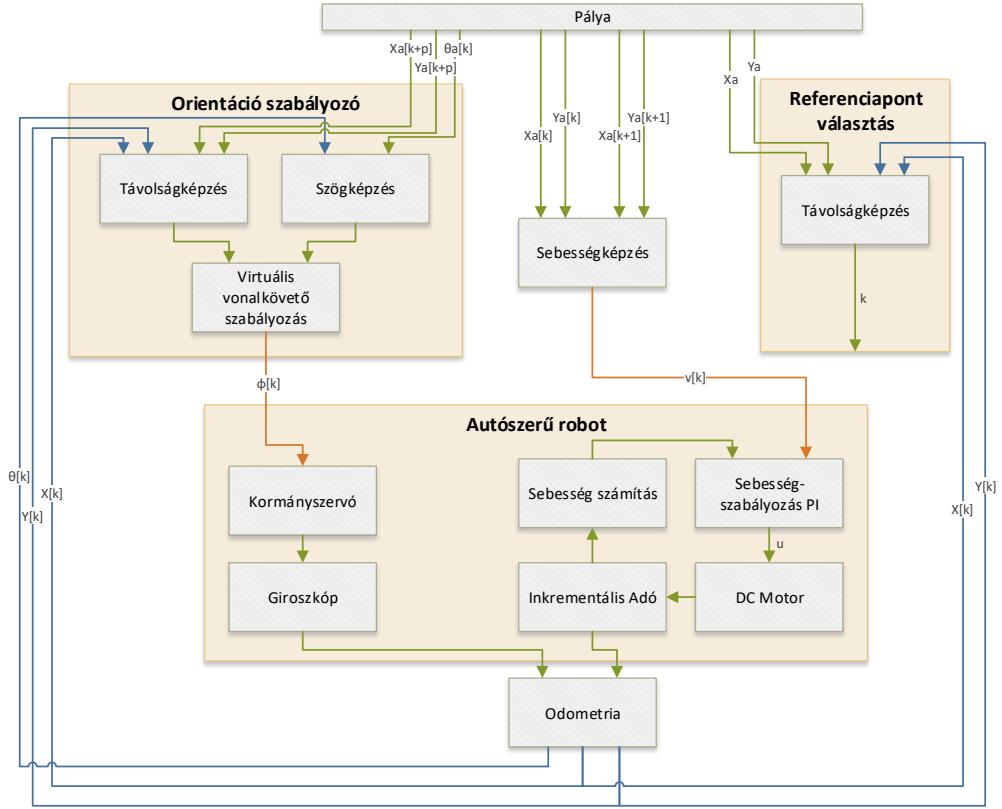
A pályakövetés alapja a szétcsatolt sebesség- és orientációs szabályozás. A szétcsatolás következménye, hogy egyszerűbb, lineáris szabályozókat használhatunk a követés során. Hárnya, hogy ez a modell nem minden esetben követi megfelelően a valóságot.

4.1.1. Sebességszabályozás

Alacsony szinten a robotsebességénél történik meg a sebességszabályozás. Az általam használt valós robot négy kerék meghajtású, egyetlen DC motor gondoskodik a robot mozgatásáról. A robot középső erőátviteli tengelyére csatlakozik egy inkrementális adó, mely biztosítja a sebességszabályozás számára a visszacsatolást. Ahogyan a feszültségvezérelt

¹Ellentétben a differenciális robottal, amely képes egy helyben való elfordulásra, így akár törtszakaszok követésére is.

egyenáramú motorok esetén gyakran lenni szokott, én is egy módosított PI szabályozót használtam sebességszabályozásra.



4.1. ábra. A pályakövetés áttekintő blokkdiagramja.

A PI szabályozók esetében gyakran előforduló probléma az elintegrálódás [14]. Az elintegrálódás a rendszerben lévő beavatkozószerv telítése miatt lép fel, kiküszöbölése történhet többféleképpen, szabályozó típusától függően. Esetünkben egy FOXBORO struktúra segítségével előzzük meg az elintegrálódást.

Az általam implementált szabályozó figyelembe veszi a motor és mechanika nemlinearitását is. Ehhez fel kellett vegyem a rendszer karakterisztikáját, tehát, hogy egy adott kimeneti feszültség mekkora végsebességet jelent. Ennek a karakterisztikának az inverzét építettük be a rendszer modelljébe, így elméletileg a nemlinearitást kiejtettük.

4.1.2. Referenciapont-választás

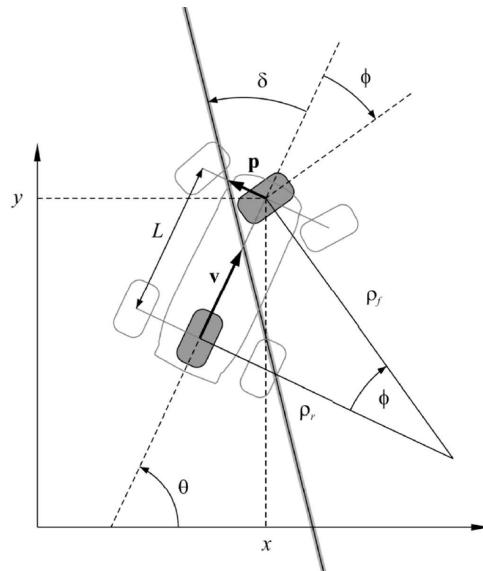
A sebességszabályozók számára a sebesség alapjelet a pálya biztosítja, hiszen az időparaméterezés során olyan pálya készült, amely időben egyenletesen mintavételezett, és így a pályapontok közötti távolságból a robot sebessége kiszámolható.

Már csak azt kell eldöntenünk, hogy a pálya melyik pontjához tartozó sebesség alapjelet alkalmazzuk az adott mintavétnél. Ezt hívjuk *referenciapont-választásnak*. Az eljárás első közelítésben igen egyszerű, a pálya pontjai közül a robot pozíciójához legközelebbi pályapontot választjuk referenciapontnak, és így már egyértelműen adódik a sebesség alapjelünk

is.

A fejlesztés egy korai stádiumában felmerült, hogy ezt a referenciapontot ne így határozam meg, hanem folyamatosan léptessem a pálya mentén. Ezzel kvázi előírtuk, hogy a robot adott időpontban a pálya mely pontjában tartózkodjon. Mivel nem biztos, hogy a robot ténylegesen a kívánt pozícióban található, egy külön szabályozó segítségével korrigáltam a pályába kódolt sebesség alapjelet, hogy a robot elérje a referenciapontot.

