Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

Papp Dávid

ÖNVEZETŐ AUTÓK PÁLYATERVEZÉSE SZŰK KÖRNYEZETBEN, FOLYTONOS GÖRBÜLETŰ PÁLYAELEMEKKEL

KONZULENSEK

Kiss Domokos

Csorvási Gábor

BUDAPEST, 2016

Tartalomjegyzék

Kivonat

Dolgozatomban autószerű mobil robotok pályatervezésével foglalkozom, ami fontos részfeladata az önvezető autók fejlesztésének. A hagyományos autóipari gyártók mellett egyre több szoftvergyártó folytat ilyen irányú kutatásokat, hiszen sokan úgy gondolják, hogy a következő évek egyik legígéretesebb változását hozhatja el az emberiség számára, ami kényelmesebbé és biztonságosabbá teszi a közlekedést.

Dolgozatomban többféle pályatervező algoritmust mutatok be, amiket akadályokat tartalmazó környezetben lehet használni. Approximációs pályatervezési eljárást használok, amelynek köszönhetően különböző globális és lokális tervezőket össze lehet kapcsolni és a kapott pálya ütközésmentesen bejárható lesz, valamint biztosítja a lokális tervező által a modell kinematikai korlátozásait is.

A lokális pályatervező figyelembe veszi a mobil robot kinematikai korlátozásait, ami a valós autóknál nem csak a fordulási sugár minimumát jelenti, hanem azt is, hogy a görbületnek folytonosnak kell lennie. Kétféle ilyen típusú pályatervező algoritmust vizsgálok, az egyik az irodalomból ismert CCRS (Continuous Curvature Reeds and Shepp), a másik az Automatizálási és Alkalmazott Informatikai Tanszéken fejlesztés alatt lévő T*TS tervező. Közös bennük, hogy egyszerűbb algoritmusok általánosításával, illetve módosításával jöttek létre. Előbbi az autószerű robotok körében klasszikusnak számító Reeds–Shepp pályákon alapul, utóbbi pedig a tanszéken korábban fejlesztett C*CS pályatervező általánosítása. Mindkét eljárás azonos megközelítést használ a folytonos görbület eléréséhez: az egyenes és körív alakú pályaelemeket klotoid görbékkel kötik össze. Mindkét lokális pályatervező bizonyítottan konvergens approximációs algoritmust eredményez.

Dolgozatomban részletesen bemutatom a T*TS tervezési módszert, illetve bemutatom mindkét lokális pályatervező működését az approximációs algoritmus részeként, többféle globális pálya esetére alkalmazva. Az algoritmusokat egy C++ nyelvű függvénykönyvtárban implementáltam. Kidolgoztam egy tesztelési eljárást, amely több szempont alapján összehasonlítja a globális és lokális pályatervezők kapcsolatát, és kvantitatív értékelésre ad lehetőséget az algoritmusok performanciája és az eredményezett pályák minősége szempontjából.

Abstract

In my thesis, I examine the path planning of car-like robots that is a vital subtask of developing autonomous cars. Besides the traditional automobile manufacturers, many software developers conduct research related to the aforementioned topic, as it is considered to be one of the most promising development for a more comfortable and more secure transportation.

Various path planning algorithms are explained in my study that can be utilized in an environment populated with obstacles. Owing to the approximation path planning method I'm investigating, different global and local planners can be connected in order to avoid collision on the path, and also ensure the kinematic constraints of the model.

The kinematic constraints of the mobile robot are being taken into account by the local path planner, that means not only the minimal turning radius of real cars, but also the continuity of the curvature. I examine two types of such path planning algorithms: the CCRS (Continuous Curvature Reeds and Shepp), and the T*TS planner that is currently under development at the Department of Automation and Applied Informatics. Both of them are created by generalizing and altering simpler algorithms. The CCRS is based on the so-called Reeds-Shepp paths that is widely used by car-like robots, meanwhile the T*TS is the generalization of the C*CS path planner that is previously developed at the department. Both methods apply the same approach for reaching the continuous curvature: the straight and arc shaped path elements are connected with clothoid curves. Both of the local path planners are proved to result in a convergent approximation algorithm.

The T*TS planning method is explained in detail, and the operation of both local path planners is presented as part of an approximation algorithm for various global paths. I implemented the algorithms as part of a C++ library. Beyond that I developed a testing process that compares the relation of the presented local planners with global planners from different perspectives, and it makes the quantitative evaluation of the performance of algorithms and the quality of the resulting paths possible.

1 Bevezetés

Dolgozatomban olyan pályatervező algoritmusokat vizsgálok, amelyek autószerű mobil robotok számára terveznek folytonos görbületű pályát akadályokat tartalmazó környezetben. Az implementáció C++ nyelven történt, egy tanszéki pályatervező függvénykönyvtárba illesztettem a saját megoldásaimat. A program írásakor, ha lehetőségem volt, akkor mindig a könyvtár függvényeit használtam, illetve a saját osztályaimat úgy írtam meg, hogy a többi osztállyal kompatibilis legyen.

A második fejezetben egy rövid elméleti bevezetéssel kezdem a téma ismertetését. Az alapvető definíciókon túl bemutatom a különböző pályatervezők csoportosításának lehetőségeit, valamint az autószerű robot kinematikai modelljét, amihez a későbbiekben definiálom a kinematikai korlátozásokat.

A harmadik fejezetben a dolgozatom kulcstémáját részletezem, a folytonos görbületű pályákat. Szemléletes példákkal és ábrákkal illusztrálom, hogy miért célszerű ilyen típusú pályákat tervezni az autószerű robotok számára, majd még ebben a fejezetben ismertetem azt a geometriai elemet, amelyet felhasználva a kívánt simaságú görbéhez jutunk. Az elem tulajdonságain és a leírását szolgáló matematikai képleteken túl egy eljárást is adok arra, hogy az elem felhasználásával minden pályaponton teljesüljenek a kívánt feltételek.

A következő két fejezetben a CCRS és T*TS lokális pályatervezőkről írok, különös hangsúlyt fektetve a pályatípusaikra, amelyeknek tervezési lépéseit részletesen leírom. Mindkét pályatervező hasonló jellegű pályát tervez és a felhasznált elemek is ugyanazok, ezért a dolgozatom végén több szempontból is összehasonlítom őket, keresve a választ, hogy melyiket milyen környezetben célszerű választani. Előtte azonban még a felhasznált globális tervezőket is bemutatom, az approximációs eljárással egyetemben.

2 Pályatervezés elmélete

2.1 Problémafelvetés

A mobil robotok alapfeladata, hogy képesek legyenek megtervezni azt a pályát, ami során eljutnak a start pozíciójukból a kívánt célpozícióba. A feladat nehézsége abban rejlik, hogy a robot általában nem tudja tetszőlegesen változtatni a mozgásállapotát, a különböző fizikai kényszerek és korlátozások miatt. A probléma bonyolultsága tovább nő, ha a térben akadályok találhatóak, amik csökkentik a robot mozgásterét.

Sok kutatásnak volt a fő témája a mobil robotok és azon belül is a pályatervezés, és az önjáró autók kapcsán a népszerűségük valószínűleg tovább fog nőni az elkövetkezendő néhány évben. Mielőtt részletesen kitérek az algoritmusokra, érdemes pár fogalmat bevezetni.

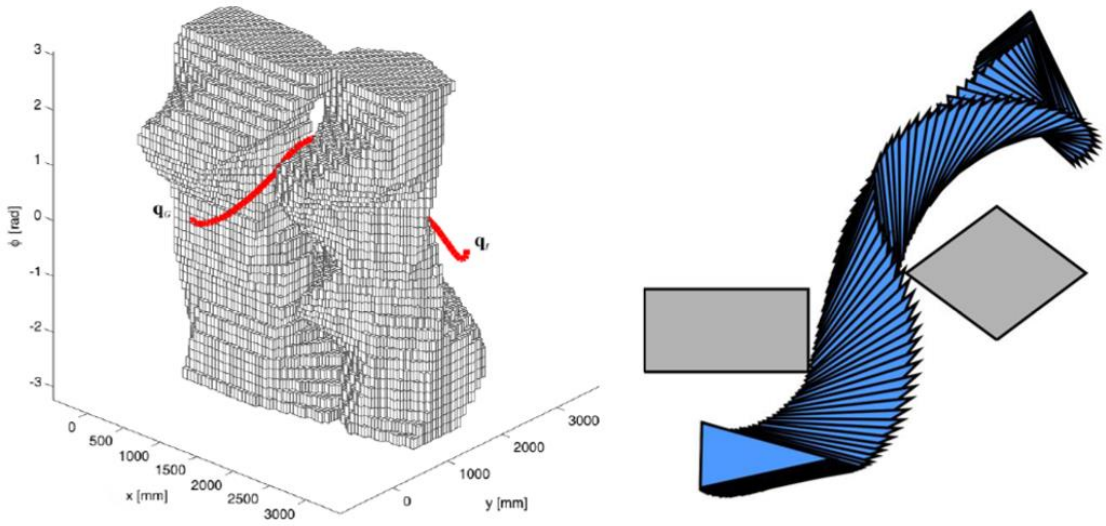
2.2 Fogalmak

Mesterszakos villamosmérnöki tanulmányaim, konzulensem útmutatása és diplomamunkája alapján [1] csak azokat a fogalmakat tárgyalom, amik szorosan a dolgozatomhoz kapcsolódnak. A téma iránt érdeklődő olvasók figyelmébe ajánlom a [1] irodalmat.

