FELADATKIÍRÁS

A feladatkiírást a tanszéki adminisztrációban lehet átvenni, és a leadott munkába eredeti, tanszéki pecséttel ellátott és a tanszékvezető által aláírt lapot kell belefűzni (ezen oldal *helyett*, ez az oldal csak útmutatás). Az elektronikusan feltöltött dolgozatban már nem kell beleszerkeszteni ezt a feladatkiírást.

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 8.

Tartalomjegyzék

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 átvettem, egyértelműen, a forrás megadásával megjelöltem.

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

Budapest, 2014. december 8.

Csorvási Gábor
hallgató

Kivonat

Jelen dokumentum egy diplomaterv sablon, amely formai keretet ad a BME Villamosmérnöki és Informatikai Karán végző hallgatók által elkészítendő szakdolgozatnak és diplomatervnek. A sablon használata opcionális. Ez a sablon LaTeX alapú, a *TeXLive* TeX-implementációval és a PDF-LaTeX fordítóval működőképes.

Abstract

This document is a LaTeX-based skeleton for BSc/MSc theses of students at the Electrical Engineering and Informatics Faculty, Budapest University of Technology and Economics. The usage of this skeleton is optional. It has been tested with the *TeXLive* TeX implementation, and it requires the PDF-LaTeX compiler.

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.

Célom azon megközelítések és módszerek áttekintése, amelyek megoldást nyújtanak az autonóm pályatervezés kérdésére. Néhány módszert részletesebben 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 használható eljárásokat kapjunk.

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.

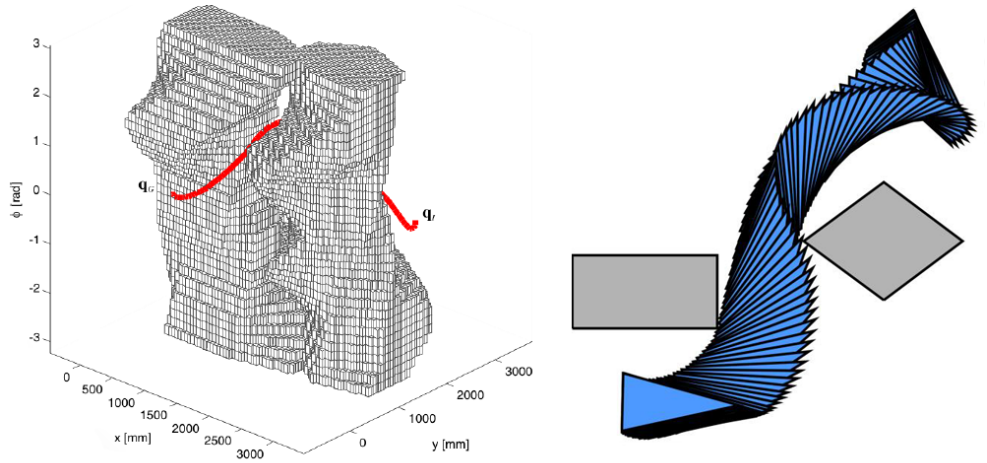
1.2.1. Alapvető fogalmak

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

$$q = (x, y, \theta), \tag{1.1}$$

ahol q a robot konfigurációja, x, y határozza meg a robot pozícióját 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 jelzi a robot útját a célpontja felé [3].*

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

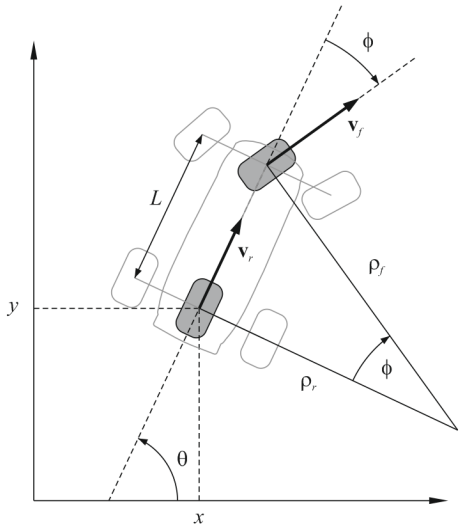
A korlátozások ismerete alapvető fontosságú a pályatervezés és mozgásirányítás során. A környezetben elhelyezkedő akadályokat *globális korlátozásoknak* tekintjük, a robothoz kapcsolódó korlátozásokat pedig *lokális korlátozásoknak* [2]. A lokális korlátozásokat a robot konfigurációs változóinak differenciál-egyenletével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

1.2.2. Pályatervezők osztályozása

Mielőtt belekezdenék az általam megvizsgált pályatervező algoritmusok részletesebb ismertetésébe, 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 veszik, 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 eldönteni, 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.