Amennyiben nem ideális modellt használtunk, a megoldás nem működött, a rendszer instabiltá vált. Később beláttam, hogy a megoldás problémája az volt, hogy nem csak azt írtuk elő a robot számára, hogy mekkora sebességgel haladjon a pálya mentén, hanem a referenciaponton keresztül azt is, hogy hol tartózkodjon az adott időpontban. Ez már azért sem lehetséges, mivel, ha a robot a referenciaponthoz képest lemaradásban van (általában ez történik), akkor a sebességalappel korrekció növelné a sebességet, pedig azt már alapból úgy írtuk elő, hogy a lehető leggyorsabban haladjon a robot a pálya mentén. Tehát az alapjel módosító szabályozóval arra kényszerítenénk a rendszert, hogy szegje meg a saját korlátozásait.



4.2. ábra. Ferde vonal és robot modellje [15]

A végleges megoldásnál ezzel szemben a referenciapontot alakítjuk a robothoz, nem pedig fordítva. Ez azt jelenti, hogy nem írjuk elő, hogy a robot a pályát mennyi idő alatt járja be, csak azt, hogy a pálya adott pontjában mekkora sebességgel avatkozzunk be.

A pályakövető algoritmusnál lényeges szempont a futási idő, mivel a roboton valós időben kell működnie, ezért a referenciapont meghatározásánál nem megyünk végig a pálya összes pontján. Hogy ezt megtegyük, a legközelebbi pont keresését az előző iterációban használt referenciapontnál kezdjük, és csak egy bizonyos számú pontot vizsgálunk meg. Ha a robot korlátai megfelelően lettek beállítva, akkor az egymás utáni referenciapontoknak időben sorban kell következniük. Ezért teljesen felesleges a pálya összes pontját megvizsgálnunk.

4.1.3. Túlhaladás problémája

A pályakövetés valós roboton való tesztelése során egy érdekes jelenséget tapasztaltunk, miszerint néha a pálya közepén megállt a robot és nem indult tovább. Későbbi elemzés során kiderült, hogy ez abban az esetben történik, mikor túlhalad egy szegmensen, tehát nem pontosan a szegmens utolsó pontjában áll meg a robot.

A probléma ebben az esetben az, hogy a referenciaPont-választás alapján a következő szegmens legelső pontját találja meg, és az ehhez tartozó sebességérték viszont igen alacsony. Míg el nem éri a pálya kezdetét, addig ezt a sebességet tartja a robot.

A probléma elkerülését úgy oldottuk meg, hogy amíg a pályaszakasz első pontja van legközelebb a robot pozíójához, addig nem a pályába kódolt sebességet továbbítjuk az alsóbb szintű szabályozónak, hanem a gyorsulás és sebességkorlát alapján növeljük a robot sebességét. A valóságban ez csak köztes szakaszok váltásánál jelentkezik, mivel a pályát eleve a robot kezdőpozíciójából tervezzük. Ezzel szemben két szegmens közt nem garantált a pálya pontos követése, mivel nem pozíciószabályozást használunk.

4.2. Virtuális vonalkövető szabályozás

Az orientációszabályozás feladata a kormányszög alapjel biztosítása a pályakövetés során. Erre a célra egy virtuális vonalkövetést valósítottam meg. A vonalkövető autók rendszermodelljénél és szabályozásánál azzal a feltételezéssel élünk, hogy az autó első keréktengelye alatt egy kereszttirányú, egydimenziós vonalszenzor helyezkedik el. A jelenlegi esetben a pályakövető szabályozás elvét egy ehhez hasonló „virtuális szenzor” segítségével fogalmaztam meg. Ez a módszer nagyon hasonló a RoboNAUT [15] versenyen is látott vonalkövető autók szabályozására, azzal az előnnyel, hogy itt sokkal pontosabban ismert a „vonal” helye és orientációja.

Mivel egy ilyen vonalkövető autó esetén a vonalszenzor mozgására van szükségünk, így módosítani kell a robotunk modelljét ((1.2)), amit a következőképpen tehetünk meg:

$$\begin{aligned}\dot{x} &= v_r \frac{\cos(\theta + \phi)}{\cos \phi} \\ \dot{y} &= v_r \frac{\sin(\theta + \phi)}{\cos \phi} \\ \dot{\theta} &= v_r \frac{\tan \phi}{L},\end{aligned}\tag{4.1}$$

ahol a jelölések megegyeznek az 1.2 esetén használtakkal, de a továbbiakban a sebességet egyszerűen csak v -vel jelöljük. Ezen egyenletek és a 4.2 ábra modellezésével a következő egyenleteket kapjuk: [15]

$$\begin{aligned}\dot{\delta} &= -v \frac{\tan \phi}{L} \\ \dot{p} &= v \cdot \tan \delta - v \cdot \tan \phi - v \cdot \frac{p}{L} \tan \delta \tan \phi\end{aligned}\tag{4.2}$$

Látható, hogy ez egy nem lineáris rendszer, de a szabályzótervezéshez ezt linearizálnunk kell. Mivel az a célunk, hogy a robot a vonalon, és azzal párhuzamosan helyezkedjen el, így a munkapont, amely körül a linearizálást elvégezzük a $p = 0$, $\phi = 0$ és a $\delta = 0$. Így a következő egyenletekkel számolhatunk:

$$\begin{aligned}\dot{\delta} &= -\frac{v}{L}\phi \\ \dot{p} &= v(\delta - \phi - 0)\end{aligned}\tag{4.3}$$

A linearizálás egyszerű, mert a tangens 0 környezetében jól közelíthető az argumentumával. Látható, hogy a \dot{p} esetén az utolsó tagot elhanyagoljuk, mivel a két kis szög szorzata annyira kis számot eredményez, hogy ez gond nélkül megtehető. Ha ezt kicsit más formában írjuk fel, rögtön megkapjuk a linearizált rendszer állapotteres leírását:

$$\begin{aligned}x &= [\delta \quad p]^T \\ \dot{x} &= \begin{bmatrix} 0 & 0 \\ v & 0 \end{bmatrix} x + \begin{bmatrix} -v/L \\ -v \end{bmatrix} \phi \\ p &= [0 \quad 1]x + 0 \cdot \phi\end{aligned}\tag{4.4}$$

Ellentétben egy valós vonalkövető autóval, viszonylag pontosan meg tudjuk határozni a rendszer állapotváltozóit. Így célszerű közvetlenül ezek visszacsatolása, mivel így szabadon megválaszthatóak a visszacsatolt rendszer pólusai. Ezt érdemes úgy megtenni, hogy minimálisra csökkentsük a túllendülést. Ha a rendszer válaszát kéttárolós lengőtaggal közelítjük, akkor annak átviteli függvénye a következő:

$$W(s) = \frac{\omega_0^2}{\omega_0^2 + 2\xi\omega_0 s + s^2},\tag{4.5}$$

ahonnan a pólusok:

$$s_{1,2} = -\omega_0\xi \pm j\omega_0\sqrt{1 - \xi^2},\tag{4.6}$$

ahol ω_0 a rendszer csillapítatlan sajátfrekvenciája és ξ a csillapítási tényező. Ha túllendülés mentes rendszert szeretnénk, de a lehető leggyorsabb beállási idővel, akkor $\xi = 1$ -et kell választanunk. Az ω_0 megválasztására nincsen hasonló korlátozásunk, ezt az aktuális pályához tudjuk igazítani.