A robot aktuális állapotát a konfigurációja írja le, ami síkban mozgó robotoknál a kétdimenziós pozíció koordinátáit és az orientációt jelöli:

$$q = (x, y, \theta) \tag{2.1}$$

A robot összes állapotát egy adott környezetben a konfigurációs tér írja le, ami a lehetséges konfigurációk halmazát tartalmazza. A konfigurációs tér két diszjunkt részhalmazból áll, az egyik az akadályokat (C_{obst}), a másik az akadályoktól mentes, a robot által szabadon elérhető konfigurációkat tartalmazza (C_{free}). A 2.1. ábrán a konfigurációs térre látható egy példa. A kék háromszög alakú robot ütközésmentes pályája látható a baloldalon, jobb oldalon pedig ennek a pályának a konfigurációit jeleníti meg a piros görbe.

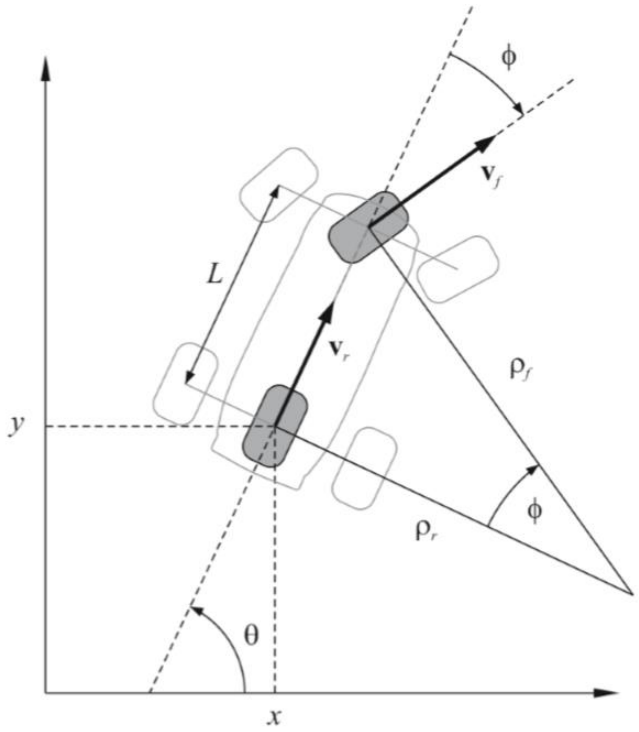


2.1. ábra. Síkban mozgó mobil robot konfigurációs tere [3]

A bevezetőben említettem, hogy a pályatervezést a különböző korlátozások teszik bonyolult problémává. Két csoportra oszthatjuk ezeket. A környezetben található akadályok globális korlátozásokat, míg a mobil robot fizikai tulajdonságaiból adódóak lokális korlátozásokat jelentenek. A lokális korlátozásokat differenciális korlátozásoknak is nevezzük, mert a mobil robot konfigurációs változóinak differenciális egyenleteivel írjuk le azokat. A differenciális korlátozás jelenthet dinamikai és kinematikai megkötést is. Dolgozatomban én csak az utóbbiakkal foglalkozom, azaz csak a sebesség jellegű korlátozásokkal számolok, a gyorsulás jellegűekkel nem.

Pályatervezés szempontjából megkülönböztetünk globális pályatervezést, amikor az akadályok figyelembe vételével tervezünk pályát, illetve lokális pályatervezést, amikor akadályoktól mentes környezetben, csak a mobil robot kinematikai megkötéseit tartjuk be. Előbbi számításigénye jelentős lehet az algoritmus hatékonyságától és a környezet bonyolultságától függően.

Dolgozatomban autószerű mobil robotokat vizsgálok, amelyek anholonom rendszernek tekinthetőek modellezés szempontjából. Szemléletesen ez azt jelenti, hogy az adott konfigurációból nem tudunk tetszőlegesen a konfigurációs tér bármely irányába elmozdulni. Kinematikai feltételként megfogalmazva, az autónak van egy minimális fordulási sugara, aminél kisebb kanyart nem tud bevenni. Az autószerű mobil robot modellje a következő ábrán látható:



2.2. ábra. Autószerű robot modellje [1]

$$\dot{x} = v_r \cos\theta \tag{2.2}$$

$$\dot{y} = v_r \sin\theta \tag{2.3}$$

$$\dot{\theta} = \frac{v_r}{L}\tan\varphi \tag{2.4}$$

L az első és hátsó tengelyek távolsága, φ a kormányszög, v_r pedig a hátsó tengely középpontjának tangenciális sebessége, amelyet a robot referenciapontjának nevezünk, és θ szöget zár be az x tengellyel.

3 Folytonos görbületű pályatervezés

A CCTurn egy elemi pályaszakasz a lokális pályatervezésnél, amit az [3] publikáció alapján mutatok be. A CCTurn tervezésekor az alkotók fő célja a folytonos görbület elérése a pálya minden egyes pontján, de emellett törekednek a lehető legrövidebb pályahosszra viszonylag kevés számítás mellett. A pályatervező algoritmust részletesen a következő fejezetben írom le, de előtte ismertetem a CCTurn tulajdonságait és alkalmazásának az okait.

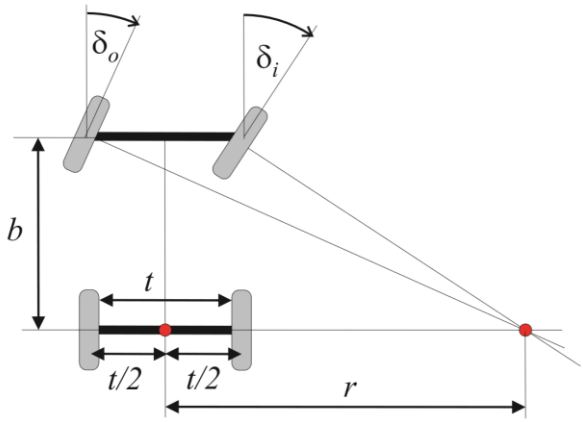
3.1 Kinematikai feltételek

Egy mobil robot pályájának megtervezésekor fontos, hogy olyan görbét tervezzünk, amit az adott robot be tud járni, vagyis a pályatervező algoritmusnak figyelembe kell vennie a modell által előírt kinematikai korlátozásokat. Autószerű robotoknál a pálya alakjára vonatkozóan két kinematikai korlátozást érdemes hozzáadni a modellhez annak érdekében, hogy a valóságot megfelelően közelítsük.

3.1.1 Minimális fordulási sugár

Az első triviális feltétel, amit már a legelső algoritmusok is használtak, a kormányszög felső korlátjából adódó minimális fordulási sugár. Ezt könnyen kiszámolhatjuk [5] a jármű fizikai paramétereiből, ha feltesszük, hogy a kerekek egy pont körül fordulnak. A járműveknek ezt a tulajdonságát az autógyártók is próbálják minél pontosabban megvalósítani, hogy a kanyarstabilitást növeljék.

A 19. században Rudolph Ackermann lovaskocsikat vizsgált és megállapította, hogy ideális esetben az első kerekek elfordulási szöge különböző. Ennek oka, hogy a különböző ívhosszt bejáró kerekekhez különböző fordulási sugár tartozik, ha a középpontjuk megegyezik. Ellenkező esetben a kerekek nem csúszásmentesen gördülnek és csökken az irányíthatóság. Az elmélet gyakorlati megvalósítása a kormánytrapéz és a szakirodalom [5], Ackermann kormányzásként is hivatkozik rá.



3.1. ábra. Ackermann kormányzás [5]

$$\tan \delta_i = \frac{b}{r - t/2} \tag{3.1}$$

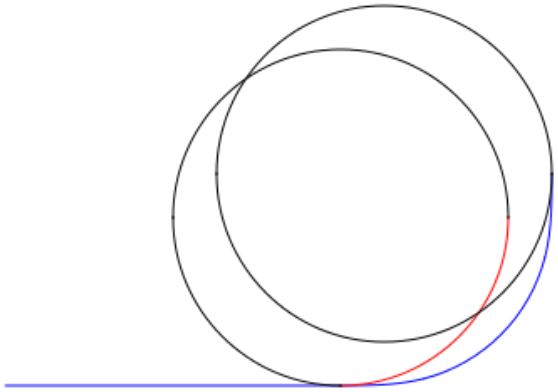
$$\tan \delta_o = \frac{b}{r + t/2} \tag{3.2}$$

$$r = \frac{b}{\tan \delta} \tag{3.3}$$

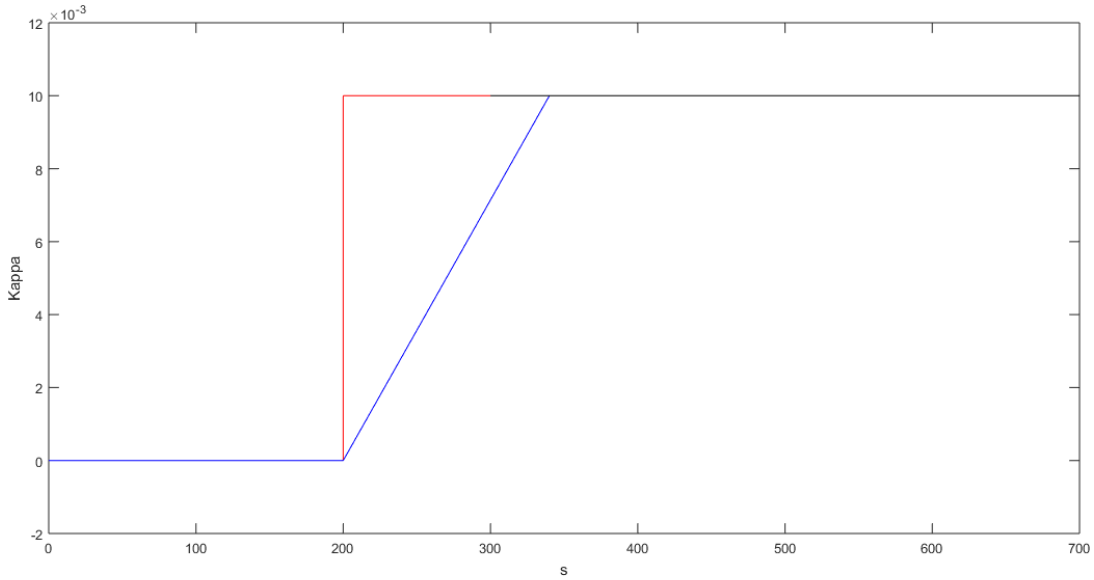
A szimmetriatengelyre helyezett kerékpár modell segítségével megkaphatjuk a fordulási sugár nagyságát.