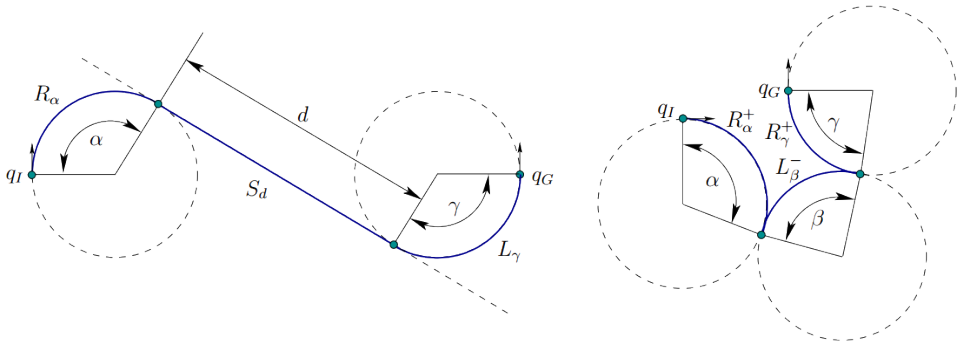
2. fejezet

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 akadályoktól 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 akadályoktól mentes környezetben találhatunk optimális útvonalat, de ennek hátránya, hogy mindig minimális sugarú pályákat feltételez, mely egy valós esetben nem életszerű, illetve a pályák lehetnek igen bonyolultak is. De 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 nyújtsanak, 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 feltesszük, 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:

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

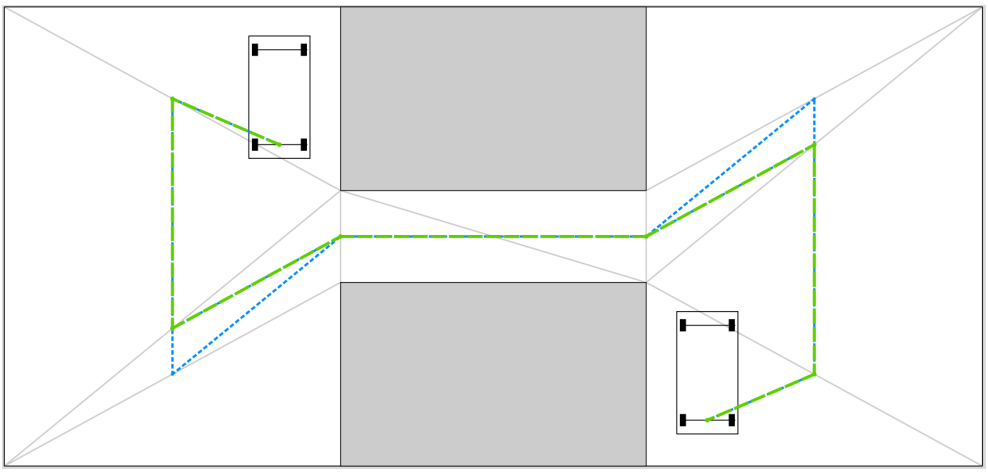
Ha a kiadódó sugár kisebb mint a minimálisan megengedett ($|\rho_{I,\tilde{G}}| < \rho_{min}$), esetleg $\theta_I = 0$, akkor egy egyenes vagy egy kör segítségével egy köztes kezdőkonfigurációba ($\tilde{q_I} = (\tilde{x_I}, \tilde{y_I}, \tilde{\theta_I})$) kell eljutnunk, ahol biztosított, hogy $\tilde{\theta_I} \neq 0$ 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 sugara 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 robotok számára 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.

[1] Bár 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 célra egy celladekompozíción alapuló algoritmust használtunk. Ez az eljárás a környezetet háromszögekre bontja, majd ezeknek a háromszögeknek az oldalfelező pontjait összekötve gráfot alkot. Ebbe beszúrja 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 [6] segítségével megkeressük a legrövidebb utat. Erre egy példát a 2.2 ábrán láthatunk.