Az algoritmust úgy készítettem el, hogy az inicializálási fázisban a kívánt pólusoknak megfelelően, az Ackermann-képlet [16] segítségével kiszámítja az erősítési tényezőket, és később ezt használja fel a szabályozási fázisban. Az eredmények azt mutatták, hogy az így készült szabályozóval a szimulációban a robot trajektóriája a kanyarokat levágta. Ez az

eredmény egyáltalán nem meglepő, mivel a szabályozást úgy írtuk fel, hogy az autó eleje kövesse a pályát, de a pályatervezés során a robot referencia pontjának pályáját terveztük meg. Szerencsére ezt egyszerűen orvosolhatjuk, ha a mintavételezett pálya minden pontját eltoljuk az autó hosszával.

4.3. Eredmények

Most pedig tekintsük át a pályakövetés alakulását különböző paraméterek esetén. A sebességszabályozásnál láttuk, hogy egy újabb szabályozás bevezetése csak hibát okoz a rendszerünkben, így csak az alacsony szintű PI szabályozó paramétereivel tudjuk módosítani a sebességprofil követésének minőségét.

Az orientációs szabályozás esetén az előírt pólusok értékein kívül a szenzorsor pozíójával is lehetőségünk van módosítani a szabályozás minőségét. Intuitívan beláthatjuk, hogy minél előrébb tekintünk, annál gyorsabban tudunk reagálni, így alacsonyabb beavatkozójel szükséges a szabályozás során. Ezt a következőképpen írhatjuk le [15]:

$$\tan \phi = \tan \phi \cdot \frac{L}{L + d}, \quad (4.7)$$

ahol d az előretekintés távolsága az eredeti szenzor pozíciótól, γ a számított beavatkozójel, ϕ a valós kormányszög. Természetesen az előretekintés mértékének túl nagyra választása azt eredményezi, hogy a kanyarokat hajlamos a robot levágni. Mégis jelentős előnyvel rendelkezik, mivel így részben kiküszöbölnihető a kormányszervó nem nulla beállási ideje.

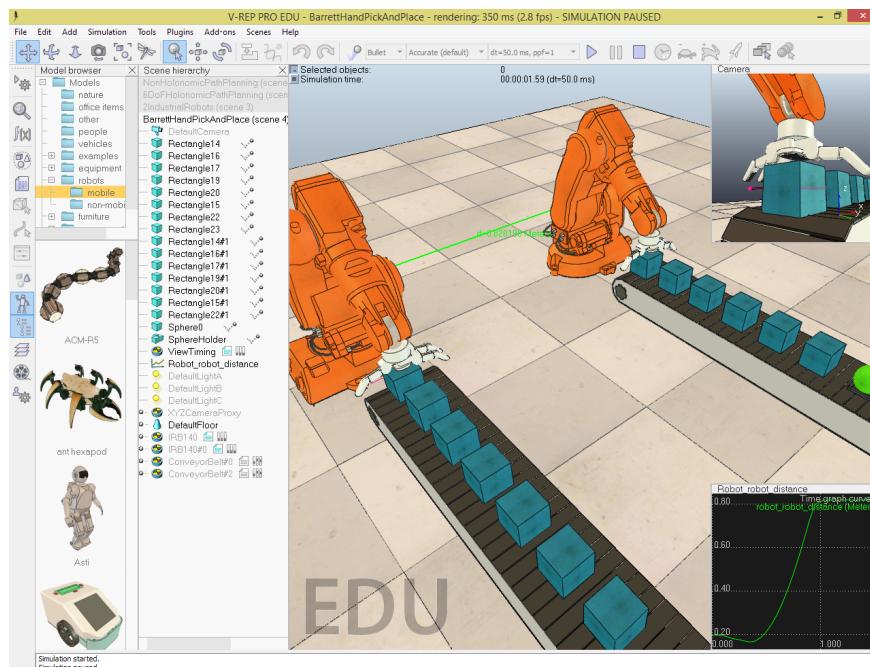
5. fejezet

Algoritmusok megvalósítása

Ebben a fejezetben az algoritmusok megvalósításáról, és az azokhoz használt eszközök-ről beszélek. Bemutatom a használt szimulációs környezetet, és a köré készült programok működését, majd leírom a szimulátorban és a valós robotokon elért eredményeimet.

5.1. Szimuláció – V-REP

A robot mozgásának szimulálására a V-REP robotszimulátort használtam. A program a Coppelia Robotics terméke [17], amely oktatási célból ingyenesen letölthető és használható. Nagyon széleskörűen használható program a robotika minden ágában. Tesztelhető benne ipari szerelőrobotok működése, ahogyan az az 5.1 ábrán is látható, felhasználható ilyen robotok programozásának oktatására is, de a mobil robotok területén is kifejezetten praktikus eszköz. Jól dokumentált, sok oktató anyaggal, példaprogramokkal együtt. Folyamatosan frissítik és új funkciókkal bővítik a programot.