3.1.2 Folytonos görbület

Autószerű mobil robotoknál a görbület maximuma mellett (a minimális fordulási sugár reciproka), célszerű annak folytonosságát illetve változási gyorsaságának maximumát is előírni a kinematikai korlátozásoknál. Mindkét feltétel szükséges ahhoz, hogy a valós járművekkel pontosan bejárható pályát kapjunk, anélkül hogy az autónak meg kellene állnia a különböző görbületű pályaelemeknél. Ennek az állításnak a magyarázatát segíti a következő ábra.



3.2. ábra. Különböző görbületű pályaelemek kapcsolódása



3.3. ábra. Pályaelemek görbülete

A 3.2. ábrán kétféle kapcsolódást láthatunk arra, hogy hogyan lehet összekötni egy egyenest egy körrel. Az egyik lehetőség, hogy a kört (piros) úgy illesztjük a szakaszhoz, hogy az az érintője legyen. A második lehetőség (kék görbe), hogy egy speciális görbével fokozatosan „kanyarodunk rá” a körívre (fekete). Jól látszik, hogy mindkét esetben ugyanolyan sugarú körre (fekete) állunk rá a képzeletbeli autónkkal, és első ránézésre talán nem magától értetődő, hogy miért járunk jobban, ha a második megoldást választjuk.

A 3.3. ábrán a pálya görbülete látható (a két ábra közti megfeleltetést a színek segítik). Első esetben a nulla görbületű egyenesről ugrásszerűen váltunk a körív görbületére. Ez a függvényben elsőfajú nem megszüntethető szakadást eredményez. A valóságban pedig a járműveknél végtelen nagy energiájú kormány szervó beavatkozásra lenne szükség, ugyanis adott fordulási sugárhoz adott kormányszög tartozik, és ennek az időbeli deriváltja tart a végtelenhez, ha δt tart nullához. A gyakorlatban ennek elkerülése megoldható, ha a görbület ugrásoknál megáll az autó és addig nem indul el, amíg be nem állnak a kerekek a megfelelő irányszögbe. Ez persze a valóságban utazási sebességnél már nem megoldás, és parkoláskor is körülményes, valamint a döcögős vezetés csökkenti az utasok komfort érzetét.

A második megoldásnál viszont a görbület függvénye folytonos, változási gyorsasága pedig megfelel a kormánykerék szögsebességének, ezért érdemes a deriváltját felülről korlátozni. A görbületre vonatkozó kinematikai korlátozás a következő:

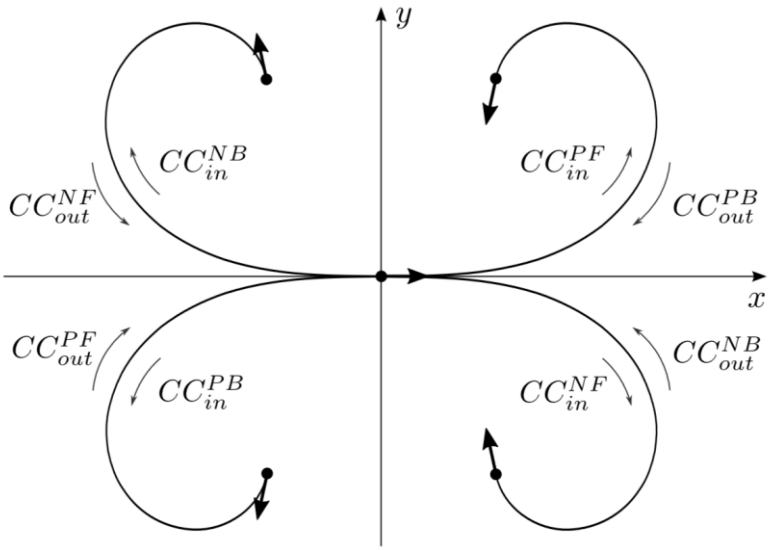
$$\kappa(t) = \sigma_{max} t + \kappa_0 \tag{3.4}$$

$$\dot{\kappa}(t) = \sigma_{max} \tag{3.5}$$

Ha σ_{max}-t konstansnak választjuk, akkor a görbület függvénye elsőfokú és folytonos lesz, deriváltja pedig korlátos. Ennek gyakorlati hasznát az előző fejezetekben beláttuk. A probléma matematikai megközelítése egy megfelelően sima görbe számítását jelenti. Erre különféle matematikai apparátusok is a rendelkezésünkre állnak (pl.: B-Spline vagy egyéb Spline-ok), de a parametrizálhatóság miatt a CCTurn klotoid görbéket használ.

3.2 Klotoid

A klotoid görbülete lineárisan változik az ív hosszával, és ezt a változási gyorsaságot σ_{max}-szal tudjuk megválasztani. Attól függően, hogy kisebb görbületről nagyobbra vagy nagyobbról kisebbre történik az átállás, két típusát különböztetjük meg. A fordulás és haladás iránya szintén két-két csoportra osztja a lehetőségeket, összesen tehát nyolcféle klotoidról beszélhetünk, amelyek a következő ábrán láthatóak.



3.4. ábra. Klotoid típusok [7]

Pontjainak számolása, ha $q_0 = [0,0,0,0]^T$ konfigurációból indul:

$$x = \sqrt{\frac{\pi}{\sigma_{max}}} C_f\left(\sqrt{\frac{\kappa^2}{\pi\sigma_{max}}}\right), \qquad C_f(a) = \int_0^a \cos\left(\frac{\pi}{2}t^2\right)dt \tag{3.6}$$

$$y = \sqrt{\frac{\pi}{\sigma_{max}}} S_f\left(\sqrt{\frac{\kappa^2}{\pi\sigma_{max}}}\right), \qquad S_f(a) = \int_0^a \sin\left(\frac{\pi}{2}t^2\right)dt \tag{3.7}$$

$$\theta = \frac{\kappa^2}{2\sigma_{max}} \tag{3.8}$$

$$\kappa = \sigma_{max} t \tag{3.9}$$

C_f és S_f a Fresnel integrálokat jelöli. Nincs zárt alakban megoldásuk, ezért a kiszámításukhoz sorfejtés alapú közelítést alkalmaztam:

$$S_f(x) = \sum_{i=0}^{\infty} (-1)^i \frac{x^{4i+3}}{(2i+1)!\,(4n+3)} \tag{3.10}$$

$$C_f(x) = \sum_{i=0}^{\infty} (-1)^i \frac{x^{4i+1}}{(2i)!\,(4n+1)} \tag{3.11}$$

Az első húsz tag kiszámolása megfelelően pontos eredményt adott, azonban a program során ez a függvény nagyon sokszor lefut, ezért a kód optimalizáltsága jelentősen hat az egész pályatervező algoritmus futási idejére. Célszerű az argumentumtól független tagokat előre kiszámolni és egy tömbben eltárolni. Megjegyzendő, hogy a fenti sor nem teljesen azonos a klotoidnál használt integrállal, ott ugyanis normalizált Fresnel integrál van, ami az eredeti görbének egy skálázott változata. A sorral történő közelítéskor ezért szükséges az argumentumot π/2 négyzetgyökével osztani, a kapott eredményt a végén pedig ugyanezzel a számmal szorozni kell.

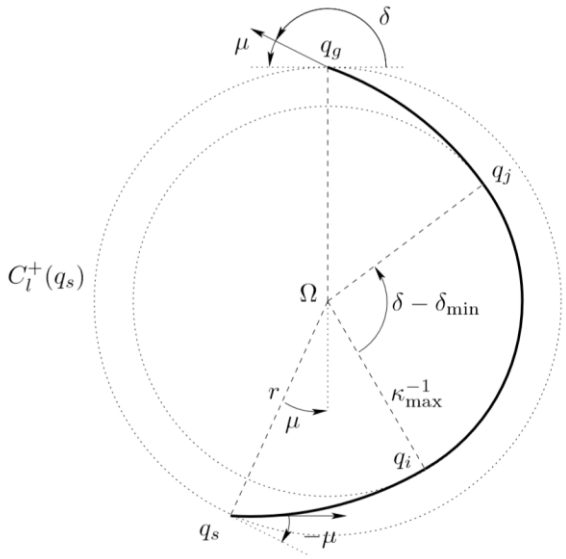
Eltérő pozícióból induló klotoidnál a számolás lépései a következő sorrendben történnek. Először mindig a függvény alapján kapott pontokat számolom, majd a klotoid típusának megfelelően tükrözöm az *x* és/vagy *y* tengelyre. Végül egy transzformációs mátrixszal a kívánt konfigurációba helyezem. Utóbbi lépésnél megvan a lehetőségem, hogy a klotoidot a kezdő, illetve a végpontja szerint is transzformálhatom, amit a pályatervező algoritmusban ki is fogok használni.

3.3 CCTurn felépítése

A CCTurn egy klotoiddal kezdődik, körívvel folytatódik, és végül egy, az elsővel egybevágó, klotoiddal végződik. A körív hossza speciális esetben nulla is lehet, illetve irányítottsága tetszőleges, azaz az autó a klotoid és a körív találkozásánál előrementből hátramenetbe (vagy fordítva) válthat. Ezek alapján három fajtája létezik a CCTurn-nek. Közös bennük, hogy az elején nulla görbületről indulnak és a végén szintén nulla görbületre állnak be. Ez a tulajdonságuk lehetővé teszi, hogy szakaszokkal és más CCTurn-ökkel egyaránt összekapcsolhatóak úgy, hogy az egész pálya görbülete folytonos marad.