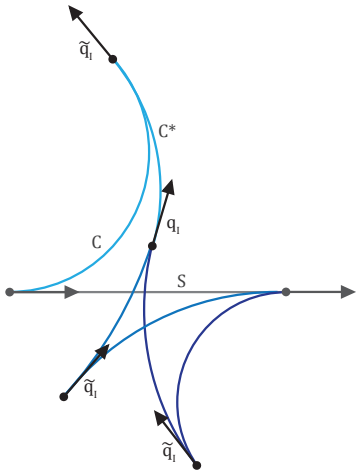
Ennek a megoldásnak az előnye, hogy a szabad terület közepén alkot pályát, így ha az autó ezt követi, akkor bármilyen irányú manőverezésre lesz lehetősége, ha a pálya ezt megengedi. További előnye, hogy ez egy kombinatorikus eljárás, így véges időn belül képes megmondani, hogy létezik-e megoldás. Az 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 akadályoktól, 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 közt keres útvonalat, ami egyszerű esetekben akár rögtön megoldásra is vezethet, felgyorsítva az algoritmus működését. Ha ez a keresés nem járt sikerrel, akkor az előzetes pályát megfelezi az algoritmus, és az első konfiguráció valamint az új célkonfiguráció közt keres megoldást. Ezt

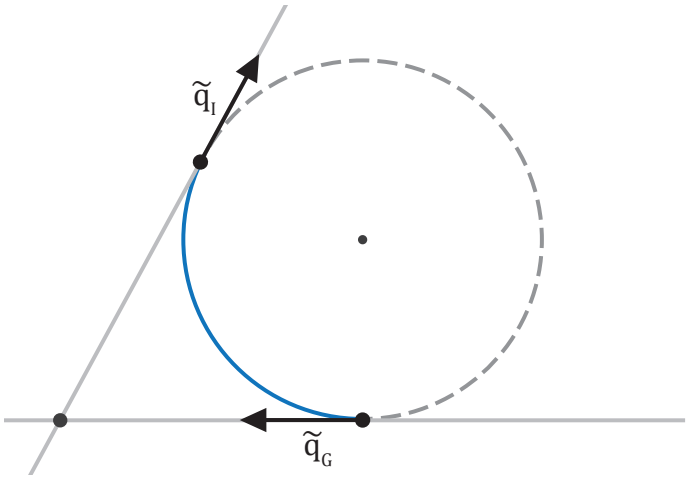
egészen addig ismétli, míg van köztes konfigurációs pont, ha elfogyott, további pontokat illeszt a pályába.

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. *$q_{\tilde{I}}$ megválasztása*

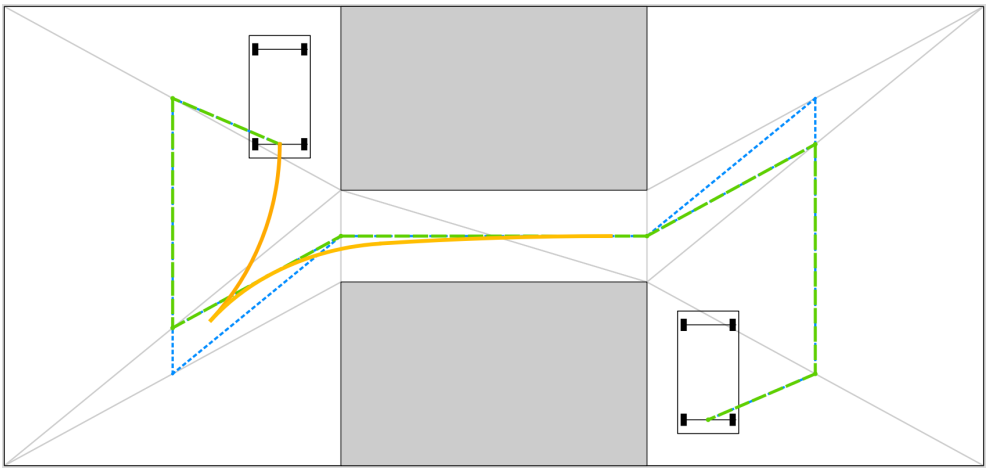
A végtelen sok megoldást a $\tilde{q_I}$ kiválasztásának szabadsága okozza. Erre egy példa látható a 2.3 ábrán. Ezért az algoritmus összegyűjti azokat a konfigurációkat, melyeket a q_I konfigurációból ütközés nélkül elérhetünk. Ehhez a környezetet fel kell osszuk egységnyi távolságokra, mivel így véges 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 csúcsára ki kell számoljuk, hogy ütközést tudjunk 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 lépésben 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.