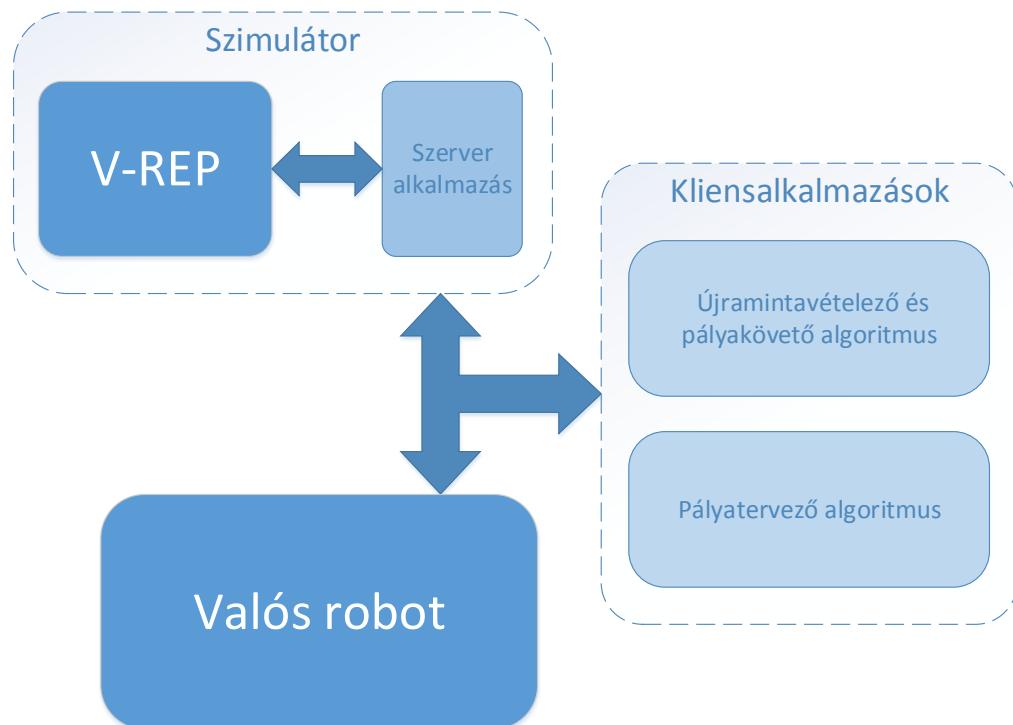


5.1. ábra. A V-REP szimulációs program

5.1.1. Szerver program

Több módon is kiegészíthető a program működése. A megoldásunkban egy szerver programot hoztunk létre, mely kommunikál a V-REP egy lua szkriptjével, majd a kapott üzenet alapján mozgatja a szimulált robotot. A szerver nem csak a szimulátorral áll kapcsolatban, hanem hozzá csatlakozhatnak a különböző egyéb algoritmusok, ahogy az az 5.2 ábrán is látható. A fejlesztés során próbáltuk a szimulációt a robot típusától függetlenné tenni, ezt a következőképpen valósítottuk meg:

A szimuláció indulásakor a lua szkript elküldi a szimuláció módját, és a hozzá tartozó paramétereket. Innen a szerver alkalmazás eldönti milyen paraméterek és egyéb adatok érkezhetnek, illetve, hogy melyik kliensre kell várjon. Ha a kapcsolat létrejött a kliens alkalmazással, akkor az a paramétereknek megfelelően végzi a feladatát, majd az eredményt a szerver alkalmazáson keresztül elküldi a szimulátornak. Ez a struktúra első ránézésre igen bonyolultnak tűnik, de ez a módszer biztosítja, hogy a szimulátor egyszerűen lecserélhessük egy valós robotra, vagy akár párhuzamosan is működhessen vele. A szerver működése teljes mértékben transzparens, a későbbiekben ennek működését a szimulátor részének tekinthetem.



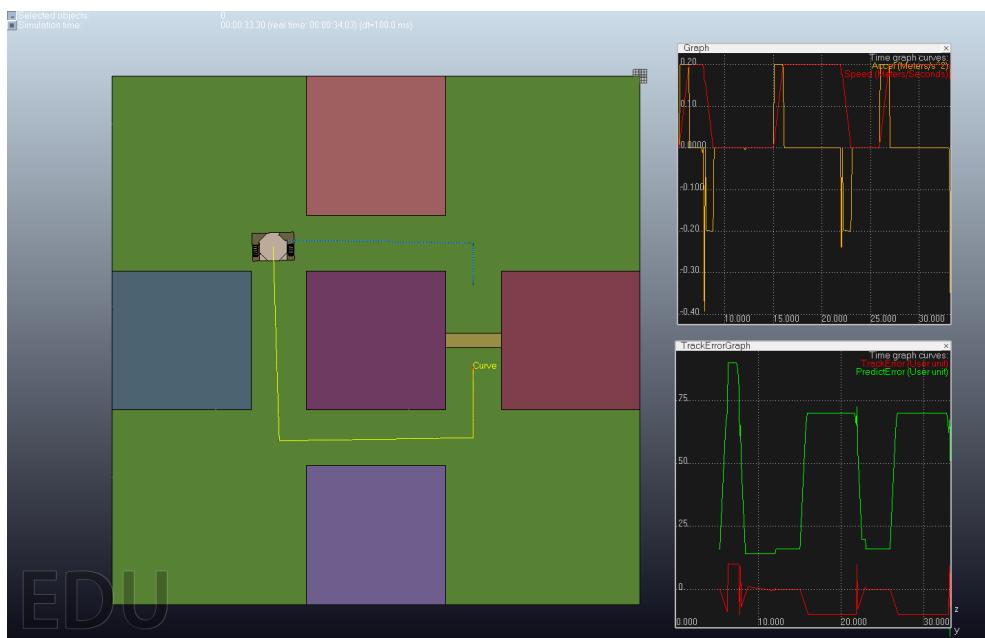
5.2. ábra. Az elkészült keretrendszer blokkvázlata

5.1.2. Kliens program

A különböző kliensprogramok más-más paramétereket várnak, ezt biztosítja a szerver alkalmazás. Az időparaméterező és a pályakövető szabályozás teszteléséhez készült egy kliens, mely a szimulártól fogad egy előre elkészített pályát, majd ezt újramintavételezi. Az így

készült pályát visszaküldi a szimulátornak, ami kirajzolja az új pályát, majd elküldi a robot aktuális pozíóját. Innen átveszi a működést a pályakövető algoritmus, a kapott pozíciót feldolgozza, és ez alapján az előző fejezetben részletezett módon kiszámítja a beavatkozó jeleket, amit visszaküld a szimulátornak. A szimulátor és a pályakövető alkalmazás működése szinkronizálva van, azaz megvárják egymást a következő lépéssel.

A másik elkészült kliens alkalmazás a pályatervező program. Ez nem vár pályára a szimulátortól, csak az előre meghatározott környezet nevére. Itt kompromisszumot kellett kötnünk, mivel a szimulátor speciális fájlformátumot tud csak kezelni, ezért közös fájlokkal dolgozik a két program, de a pályatervező más forrásból is elfogad pályát, így továbbra is lecserélhető marad a szimulátor. A pályatervezés után a működése teljesen megegyezik a pályakövető kliensprogramnál leírtakkal, azzal a különbséggel, hogy a pályát itt a tervező szolgáltatja.