A korábbi pályatervező algoritmusok által tervezet görbét folytonossá tehetjük, ha a körívek helyett CCTurn-öket alkalmazunk a szakaszok mellett. Természetesen egy egyszerű cserével nem kapunk jó megoldást, ugyanis a CCTurn-öket máshogy kell kapcsolni az egyenesekhez, mint ahogy az a 3.2 ábrán is jól látható. Mielőtt erre

kitérnék, bemutatom a CCTurn geometriai felépítését, és az azt leíró matematikai egyenleteket.



3.5. ábra. CCTurn általános esetben [3]

A könnyebb tárgyalhatóság érdekében a $q_s = [0\ 0\ 0\ 0]^T$ startkonfigurációt az origóba helyeztem, később a tervezés során ez tetszőleges konfigurációba eltranszformálható. A három elem tulajdonságai start-cél bejárási irányt figyelembe véve a következőek. A startkonfigurációból egy nulla kezdő görbületű klotoid indul és a q_i konfigurációba érkezve eléri a maximális görbületet (κ_{max}), amit még a mobil robot modelljének kinematikai korlátozása megenged. Ennek a pontnak az értékei a következőek (a klotoid képleteibe κ helyére κ_{max} kell helyettesíteni):

$$q_i = \begin{cases} x_i = \sqrt{\frac{\pi}{\sigma_{max}}} C_f\left(\sqrt{\frac{\kappa_{max}^2}{\pi\sigma_{max}}}\right) \\ y_i = \sqrt{\frac{\pi}{\sigma_{max}}} S_f\left(\sqrt{\frac{\kappa_{max}^2}{\pi\sigma_{max}}}\right) \\ \theta_i = \frac{\kappa_{max}^2}{2\sigma_{max}} \\ \kappa_i = \kappa_{max} \end{cases} \tag{3.12}$$

Megjegyzendő, hogy κ_{max} és σ_{max} a mobil robotra jellemző paraméter, ezért a Fresnel integrálokat pályatervezéskor csak egyszer kell kiszámolni az adott robotra, és a konfiguráció számításoknál ezt az eredményt célszerű használni, így csökkentve az algoritmus futási idejét. Fontos azonban tisztázni, hogy ez csak a lokális tervezést segíti és csak abban az esetben, ha a robot mindig a lehető legélesebben kanyarodik. Továbbá kiemelném, hogy globális pályatervezéskor szükséges az összes konfigurációs pontot

kiszámolni az ütközés detektáláshoz, és ilyenkor a klotoid képletében κ nulla és a κ_{max} között (felbontástól függően) véges sok értéket vesz fel.

A következő elem egy κ_{max} görbületű körív, q_i-tól q_j-ig tart. A hajlásszög nagyságára később visszatérek. Legvégül szintén egy klotoid található q_j-től q_g-ig, hasonló paraméterekkel mint az első, csak most a maximális görbületről indulva kiegyenesedik nulla görbületre.

Vegyük sorra a CCTurn paramétereit:

- δ: A start és a célkonfiguráció közti orientáció változását jelöli.
- δ_{min}: A klotoidok bejárása során történő orientáció változást jelöli. (Megjegyzés: a körív szöge δ-δ_{min} –mal egyenlő.)
- Ω: A CCTurn középpontja, ekörül fordul a robot a minimális sugarú köríven ($1/\kappa_{max}$).
- r: A külső kör sugara (a következő fejezetben kitérek rá.)
- μ: A külső kör érintője és a konfiguráció közti szögkülönbség.

A CCTurn külső köre (lásd 3.5. ábra) tulajdonképpen egy segédkör a pályatervezéskor és a CCTurn tulajdonságainak bemutatásakor, de ezen a körön az autó soha nem fordul. Azt szemlélteti, hogy a start és cél pont rajta van ezen a körön, de az orientációjuk mindig konstans μ szöggel tér el az adott pontban a körhöz húzható érintő egyenestől. Ez egy fontos tulajdonság, ami pályatervezéskor segíti a különböző elemek folytonos görbületű kapcsolódásának számítását. A CCTurn paraméterek számítása a következő a klotoid paramétereit felhasználva, ha a kezdő pont az origó (δ-ra, mint változó érdemes tekinteni, ugyanis ezzel tudjuk megvalósítani a kívánt elfordulást):

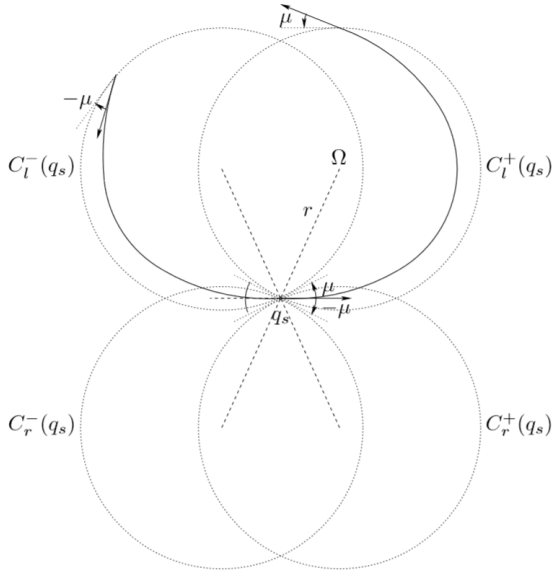
$$\Omega = \begin{cases} x_\Omega = x_i - \kappa_{max}^{-1} \sin\theta_i \\ y_\Omega = y_i + \kappa_{max}^{-1} \cos\theta_i \end{cases} \tag{3.13}$$

$$r = \sqrt{x_\Omega^2 + y_\Omega^2} \tag{3.14}$$

$$\mu = \operatorname{atan}(x_\Omega / y_\Omega) \tag{3.15}$$

$$\delta_{min} = \kappa_{max}^2 \sigma_{max}^{-1} \tag{3.16}$$

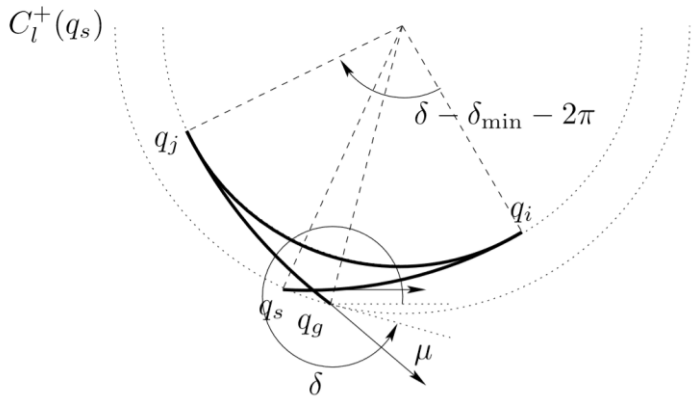
Még egy jellemző tulajdonsága van a CCTurn-nek, amiről a klotoid kapcsán már beszéltem, de itt még nem említettem. Ez pedig a haladási és kanyarodási iránytól függően a négy típusa (a 3.5. ábrán is látható C_l^+ jelölés, ami a balra előre mozgást jelenti). Ez a négy különböző típus egyenként is kiszámítható, de egy CCTurn kiszámítását követően transzformációval is előállítható a másik három típus. A következő ábrán a négy típus látható:



3.6. ábra. 4 féle CCTurn [3]

3.3.1 Nagy hajlásszögű CCTurn

A CCTurn bevezetésénél említettem, hogy lehetőség van az első klotoid után megváltoztatni a robot haladási irányát. Ezt akkor célszerű megtenni, hogyha (lásd 3.7. ábra) q_j pontból rövidebb körív járható be q_j pontig az irányváltoztatással, mint anélkül.

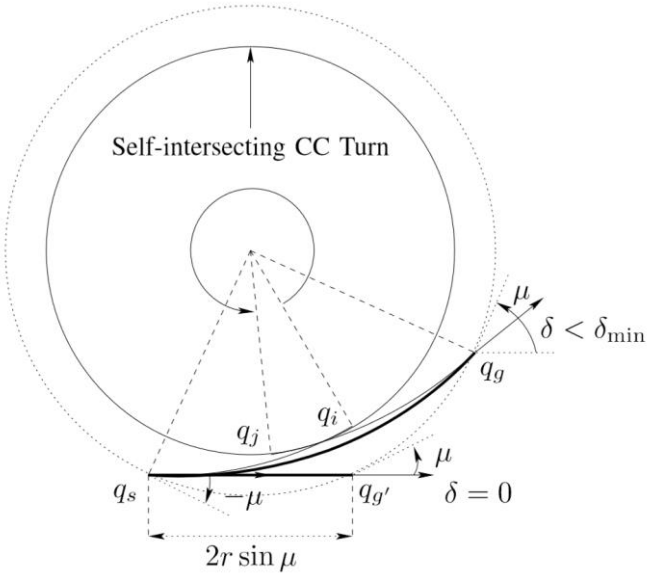


3.7. ábra. Nagy hajlásszögű CCTurn [3]

Geometriailag akkor érdemes irányt változtatni, ha a körívhez tartozó szög 180°-nál nagyobb. Azonban érdemes figyelembe venni, hogy a rövidebb út nem feltétlenül eredményez gyorsabban bejárható pályát a mobil robot számára, ugyanis az irányváltáskor történő lassítás és gyorsítás ideje adott esetben nagyobb a pályahossz különbségéből adódó időnyereségnél. Célszerű lehet meghatározni egy olyan 180°-nál nagyobb szöget melynél már biztosan megéri áttérni a nagy hajlásszögű CCTurn-re. Azonban implementáláskor eltekintettem a lehetséges időveszteségtől, mert nem kívántam a szerzők módszerétől eltérni, így az irányváltoztatásra vonatkozó előírásokat a publikált formájukban hagytam [3].