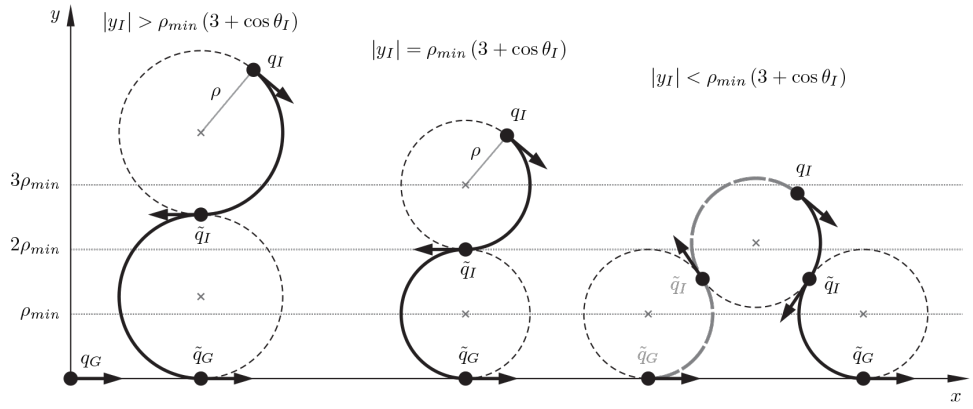
Végül az így keletkező végrehajtható pályák sokasága közül ki kell választanunk egyet. Ezt többféleképpen megtehetjük. Talán a legkézenfekvőbb a legrövidebb megoldás kikeresése és beillesztése az előzetes pályába. Itt érdemes megemlíteni, hogy a végeredmény akkor fog igazán hasonlítani a valósághoz, 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 végtelenségig, 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 olyan esetben megoldással tér vissza, mikor a globális tervező érvényes pályát ad vissza. Azaz a körívekkel való közelítés nem csökkenti a megoldás létezésének esélyét.



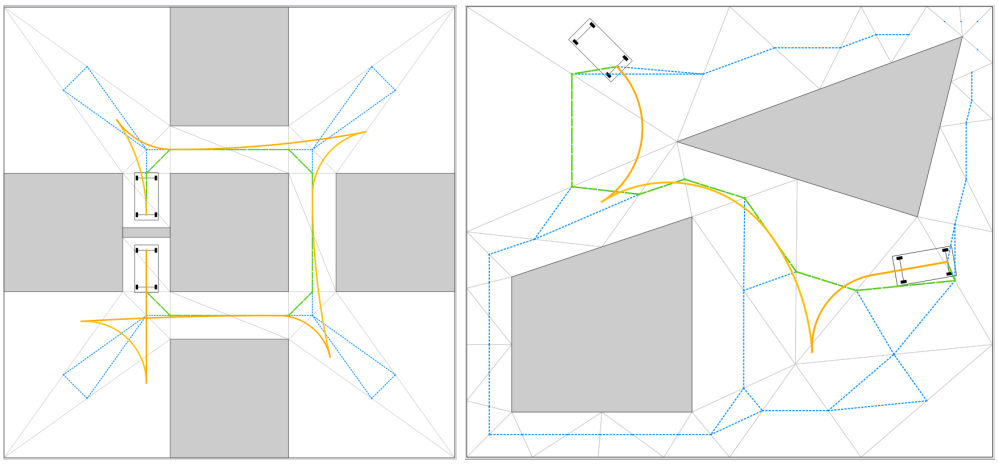
2.6. ábra. *A $c\bar{c}S$ algoritmus megoldásai különböző y_I esetén[4]*

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 $c\bar{c}S$ mindig használható, 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 script-ben 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 algoritmusnak[2]. 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.

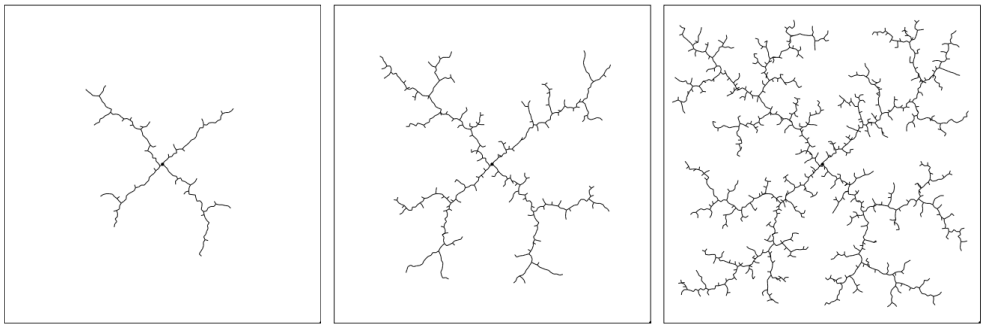
2.6. Új globális tervező

A fejlesztés során a fentebb felsorolt problémák miatt a globális tervezőt lecseréltem az RTR nevű algoritmusra. Ez is Kiss Domokos munkája [7], az implementációját Nagy Ákos végezte el.