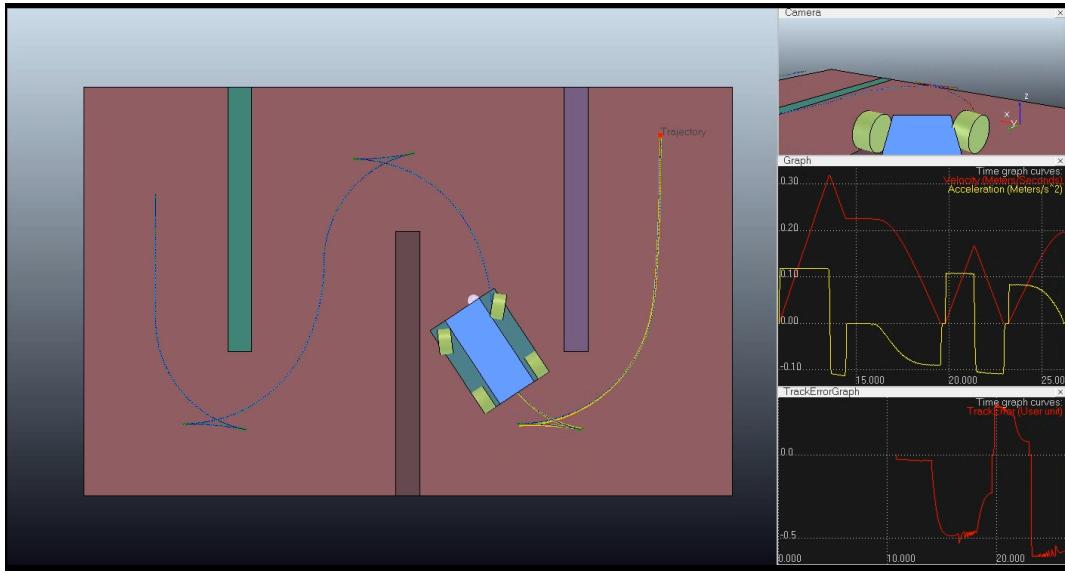


5.3. ábra. Az elkészült keretrendszer működés közben. Az ábrán az RTR algoritmus által tervezett pályán halad a differenciális robot.

5.1.3. Implementáció

A programokat a C++ nyelvet készítettem el. Azért esett a választásom erre a programozási nyelvre, mert a valós roboton, beágyazott rendszerben is könnyedén használható. A beágyazott környezet miatt igyekeztem kerülni bármiféle olyan külső szoftvercsomag használatát, aminek a használata problémás lehet a valós roboton.

Ezek mellett a fejlesztés során igen fontos volt az objektum-orientált szemléletmód, mivel így biztosítható a legjobban a modularitás, és a későbbi egyszerű fejlesztés, módosítás. Az implementálás során egyéb előnyös tulajdonságát is ki tudtuk használni, ezek közül a legjelentősebb az újrafelhasználhatóság volt. A programozás előrehaladtával a fejlesztés sebessége is nőtt, mivel az előzőleg elkészített kódrészleteket egyszerűen újra tudtuk használni.

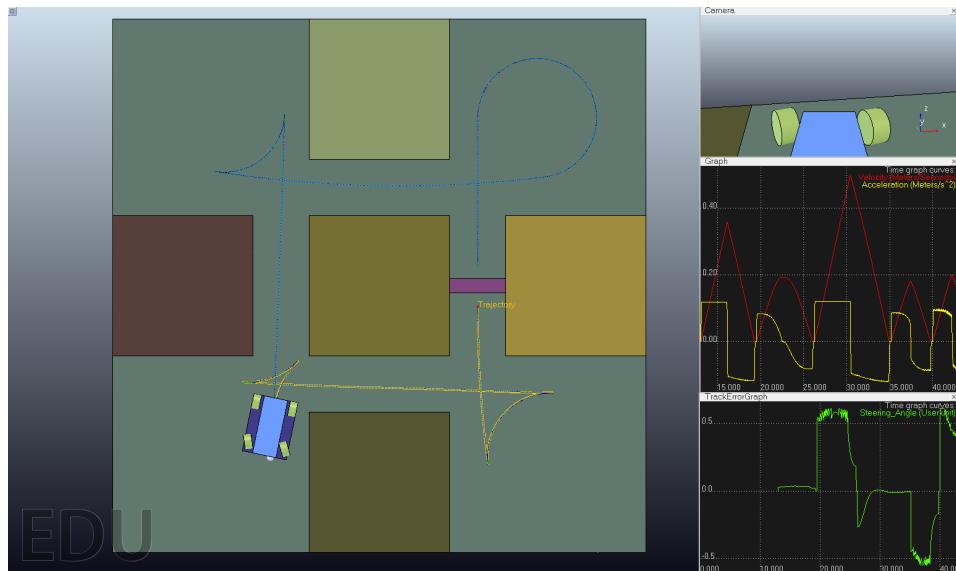


5.4. ábra. A C*CS alkalmazása autószerű robotok esetén, szimulációs környezetben

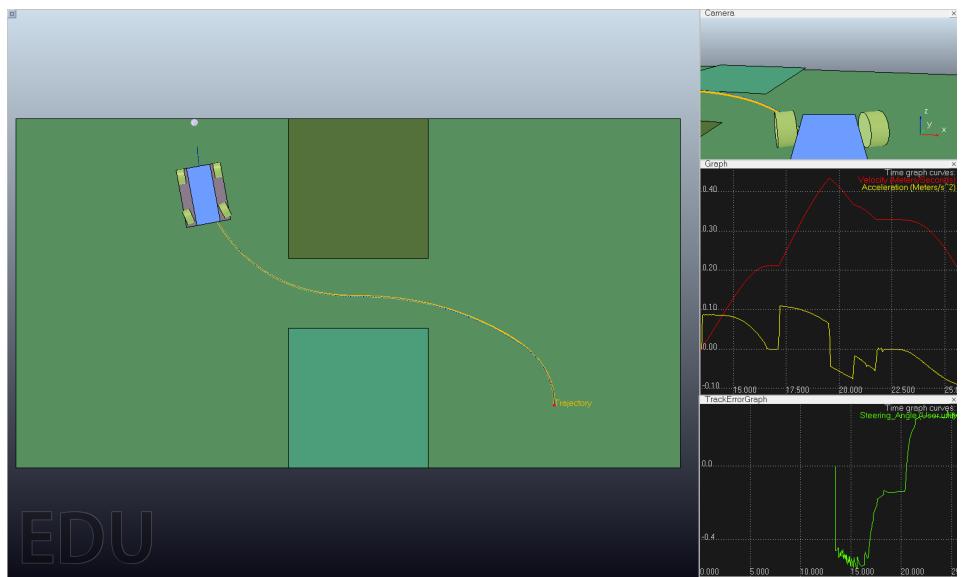
5.2. Szimulációs eredmények

Az általam ismertetett algoritmusok szimulációs környezetben a várakozásomnak megfelelően működtek. Az eredmények a következő ábrákon láthatóak. A képek a V-REP szimulátorról készültek, az akadályokat és a pálya határait színes polygonok jelzik, a pályatervező által generált pályát kék színnel jelölték, míg a robot által bezárt pályát sárgával. A robot alatt található polygon jelzi a tervezés során ténylegesen figyelembe vett robot méretét.