3.3.2 Kis hajlásszögű CCTurn

Korábban a CCTurn összetételénél írtam, hogy a körív hossza akár nulla is lehet. Persze ebben az esetben, az általános CCTurn-nél van δ_{min} orientációs hajlásszög változás a kezdeti- és a végkonfiguráció között, ami a klotoidok görbüléséből következik. Adódik a kérdés, hogy mi a teendő, ha ennél a minimális szögnél kisebbet fordulást szeretnék előidézni. Az első lehetőség az önmetsző kör alkalmazása, célszerűbb viszont ilyen esetben a fordulás élességét csökkenteni. Minkét megoldás az alábbi ábrán látható:



3.8. ábra. Kis hajlásszögű CCTurn [3]

Az élesség csökkentése egyúttal σ csökkentését is jelenti. Az új σ értéket az alábbi képlettel kaphatjuk meg:

$$\sigma = \frac{\pi\left(\cos(\delta/2)C_f\left(\sqrt{\delta/\pi}\right) + \sin(\delta/2)S_f\left(\sqrt{\delta/\pi}\right)\right)^2}{r^2 sin^2\left(\frac{\delta}{2} + \mu\right)} \tag{3.17}$$

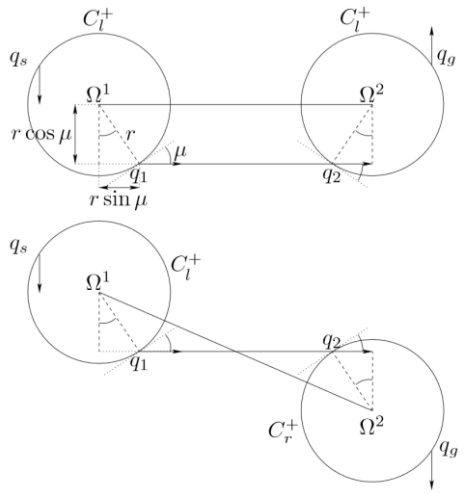
A képlet jól láthatóan számításigényes a Fresnel integrálok miatt. Használata viszont mindenképpen ajánlott, ugyanis összehasonlítva más megoldásokkal (pl. a 3.8. ábrán látható önmetsző kör alkalmazásával) sokkal gyorsabban bejárható pályát kapunk, és egyszerűsége miatt az ütközés esélye is sokkal kisebb. Ez végső soron a globális pályatervező algoritmusban kevesebb számolást eredményez.

3.4 Folytonos görbületű kapcsolódás biztosítása

A 3.2. ábra. kapcsán már említettem, hogy a CCTurn alkalmazása a körívek helyett lehetővé teszi a különböző pályaelemek közti átmenetnél is a folytonos görbület biztosítását. Ennek a kapcsolódásnak a geometriai feltételeit mutatom be ebben a fejezetben, majd a későbbiekben kitérek a C++ könyvtárba írt implementációs megoldásaimra is. Tervezési szempontból két esetre lehet osztani a feladatot, attól függően, hogy a CCTurn-ök közvetlenül vagy egy egyenes szakasz beiktatásával kapcsolódnak. Utóbbival kezdem a bemutatást.

3.4.1 CSC kapcsolódás

Ez az eset további két alesetre bomlik aszerint, hogy a második CCTurn iránya megegyezik-e az első CCTurn irányával. Ez a két lehetőség az alábbi ábrán látható.

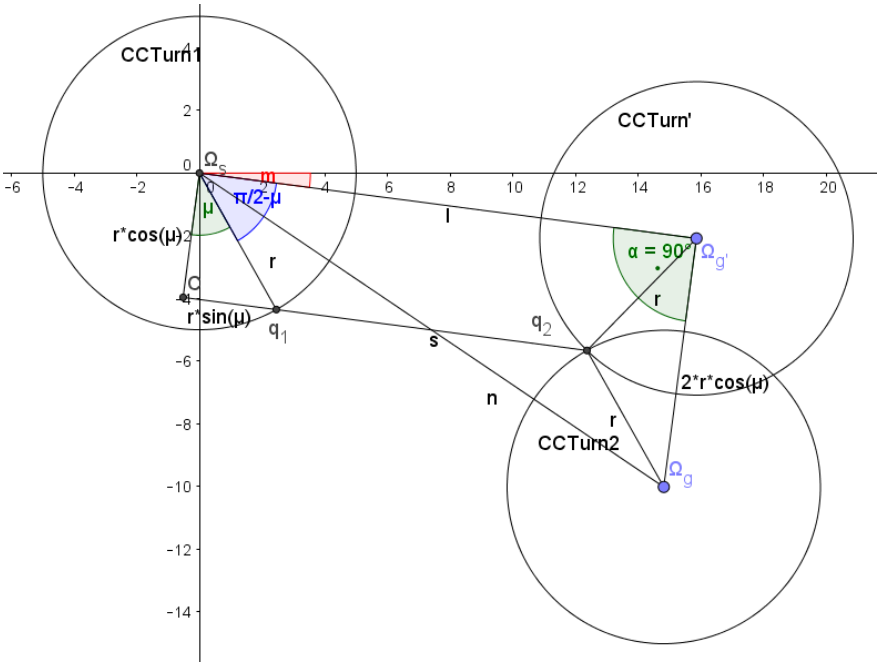


3.9. ábra. CSC kapcsolódások [3]

3.9. ábrán látható esetek közül a felső tárgyalása egyszerűbb, amikor a CCTurn-ök iránya megegyezik. Ilyenkor szükségszerű, hogy a szakasz iránya megegyezzen a CCTurn középpontok által meghatározott egyenes irányával. Keressük q_1 pontot, amiről tudjuk, hogy rajta van az első CCTurn külső körén, továbbá ismert a $q_1\Omega_1\Omega_2$ szög, ami egyenlő 90°-μ. (Ez az összefüggés abból adódik, hogy a CCTurn célkonfigurációja minden esetben konstans μ szöggel tér el a külső köréhez húzható érintő egyenestől. Az irodalom [3] „μ-tangent line”-ként hivatkozik az ilyen típusú érintőre.) Ezek az információk q_1 konfigurációt egyértelműen meghatározzák.

A q_1 konfigurációból indított egyenes metszéspontja a második CCTurn külső körével meghatározza q_2 pontot (q_1-hez közelebbi metszéspontot kell venni). Az iránya q_2 konfigurációnak megegyezik q_1 irányával. Ezek után ismert az összes szükséges konfiguráció q_S, q_1, q_2 és q_G, mind a három pályaelem szerkeszthető. A következő fejezetben, bemutatom, hogy hogyan oldottam meg a CCTurn pályák számolását a különböző paramétereiknek ismeretében.

Abban az esetben, ha a második CCTurn-nél irányváltás történik, akkor az összekötő szakasz végpontjainak számolása már jóval bonyolultabb, mint az előbbi esetben. Ezt a lehetőséget mutatja a 3.9. ábra. alsó pályaterve. A köztes konfigurációs pontok meghatározásához egy saját magam által készített GeoGebra ábrát használtam fel (3.10. ábra).



3.10. ábra. CSC kapcsolódás számolása

A megoldáshoz a problémát az előző feladatra vezettem vissza. Az ábrán szereplő CCTurn' egy segédelem, ami a következő tulajdonságokkal rendelkezik. Szintén a q_2 pontból indul, mint a CCTurn2, de iránya megegyezik a CCTurn1 irányával. Ez esetben az elhelyezkedésére az előbb leírt szabályok vonatkoznak. Ha minden egyes CCTurn2-höz egyértelműen létezik ilyen tulajdonságú CCTurn', akkor a feladatot sikerült visszavezetni egy már ismert problémára.

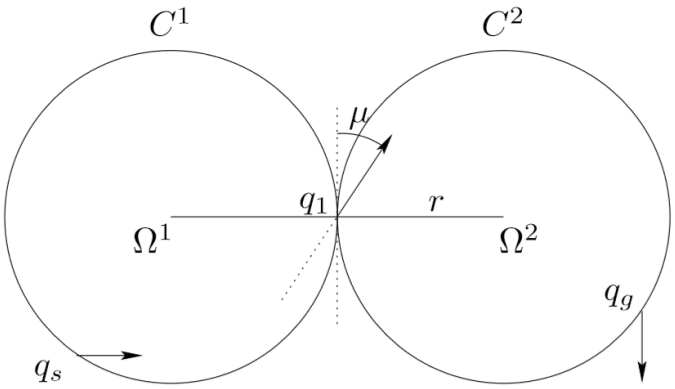
A megoldáshoz az egy pontból induló négyféle CCTurn típusok ábra nyújt segít (3.6. ábra). A CCTurn-ök q_s start pontját a q_2 konfigurációba transzformálva megkapjuk a CCTurn' helyzetét. Középpontját a következőképpen számolhatjuk. Ω_s és Ω_g pont ismert, $\Omega_{g'}$ rajta van $\Omega_s\Omega_g$ szakasz Thalész körén, mert a három pont egy derékszögű háromszöget alkot. (A derékszögűség a négy CCTurn középpontjainak elhelyezkedéséből és az egyirányú CSC kapcsolódásból adódik.) Ω_g és $\Omega_{g'}$ távolsága $2r\cos(\mu)$-vel egyenlő (későbbi kapcsolódásnál kitérek ennek a számolásnak a bizonyítására). Az Ω_g középpontú, $2r\cos(\mu)$ sugarú kör, a Thalész körön kimetszi $\Omega_{g'}$ pontot. Mivel két metszéspont adódik, ezért fontos, hogy az irányokat mindig ugyanúgy vegyük fel, és a megfelelő félsíkon lévő megoldást válasszuk.