[2]Intel 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 [8]. A szakirodalomban egyik leggyakrabban használt ilyen algoritmus a *Rapidly Exploring Random Trees* [9]. 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.

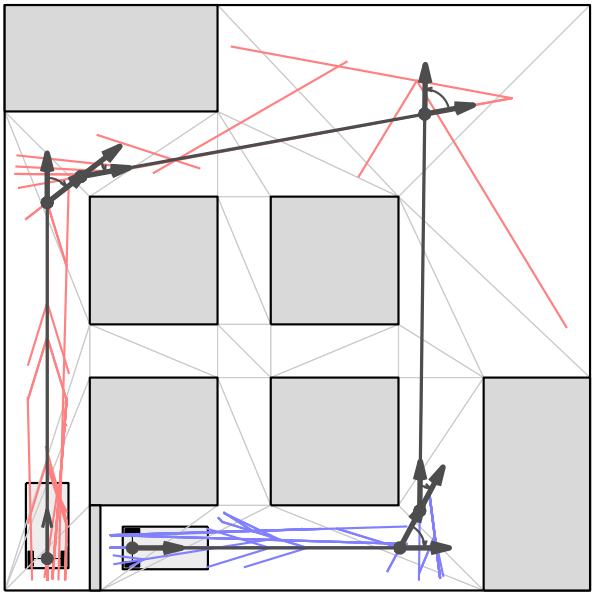


2.8. ábra. *Az RRT algoritmus három különböző iterációnál [9].*

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 növelve a megoldás megtalálásának lehetőségét.

Számomra ez az algoritmus azért előnyös, mert 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. Fekete a megoldást jelöli.*

3. fejezet

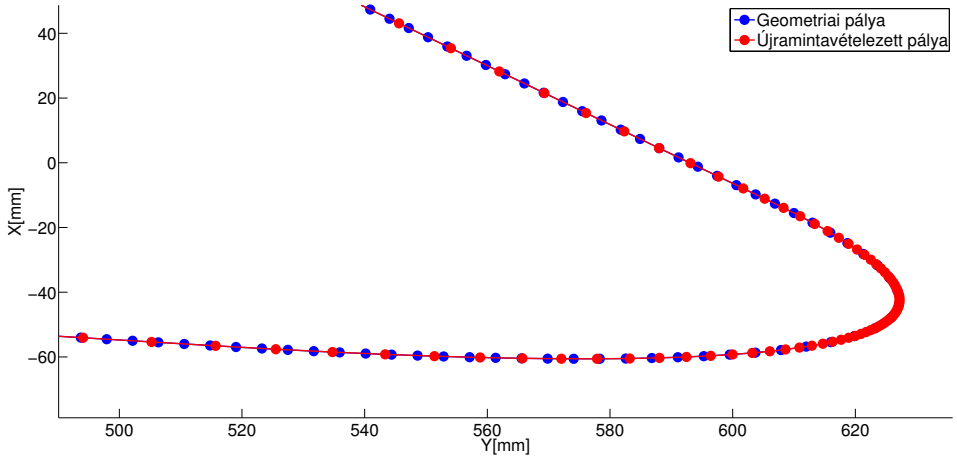
Pálya időparaméterezése

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 elsősorban a robot korlátait használja fel, de arra is alkalmas, hogy meghatározzuk a pálya bejárásának idejét.

A módszert Nagy Ákos dolgozta ki differenciális robotokhoz, majd én módosítottam autószerű robotok számára. Az elveket végül úgy alakítottuk ki közösen, hogy az ne függjön 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 találhatunk [10]. 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 [11]. 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 (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.

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

Azokban az esetekben, amikor a robot egyik kerekére vonatkozó mennyiségekről beszélünk, külön jelöljük, hogy bal (l), jobb (r), első (f) vagy hátsó (r) kerékről van szó. Ezenkívül a kerekeknél megkülönböztetjük, hogy tangenciális (a_t), centripetális (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 egyenessel kötnénk össze a pályapontokat, akkor a görbületnek szükségszerűen nullának kellene lennie.