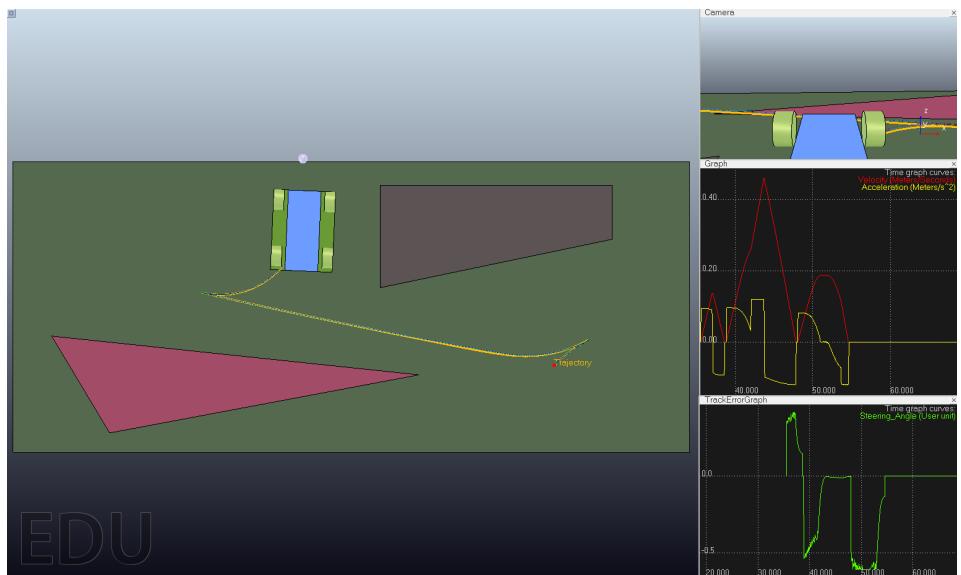
A robotautó szimulálásakor jobb oldalt középen a sebesség és a gyorsulás profil látható, alatta pedig a kormányszög értéke. A felső grafikon itt is m/s és m^2/s mértékegységben szerepelnek a mennyiségek, míg a kormányszög radiánban. A grafikonok felett az autó elejére szerelt kamera képét látjuk.



5.6. ábra. Szűk folyosók. Autószerű robot esetén igen bonyolult pálya adódik.



5.5. ábra. A 2.5. ábrán látható pálya RTR globális tervező esetén.



5.7. ábra. A tervezett pálya igen hasonló egy valós esetben végrehajtott parkoláshoz

6. fejezet

Megvalósítás valós roboton

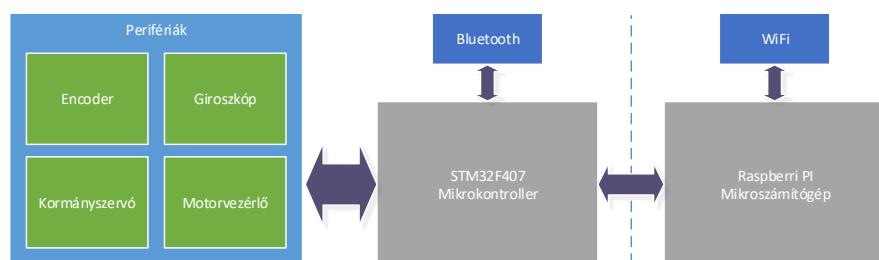
Az elkészült algoritmusok elméleti szinten és a szimulátorban is a vártnak megfelelően működtek. A szimuláció során használt modelleket próbáltam minél inkább a valósághoz igazítani, de a valósággal teljes mértékben megegyező modell megalkotása természetesen nem lehetséges, így az elkészült algoritmusokat valós roboton is kipróbáltam. Ebben a fejezetben az elkészült robotról, a fejlesztés és az implementálás közben felmerült nehézségekről és a jövőbeli tervekről fogok beszélni.

6.1. Felépítés

A robot elkészítése a szakmai gyakorlatom keretében valósult meg. A fejlesztés során a célom egy olyan autószerű mobilrobot készítése volt, amely a környezet ismeretében képes a pályatervező algoritmusok futtatására és egy elkészült pálya követésére.

Az elkészült rendszer két részre bontható a feladat jellegéből fakadóan. Az alacsonyabb szintű feladatok, mint a motor vezérlése, szabályozása, kormányzás és szenzorfúzió egy beágyazott mikrokontroller feladatai közé tartoznak. Emellett beépítésre került egy bluetooth modul, amely segíti a hibakeresést. Ezen felül ez a modul lehetővé teszi, hogy ezt az alacsonyabb szintű alrendszeret leválasszuk, az erőforrásigényesebb algoritmusok fejlesztése során.

A magasabb szintű feladatok, mint például a pályatervezés és követés, már egy Raspberry PI mikroszámítógép feladata. Ezen a számítógépen működhet a robot felügyeleti szerve



6.1. ábra. A robot rendszermodellje

is, illetve ez teszi lehetővé vezeték nélküli hálózathoz való csatlakozást is.