Összefoglalva a szerkesztés menetét, Ωs és Ω_g ismeretében meghatároztam Ωg'-t, ami után a korábban ismertetett egyirányú CSC módszer segítségével megkaptam q_1 és q_2 pontot. Az elemek kezdő- és végkonfiguráció pontjainak ismeretében már szerkeszthető a pálya.

Eddig nem említettem, de a CSC típusú összekötések nem minden esetben valósíthatóak meg. Létezésük feltétele, hogy a középpontok távolsága legalább $2r\sin(\mu)$ vagy $2r$ nagyságú legyen attól függően, hogy van-e irányváltás vagy nincs (3.9. ábra).

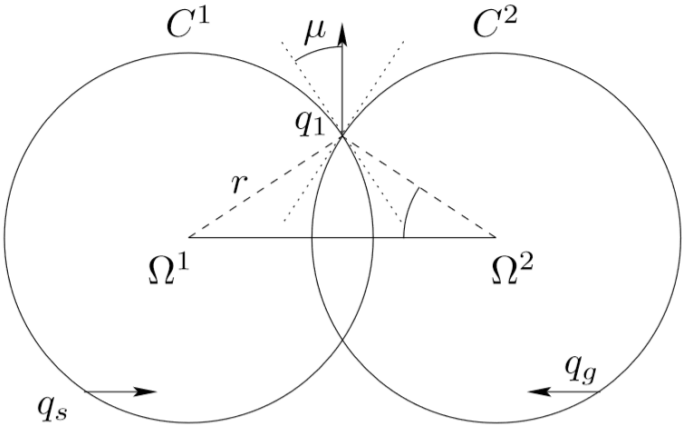
3.4.2 CC kapcsolódás

Két CCTurn közvetlen összekapcsoláskor mindig van irányváltás, (ha nem lenne, akkor elég lenne egy CCTurn) viszont a haladási irányt tekintve lehet változás, ezért eszerint két osztályt különböztetünk meg. Először vegyük azt, amikor nincs irányváltás. Ilyenkor a két CCTurn érintőlegesen kapcsolódik egymáshoz, és a metszéspontban a mobil robot a kanyarodását tekintve irányváltozás következtében áttér az egyik CCTurn-ről a másikra. Ezen eset létezésének triviális feltétele, hogy a két CCTurn középpont távolsága pontosan $2r$ nagyságú legyen (3.11. ábra).



3.11. ábra. CC kapcsolódás [3]

A második esetben irányváltás történik, ami az alsó ábrán látható (3.12. ábra). $\Omega_1\Omega_2q_1$ szög μ-vel egyenlő, ezért ezen konstrukció létezésének a szükséges és elégséges feltétele, hogy a CCTurn-ök középpontjai $2r\cos(\mu)$ távolságra legyenek egymástól. A CCTurn-ök megfelelő metszéspontjában a mobil robot fordulási és haladási irányt váltva áttér az első CCTurn-ről a másodikra. A helyes q_1 pontot úgy kapjuk meg, hogy Ω_1-ből Ω_2-be mutató vektor bal félsíkján lévő metszéspontot választjuk.



3.12. ábra. C|C kapcsolódás [3]

3.5 Implementáció

A CCTurn kapcsolódások leírása során többször említettem, hogy elég a kezdő- és a végkonfigurációt meghatározni, és utána már szerkesztethetőek a pályaelemek. Most ennek a C++ könyvtárbeli, általam készített implementációját mutatom be röviden. Tervezéskor felhasználtam a korábban elkészített osztályokat (pl.: pont, konfiguráció, szög, stb…) és függvényeket (transzformáció, távolság számolás, metszéspont keresés, stb…). Algoritmusaimat a könyvtárban kialakított sémának megfelelően építettem fel, illetve az objektumorientált programozás elvei szerint úgy

kódoltam le, hogy általánosan alkalmazhatóak legyenek. Így nem csak az eddig megírt részekkel működnek együtt, hanem a későbbiekben is könnyen felhasználhatóak.

A CCTurn elemnek létrehoztam egy osztályt, hasonló felépítés szerint, mint amilyen a többi pályaelemnek is van. A deklarációjában a mobil robot paramétereit kell megadni (maximális görbület és kanyarélesség) valamint a kanyar irányát (előre/hátra, balra/jobbra). Ezek ismeretében már lehet számolni a görbe paramétereit, ha feltesszük, hogy kezdetben az origóból indul. Főleg a CCTurn középpontjára, külső körének sugarára és μ-re van szükség, ugyanis ezekkel már a tényleges helyére transzformálható az objektum. A transzformáció történhet a kör középpontja szerint (kapcsolódáskor ezt többször is felhasználtam) vagy a kezdő/vég konfiguráció szerint.

Ha a középpont szerint történt a transzformáció, akkor se a kezdő-, se a végkonfiguráció nem ismert. Ebben az esetben következő lépésként az egyiket meg kell adni. Utolsó lépésben vagy a hiányzó konfigurációt vagy a CCTurn hajlásszögét kell megadni. Ezzel a CCTurn objektum teljessé válik, az elem pontjai a klotoid függvényeinek és a CCTurn összefüggéseinek ismeretében számolhatóak.

Az osztály kényelmesebb használatának érdekében még két segédfüggvényt implementáltam. Az első a CSC kapcsolódást segíti azzal, hogy az utolsó lépésben δ beállítása helyett elég a második CCTurn középpontját megadni. Ezután a középpont felhasználásával az algoritmussal eldöntöm, hogy melyik kapcsolódás fajta hozható lehet (van irányváltás vagy nincs), valamint kiszámolom a hozzá tartozó deltát.

A második segédfüggvénnyel szintén a kapcsolódás tervezését szerettem volna könnyebbé tenni. Két CCTurn összekapcsolásakor elég az első CCTurn objektumnak megadni a második CCTurn középpontját. Hasonlóan az előzőhöz, az algoritmusom a két CCTurn középpont távolságának függvényében eldönti, hogy melyik kapcsolódási fajta szerkeszthető meg és kiszámolja a hozzá tartó δ értéket. A második CCTurn konstruálásakor pedig már ismert a kezdő- és végkonfiguráció, ami alapján történő tervezését korábban ismertettem.

Az előbbiekben leírt két segédfüggvény lényegesen megkönnyíti a pályatervezés menetét. Ugyanakkor fontos megemlítenem, hogy csak akkor működnek megfelelően, ha a pályatervező algoritmusban számolt pontok ténylegesen az adott paraméterű CCTurn valós pontjai. Ezért a pályatervező algoritmus implementálásakor kiemelt figyelmet kellett fordítanom arra, hogy az összes előforduló lehetőséget megvizsgáljam

és megfelelően kezeljem. A program lehető legkisebb funkcionális egységeinek tesztelésével biztosítottam az algoritmus megfelelő működését.

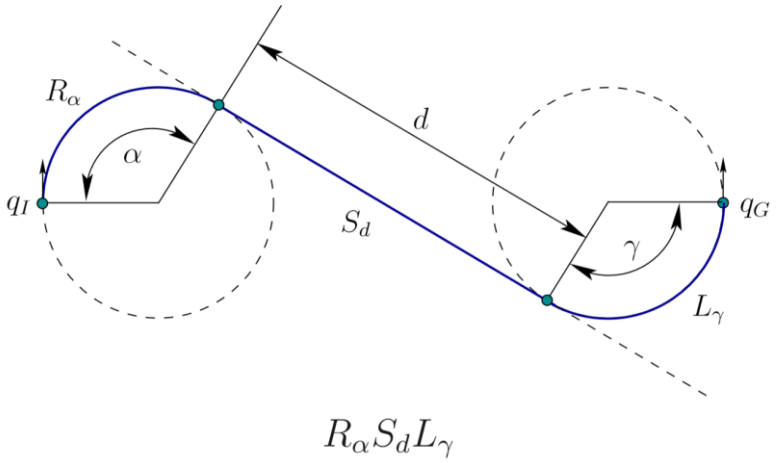
A következő két fejezetben bemutatom a két lokális pályatervező algoritmust, ami CCTurn elemek használatával biztosítja a folytonos görbületű pályát. Részletesen kitérek a pályatervezés menetére, ahol felhasználom az itt leírt CCTurn tulajdonságait különösképpen a folytonos görbületű kapcsolódáshoz szükséges számolásokat. Emiatt tartottam fontosnak, hogy érthetően, szemléletes ábrák segítségével mutassam be a CCTurn elem működését és számítását.

4 CCRS lokális pályatervezés

A dolgozatom egyik feladata, hogy egy a tanszéken fejlesztett C++ könyvtárba implementáljam a CCRS algoritmust [3], mely a jól ismert Reeds-Shepp (továbbiakban RS) lokális pályatervező algoritmus általánosításának tekinthető. Ugyanazokat a pályakombinációkat (4.1. táblázat) tartalmazza mint az RS, csak a körívek helyett CCTurn pályaelemeket használ a folytonos görbületű pályatervezés céljából. A CCRS az RS általánosítása, amiből az RS-t kapjuk vissza, ha a klotoid görbületi változásának gyorsaságát végtelenhez közelítjük. Mielőtt rátérnék a CCRS lokális pályatervezőre, ismertetem röviden az RS algoritmust.

4.1 Reeds-Shepp algoritmus

Anholonom járművek optimális irányítására általános algoritmus nem létezik, azonban pár speciális korlátozás bevezetése után a probléma lényegesen egyszerűbb lesz. Ilyen korlátozásokat tartalmaz a Reeds-Shepp [13] féle jármű modell, ami az autó sebességét végig konstansnak írja elő, csak az előjele, vagyis az iránya változhat. Az autó vagy egyenesen előre megy, vagy balra/jobbra kanyarodik minimális sugarú köríven. (Ez utóbbi feltételből látszik, hogy a pályája nem folytonos görbületű ellentétben a CCRS algoritmussal.)