3.3. Korlátozások

A robot mozgását általános esetben a **??**. ábra mutatja be. Az időparaméterezés során figyelembe vesszük a robot pályamenti sebességét és szögsebességét valamint a robot kerekeinek tangenciális és eredő gyorsulását. Hogy ezeket minden kerékre ki tudjuk számolni fel kell írnunk a robotunk mozgásegyenletét.(1.2) harmadik egyenletéből könnyen adódik, hogy a referenciapont által bejárt kör sugara az alábbi módon számítható:

$$
\rho = \frac{L}{\tan\phi} \tag{3.2}
$$

innen a hátsó kerekek által bejárt kör sugara a következőképpen adódik:

$$
\begin{aligned}
\rho_{rl} &= \rho - \frac{d}{2} \\
\rho_{rr} &= \rho + \frac{d}{2},
\end{aligned} \tag{3.3}
$$

Ahhoz, hogy fordulás közben ne csússzanak meg oldalirányba az első kerekek, a két oldali keréknek különböző szögben kell állnia. Ezt nevezzük Ackermann-hajtásnak. Ez a különbség csak a különböző fordulókörrel áll összefüggésben, és a következőképpen számolható a

kormányszögből:

$$\phi_r = \arctan\left(\frac{L}{\rho - \frac{d}{2}}\right) \tag{3.4}$$
$$\phi_l = \arctan\left(\frac{L}{\rho + \frac{d}{2}}\right)$$

Ezekből a következő módon számítható az első kerekek fordulókörének sugara:

$$\rho_{fl} = \frac{\rho - \frac{d}{2}}{\cos\phi_l} \tag{3.5}$$
$$\rho_{fr} = \frac{\rho + \frac{d}{2}}{\cos\phi_r}$$

Könnyen belátható, hogy a kerekek sebességkülönbsége csak a különböző fordulókörökből fakad, és ez hasonlóan igaz a kerékgyorsulásokra is, tehát egy adott pályapontban a pálya görbületéből, és a robot tangenciális sebességéből kiszámítható bármely kerék gyorsulása is.

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

$$v^{max} : \text{A robot pályamenti sebesség korlátja} \tag{3.6}$$
$$\omega^{max} : \text{A robot szögsebesség korlátja}$$
$$a_w{}^{max} : \text{A robot egy kerekének maximális gyorsulás korlátja}$$
(3.7)

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

$$a_{max} = \mu_{tap_{max}} \cdot g, \tag{3.8}$$

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

$$a(k) = \sqrt{a_{wc}(k)^2 + a_{wt}(k)^2} \leq g \cdot \mu_{tap}, \tag{3.9}$$

ahol $a_{wc}(k)$ egy kerék centripetális gyorsulása és $a_{wt}(k)$ a tangenciális gyorsulása

(3.9) egyenletben azzal a feltevéssel élünk, hogy a robot kerekei és a talaj között a tapadási súrlódási együttható állandó és nem függ az erő irányától. Az általunk használt differenciális robotnál ez a közelítés megengedhető, mivel a gumikerekek homogénnek 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 korlátozások. Így azt tesszük 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

3.5. Újramintavételezés

4. fejezet

Pályakövető szabályozás

5. fejezet

Algoritmusok megvalósítása

6. fejezet

Megvalósítás valós roboton

Köszönetnyilvánítás

Ez nem kötelező, akár törölhető is. Ha a szerző szükségét érzi, itt lehet köszönetet nyilvánítani azoknak, akik hozzájárultak munkájukkal ahhoz, hogy a hallgató a szakdolgozatban vagy diplomamunkában leírt feladatokat sikeresen elvégezze. A konzulensnek való köszönetnyilvánítás sem kötelező, a konzulensnek hivatalosan is dolga, hogy a hallgatót konzultálja.

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] G. Katona, A. Recski, and C. SzabĂł, *A számítástudomány alapjai*. Typotex, 2. javított kiadás ed., 2001.

[7] 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.

[8] 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.

[9] S. M. LaValle, „Rapidly-exploring random trees: A new tool for path planning," tech. rep., Computer Science Dept., Iowa State University, 1998.

[10] C. Sprunk, *Planning Motion Trajectories for Mobile Robots Using Splines*. Albert-Ludwigs-Universitat Freiburg, 2008.

[11] D.-J. Kroon, „2d line curvature and normals." `http://www.mathworks.com/matlabcentral/fileexchange/32696-2d-line-curvature-and-normals/content/LineCurvature2D.m`, 2014.

Függelék