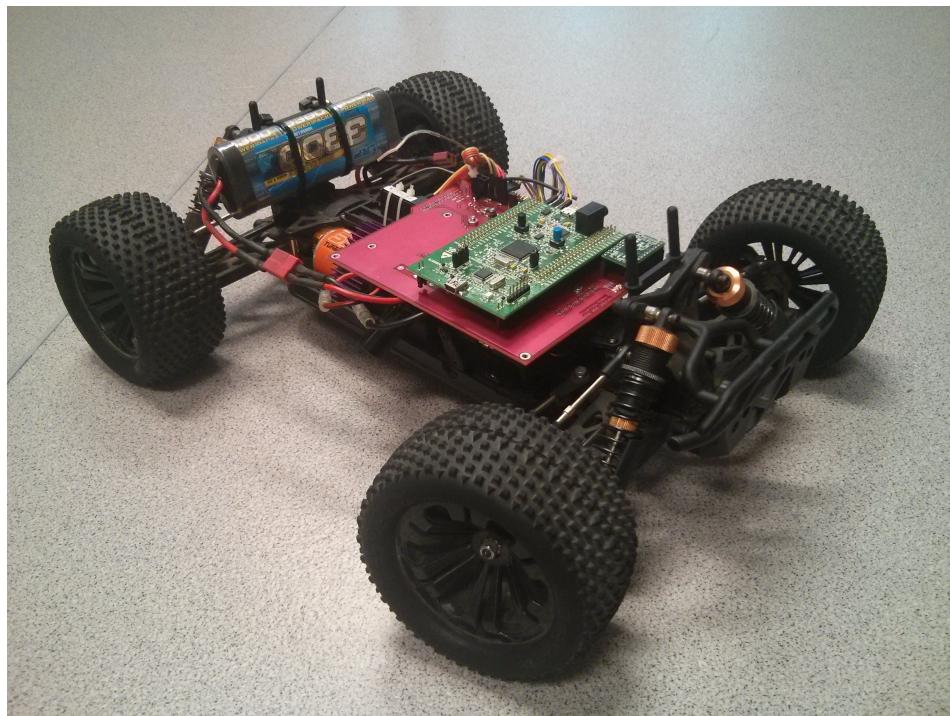
6.2. Nehézségek

A robot fejlesztése és az algoritmusok valós roboton való végrehajtása közben számos problémával kellett megküzdenem, az alábbiakban bemutatom a legjelentősebbeket.

6.2.1. Helymeghatározás

Az eredeti tervekben a robot helymeghatározása, így a pályakövetés visszacsatolása egy kamerás látórendszer feladata lett volna. A kamera képfeldolgozásáért egy a RoboCup versenyre készült látórendszert használtam volna. Ennek a módszernek a pozíció meghatározási pontossági igen jó, de az orientáció meghatározása igen nagy hibával történik. Ezen felül, mivel a program kifejezetten a robotfoci pályához készült, kalibrálása igen nehézkes, és a mért adatok kinyerése is bonyolult lett volna.

Egy későbbi tervben a tanszéken elkészült helymeghatározó rendszer használata merült fel. Ez a rendszer ultrahangos adó és vevőegységek segítségével végzi a pozíció meghatározását. Az orientáció mérése pedig optikai úton történik, az ultrahangos háromszögelő eljáráshoz nagyon hasonló módon. A robot hardver tervébe bekerült egy illesztő áramkör, és egy csatlakozó is, hogy ezzel a rendszerrel együtt lehessen használni. Ez a módszer igen jó pontossággal működik, és abszolút helymeghatározásra képes. Hátránya, hogy csak egy korlátozott területen belül képes az objektum érzékelésére. Végül mielőtt a robot működésbe lépett volna a rendszer meghibásodott, így ezt a módszert is el kellett vetni.



6.2. ábra. Az elkészült robotautó

6.2.2. Odometria

Végső megoldásként a rendelkezésre álló inkrementális adó és a giroszkóp adatait feldolgozva dead-reckoning segítségével határoztam meg a robot pozícióját. Ehhez az (1.2) egyenletet kell felhasználnunk. A pozíció meghatározásához szükségünk van az autó orientációjára θ , és a sebességére v_r . A sebességmérést az inkrementális adó segítségével végeztem el. Az adó felbontása megfelelő, az ebből fakadó hiba $1\frac{m}{s}$ sebességnél 1%-os.

Mivel a kormányszög mérésére nincs lehetőségünk az orientációt egy giroszkóppal mérjük, A giroszkóp a szögsebesség mérésére képes. A szögsebességből egy egyszerű integrálás segítségével számítható az orientáció. Sajnos a giroszkóp IC paraméterei nem megfelelők, így az általa szolgáltatott információt szúrnunk kell. Ehhez bekapcsoláskor egy egyszerű átlagolással kiszámíthatjuk a driftet, azaz a mért értékek konstans eltolódását. Majd az algoritmus futása közben egy szűrő algoritmus segítségével közvetlenül az orientációt kapjuk [18].

Mivel a dead-reckoning során integrálás segítségével számítunk pozíciót, így a maradó hiba időben növekvő mértékű lesz. Az esetünkben a giroszkóp hibája a jelentős, de ez az általam használt pályaméretek és követési idők esetén még elhanyagolható.

6.2.3. Kotyogás

A sebességszabályozás alacsony sebességek esetén nem megfelelően működik. A hibás működés oka ez erőátviteli rendszer kotyogása. Ez abból fakad, hogy a sebesség visszacsatolás a motorral szorosabb összeköttetésben lévő kardántengely segítségével történik, amely a motor elfordulásának hatására könnyen elfordul, de a kerekek még állva maradnak.

Ebben az esetben az inkrementális adó viszonylag nagy sebességet jelez, míg valós elmozdulás nem történik. A hirtelen indítás a PI szabályozó arányos tagjának következménye, mivel ilyenkor az ugrásszerű beavatkozójel nagy hibát okoz. A szabályozás a következő szabályozási ciklusban egy hasonlóan nagy, de ellenkező irányú beavatkozójellel próbálja a hibásan mért nagy sebességet ellensúlyozni. Ez a hatás oszcillációt okoz a szabályozásban.

A kotyogás a rendszer egy nemlineáris hatása, amit nehéz kiküszöbölni. Ipari eszközökben a hajtás lecserélése egyszerű fogaskerekről hullámhajtóműre megoldást jelent, de erre jelen esetben nincs lehetőség. A szabályozás szempontjából ezt több módon is lehet kezelní, de tökéletes megoldást nehéz találni.

Az egyik lehetséges módszer a PI szabályozó hangolása oly módon, hogy az integráló hatást megnöveljük, míg az arányos tagot lecsökkentjük. A módszer behangolása igen időigényes, és nem is minden esetben érhető el vele a várt eredmény. Ezen felül rendelkezik egy egyértelmű hibával, a rendszer válaszának jellege erősen megváltozik, sok esetben lelassul, ami más problémákat hozhat elő.

Egy másik módszer az inkrementális adó sebességjelének szűrése. Ezt megtehetjük egy egyszerű alul-áteresztő szűrő segítségével, de ez jelentős holtidőt vihet a rendszerbe, vagy bonyolultabb szűrőalgoritmusokat is használhatunk, például egy Kálmán-szűrőt, de ennek implementálása, és hangolása idő, és erőforrásigényes feladat.

Lehetséges a program módosítása, kiegészítése, például csak nemnegatív beavatkozójel

kiadása, vagy alacsony sebességeken más szabályozó paraméterek használata, vagy a nem linearitás figyelembe vétele, mint ahogy azt tettük a holtsáv kiküszöbölésekor, de a kotyogás egy ezzel ellentétes folyamat, ami megbonyolítja az implementálást.

6.3. Jövőbeli tervezek

A jövőben szeretném egy egységes programkönyvtárat kialakítani, ami megkönnyíteni a különböző pályatervező algoritmusok implementálását, azok eredményeinek összevetését. Ezen felül a valós roboton egy általános interfész kialakítása, mely bármely általános tervezőalgoritmus számára elérhető tenné a használatát, is célom.