4.1. ábra. Reeds-Shepp pálya [1]

Reeds és Shepp megmutatta, hogy a róluk elnevezett jármű, tetszőleges start és cél pozíció közötti legrövidebb útvonal előállítható 48 szekvenciából (elemi geometriai

elemekből összerakott pálya), amiket 9 különböző csoportba (4.1. táblázat) sorolhatunk. A 4.1. ábrán egy ilyen pálya látható, amit érdemes görbület szempontjából összehasonlítani a 3.2 ábrával.

Csoport	Szekvenciák
C|C|C	$(L^+R^-L^+)(L^-R^+L^-)(R^+L^-R^+)(R^-L^+R^-)$
CC|C	$(L^+R^+L^-)(L^-R^-L^+)(R^+L^+R^-)(R^-L^-R^+)$
C|CC	$(L^+R^-L^-)(L^-R^+L^+)(R^+L^-R^-)(R^-L^+R^+)$
CSC	$(L^+S^+L^+)(R^+S^+L^+)(L^+S^+R^+)(R^+S^+L^+)$ $(L^-S^-L^-)(R^-S^-L^-)(L^-S^-R^-)(R^-S^-L^-)$
$CC_\beta\|C_\beta C$	$(L^+R_\beta{}^+L^-{}_\beta R^-)(L^-R_\beta{}^-L_\beta{}^+R^+)(R^+L_\beta{}^+R_\beta{}^-L^-)(R^-L_\beta{}^-R_\beta{}^+L^+)$
$C\|C_\beta C_\beta\|C$	$(L^+R_\beta{}^-L^-{}_\beta R^+)(L^-R_\beta{}^+L_\beta{}^+R^-)(R^+L_\beta{}^-R_\beta{}^-L^+)(R^-L_\beta{}^+R_\beta{}^+L^-)$
$C\|C_{\pi/2}SC$	$(L^+R_{\pi/2}{}^-S^-R^-)$ $(L^-R_{\pi/2}{}^+S^+R^+)(R^+L_{\pi/2}{}^-S^-L^-)$ $(R^-L_{\pi/2}{}^+S^+L^+)$ $(R^+R_{\pi/2}{}^-S^-R^-)$ $(R^-R_{\pi/2}{}^+S^+R^+)(L^+L_{\pi/2}{}^-S^-L^-)$ $(L^-L_{\pi/2}{}^+S^+L^+)$
$CSC_{\pi/2}\|C$	$(L^+S^+L_{\pi/2}{}^+R^-)(R^+S^+L_{\pi/2}{}^+R^-)(L^+S^+R_{\pi/2}{}^+L^+)(R^+S^+L_{\pi/2}{}^+R^-)$ $(L^-S^-L_{\pi/2}{}^-R^+)(R^-S^-L_{\pi/2}{}^-R^+)(L^-S^-R_{\pi/2}{}^-L^+)(R^-S^-L_{\pi/2}{}^-R^+)$
$C\|C_{\pi/2}SC_{\pi/2}\|C$	$(L^+R_{\pi/2}{}^-S^-L_{\pi/2}{}^-R^+)(R^+L_{\pi/2}{}^-S^-R_{\pi/2}{}^-L^+)$ $(L^-R_{\pi/2}{}^+S^+L_{\pi/2}{}^+R^-)(R^-L_{\pi/2}{}^+S^+R_{\pi/2}{}^+L^-)$

4.1. táblázat. A Reeds-Shepp jármű optimális pályájának 48 lehetséges szekvenciája

Összefoglalva, az RS egy véges lépésszámú algoritmus, ami minden esetben megtalálja a legoptimálisabb pályát, viszont a görbülete nem folytonos. Véges lépésszáma miatt érdemes az algoritmus folytonos görbületű változatát (CCRS) megvizsgálni és összehasonlítani ez eredeti algoritmussal.

4.2 Pályatípusok implementálása

Ebben a fejezetben bemutatom, hogyan sikerült a lokális CCRS pályatervező algoritmust implementálnom a C++ könyvtárba. Mivel a CCRS az RS tervezőhöz hasonlóan, ugyanazokat a pályatípusokat használja, amelyek szintén kilenc osztályba

sorolhatóak, ezért célszerűnek tartom az osztályokon keresztül bemutatni az implementálást, kezdve a legelemibbtől, haladva a legkomplexebbig.

A tervezés lépései általában a CCTurn középpontok keresést, és kezdő- valamint célkonfigurációk meghatározását jelenti, köszönhetően a CCTurn osztály függvényeinek, amit korábban részletesen leírtam. Egy pontot gyakran úgy kapok meg, hogy ismerem a másik két ponttól való távolságát, és az ismert pontokból a megfelelő sugárral húzott körök kimetszik a keresett pontot. Ez a megoldás kényelmes, mert implementálva volt a C++ könyvtárba, viszont általában két megoldás van, ami közül a megfelelőt kell kiválasztani. A továbbiakban az egyszerűbb jelölés érdekében leírom a két pontot, amik egy vektort jelölnek, és utána azt, hogy bal vagy a jobb félsíkján helyezkedik el a pont, pl.: AB bal.

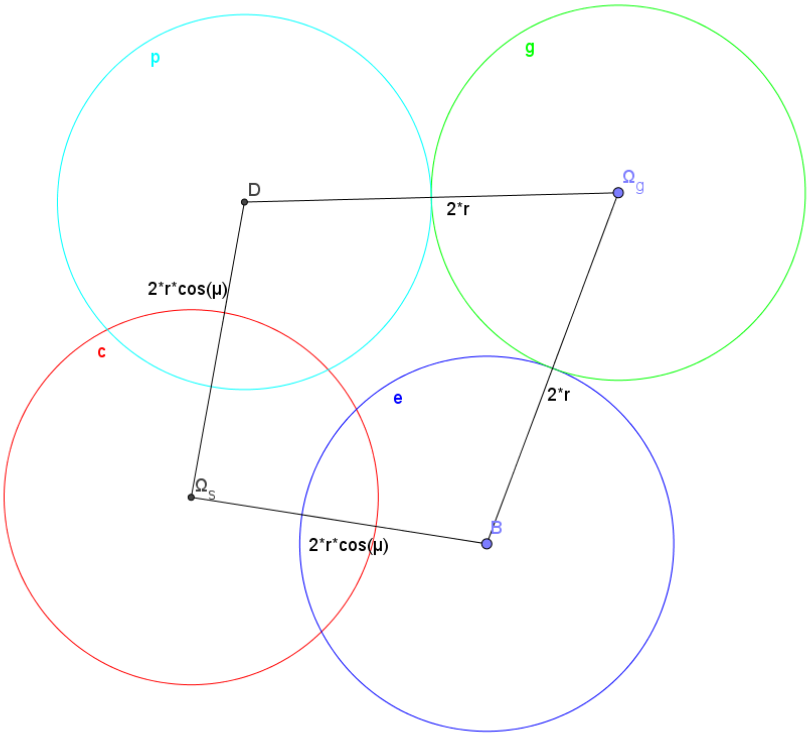
Az ábrákat a GeoGebra [6] szerkesztő programban rajzoltam. A körök a CCTurn-ök külső köreit szemléltetik, tervezés szempontjából ugyanis ezek a lényegesek. A kezdő CCTurn piros színű, az utolsó zöld, köztük lévőek pedig kékek. A fekete színű CCTurn-ök segéd elemek, nem a pálya részei, csak a tervezést segítik. A CCTurn körök helyett gyakran csak köröket fogok írni ebben a fejezetben, ami alatt a CCTurn-ök külső körét értem. Emlékeztetőül, ha a köröknél az átmenetnél nincs haladási irány változás, akkor érintik egymás és a távolságuk $2r$, ellenkező esetben metszik egymást és a távolságuk $2r\cos(\mu)$. Ezek ismeretében az ábrák jól érthetőek, csak kevés kiegészítő magyarázatot igényelnek.

4.2.1 CSC osztály

Ezt a tervezést már kifejtettem a CCTurn kapcsolódásainál, de fontosnak tartom itt is megjegyezni, mert ez is egy osztály, ami nyolc darab lehetséges pályát tartalmaz, amik a következőek: RSR, LSL, RSL, LSR, ahol a jelölések értelemszerűen; L= balra kanyarodó CCTurn, R= jobbra kanyarodó CCTurn, S pedig az egyenes szakaszt jelenti. A későbbiekben „|”-lal fogom jelölni, ha irányváltás történik a pályaelem határánál. Ebben az osztályban ilyen lehetőség nincs, tehát a négy elem kétféleképpen szerepel, előremenetben és hátramenetben. Megjegyzem, hogy bár a pályahatároknál nincs haladási irányváltás, de magán a CCTurn elemen belül lehet, ha 180°-nál nagyobb az hajlásszöge (lásd nagy hajlásszögű CCTurn fejezet).

4.2.2 CCC osztályok

Ez tulajdonképpen három osztályt jelent; C|CC, CC|C, C|C|C, de felépítésüket és tervezésüket tekintve nagyon hasonlóak, ezért egyszerre mutatom be őket. Mind a háromnál adott a start és cél kör, a köztük lévőnek pedig ismerjük a távolságát a másik két kör középpontjához képest. A két metszéspont közül bármelyik lehet a második kör középpontja, azt választom, amelyik a rövidebb pályát eredményez.