A dolgozatom elkészítése során több helyen is modelleket használtam. Ezeket a modelleket szeretném kibővíteni, finomítani, ahol ezzel jelentős eredményeket lehet elérni, például az időparaméterezés során újabb korlátozások figyelembe vétele, vagy dinamikai modell alkalmazása, vagy a tervezés és követés során a kormányszög beállási idejének figyelembe vételével.

Továbbá célom újabb robottípusok és algoritmusok megismerés, implementálása. A robotautó átalakítása, hogy a RobonAUT 2015 versenyen mint safety car működjön, és hosszabb távon szeretném, ha a tervező algoritmusokat a versenyen is használhatnák.

7. fejezet

Összegzés

Összegzésképpen elmondható, hogy a feladatban kitűzött céljaimat sikerült elérnem. Megismertem a robotikában leggyakrabban alkalmazott pályatervezési módszereket, majd implementáltam egy pályatervező algoritmust, mely autószerű robotok számára készít útvonalat, de differenciális robotok esetén is kiválóan alkalmazható. Az elkészült pálya nem csak távolságban, de időben is egyenletesen mintavételezhető az elkészült algoritmus segítségével. Implementáltam egy pályakövető algoritmus, ami lehetővé teszi a pálya stabil lekövetését, nem csak szimulátoron, hanem valós robottal is. Megismerkedtem a V-REP robotszimulátor programmal, amelyben elkészítettem egy autószerű robot modelljét. Az algoritmusok működését nem csak szimulátor, hanem valós robot segítségével is sikerült igazolnom.

Köszönetnyilvánítás

Szeretnék köszönetet mondani **Kiss Domokosnak** a folyamatos konzultációkért, tanácsokért és iránymutatásokért. Ezenkívül **Nagy Ákosnak** javaslataiért, tanácsaiért és a segítségért, valamint **Kolumbán Sándornak** a szabályozástechnikában nyújtott segítségeért.

Irodalomjegyzék

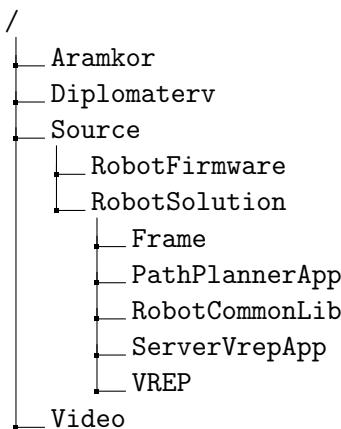
- [1] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006. Available online at <http://planning.cs.uiuc.edu/>.
- [2] B. Siciliano and O. Khatib, *Handbook of Robotics*. Springer, 2008.
- [3] D. Kiss and G. Tevesz, „A model predictive navigation approach considering mobile robot shape and dynamics,” *Periodica Polytechnica - Electrical Engineering*, vol. 56, pp. 43–50, 2012.
- [4] D. Kiss and G. Tevesz, „A steering method for the kinematic car using C*CS paths,” in *Proceedings of the 2014 15th International Carpathian Control Conference (ICCC)*, (Velké Karlovice, Czech Republic), pp. 227–232, May 2013.
- [5] J. A. Reeds and L. A. Shepp, *Optimal paths for a car that goes both forward and backwards*. Pacific Journal of Mathematics, 1990.
- [6] B. Kornberger, „Fade2d - an easy to use delaunay triangulation library for c++.” <http://www.geom.at/fade2d/html/>, 2014.
- [7] G. Katona, A. Recski, and C. Szabó, *A számítástudomány alapjai*. Typotex, 2. javított kiadás ed., 2001.
- [8] D. Kiss and G. Tevesz, „The RTR path planner for differential drive robots,” in *Proceedings of the 16th International Workshop on Computer Science and Information Technologies CSIT’2014*, (Sheffield, England), September 2014.
- [9] Ákos Nagy, *Pályatervezési és pályakövető szabályozási algoritmusok fejlesztése differenciális robothoz*. BME Automatizálási és Alkalmazott Informatikai Tanszék, 2014.
- [10] L. E. Kavraki, P. Svetska, J.-C. Latombe, and M. H. Overmars, „Probabilistic roadmaps for path planning in high-dimensional configuration spaces,” *IEEE Trans. Robot. Autom.*, vol. 12, pp. 566–580, 1996.
- [11] S. M. LaValle, „Rapidly-exploring random trees: A new tool for path planning,” tech. rep., Computer Science Dept., Iowa State University, 1998.
- [12] C. Sprunk, *Planning Motion Trajectories for Mobile Robots Using Splines*. Albert-Ludwigs-Universitat Freiburg, 2008.

- [13] D.-J. Kroon, „2d line curvature and normals.”
<http://www.mathworks.com/matlabcentral/fileexchange/32696-2d-line-curvature-and-normals/content/LineCurvature2D.m>, 2014.
- [14] I. Bézi, *Robotirányítás rendszertechnikája 3. fejezet*. BME Automatizálási és Alkalmazott Informatikai Tanszék, 2013.
- [15] K. Domokos and K. Sándor, *Segédlet a RobonAUT verseny szabályozástechnikai szemináriumához*. BME Automatizálási és Alkalmazott Informatikai Tanszék, 2014.
- [16] B. Lantos, *Egy változós szabályozások*. Akadémiai Kiadó, 2009.
- [17] C. Robotics, „Vrep.” <http://www.coppeliarobotics.com/>.
- [18] S. Madgwick, A. Harrison, and R. Vaidyanathan, „Estimation of imu and margin orientation using a gradient descent algorithm,” in *Rehabilitation Robotics (ICORR), 2011 IEEE International Conference on*, pp. 1–7, June 2011.

Függelék

F.1. Megjegyzés a CD melléklettel kapcsolatban

A dolgozat mellékletében megtalálhatóak az általam készített forráskódok, és egy videó-felvétel is a valós robotról a következő könyvtárstruktúrában:



Aramkor:

Az autószerű robot vezérlőáramköre.

Diplomaterv:

Jelen dokumentum forrása.

Source:

Forráskódok.

RobotFirmware:

A robton található beágyazott rendszer programkódja.

RobotSolution:

A keretrendszer forráskódja, Nagy Ákossal közösen fejlesztve.

Frame:

Példa környezetek.

PathPlannerApp:

Kliens program.

RobotCommonLib:

Több program által használt közös kódok.

ServerVrepApp:

Szerver program.

VREP:

V-REP környezetek.

Video:

Egy adott környezetről készült videófelvétel.