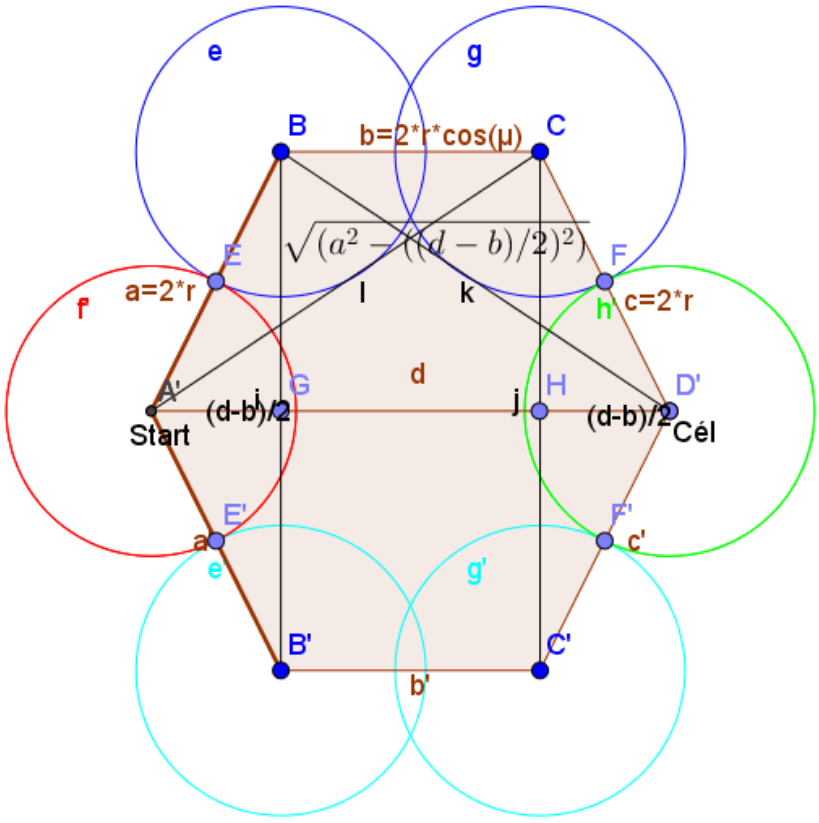


4.2. ábra. C|CC pálya

4.2.3 CC|CC osztály

A két középső körnek, a CCTurn tulajdonságaiból következően, számolhatóak a középpontjaik (4.3. ábra). A körök középpontjai egy húrtrapézon helyezkednek el szimmetriai okok miatt (az osztály definíciója szerint mindkettőnek ugyanakkora a hajlásszöge), ezt a pontok számolása során ki is használtam.

Az ábrán látszik, hogy két megoldás van általában (kivétel, ha a húrtrapéz egy szakasszá torzul), ezek közül a rövidebbet választom. Fontos megemlítenem, hogy a két megoldás értelemszerűen két-két CCTurn-t tartalmaz, de a párok közti csere nem lehetséges. Pl.: ha a sötétebb kék pályánál, az *e* kör helyett az *e'*-t vesszük, akkor az nem is folytonos görbe.



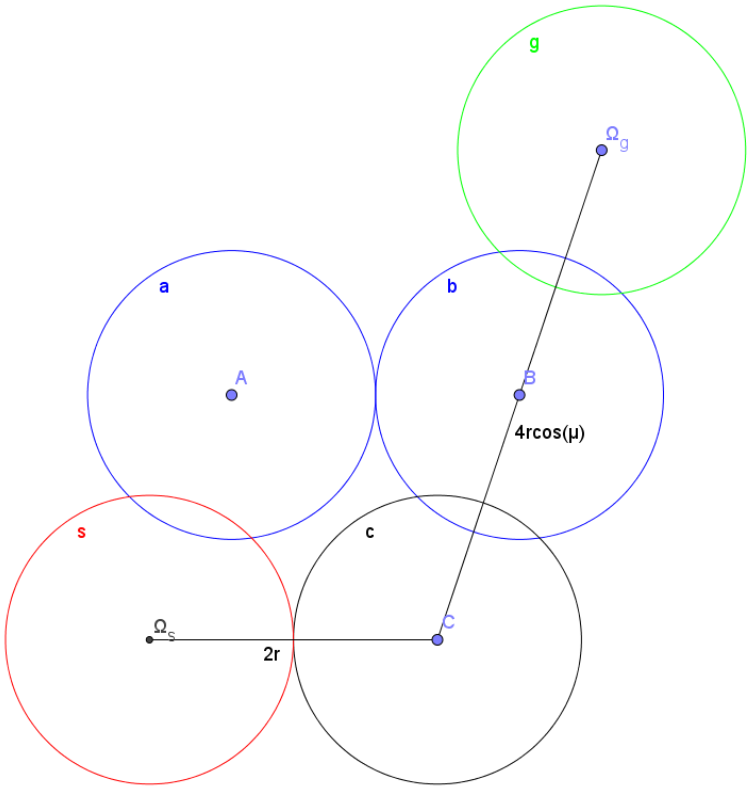
$$k = l = \sqrt{(a^2 - ((d-b)/2)^2 + (b + (d-b)/2)^2}$$
$$k = l = \sqrt{(a^2 - ((d-b)/2)^2 + ((d+b)/2)^2}$$
$$k = l = \sqrt{(a^2 + 4db}$$
$$k = l = \sqrt{(4r^2 + 8rcos(\mu)d}$$

4.3. ábra. CC|CC pálya

4.2.4 C|CC|C osztály

A C|CC|C pályatervezéshez szükség van egy segédkörre, aminek távolsága ismert a start és a cél körtől (4.4. ábra). A fiktív pályaelem indoklása a következő. $\Omega_s AB\Omega_g$ pálya helyett először egy olyan pályát tervezek, ahol Ω_g helyett C az utolsó CCTurn középpontja. C kör a starthoz haladási irányváltás nélkül csatlakozik, b-hez irányt váltva. Kihasználva, hogy az osztály definíciója szerinti a két középső CCTurn hajlásszöge ugyanakkora, a négy pont egy rombusz csúcsait alkotják, Ω_g pedig B szerinti tükörképe C-nek. Ezek után a tervezés menete triviális, C pontot ismerve, B előáll $C\Omega_g$ szakaszfelező pontjaként. A pont pedig megkapható Ω_s és B pont függvényében a CCC osztályoknál ismertetett eljárások segítségével.

Nem tértem ki rá (és az ábrára se rajzoltam rá, hogy átlátható maradjon), de itt is a metszéspontokból következően több megoldás is adódik. Ezek megkülönböztetésére különös figyelmet fordítottam az implementáció során.



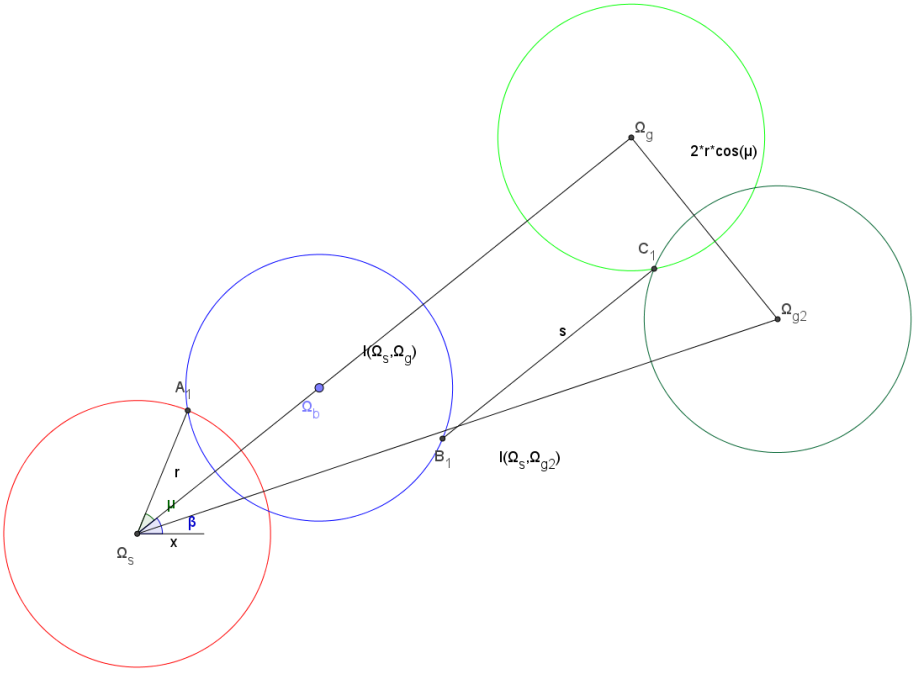
4.4. ábra. C|CC|C pálya

4.2.5 C|CSC osztály

Az osztály egy olyan megkötést tartalmaz, hogy a második CCTurn hajlásszöge 90°. Ennek következtében a második kör középpontja rajta van start és a cél körök középpontját összekötő egyenesen, abban az esetben, ha az utolsó kör iránya megegyezik a másodikéval, és távolsága az elsőtől ismert, tehát helyzete egyértelműen adott. Ez után második és a harmadik kör között CSC összekapcsolást kell létesítenünk, és megkaptuk a keresett pályát.

Abban az esetben, ha a harmadik körnél irányváltás van (4.5. ábrán a sötétebb zöld pálya), visszavezetem a problémát az előző feladatra, vagyis megkeresem a CSC összekapcsolásnál látott módon Ω_g pontot. Ω_s Ω_g Ω_{g2} derékszögű háromszög, ismert két pontja és két oldala, tehát a harmadik pont számolható Püthagorasz tétellel.

A pálya létezésének feltétele, hogy a start és a cél pontok távolsága elég nagy legyen, ami számolható az összekapcsolások tulajdonságaiból, de ennek az ismertetésétől most eltekintek. Fontos viszont, hogy a második esetben, fiktív Ω_g keresésekor, a metszéspontok közül a megfelelőt vegyem (Ω_s Ω_g bal).



4.5. ábra. C|CSC pálya

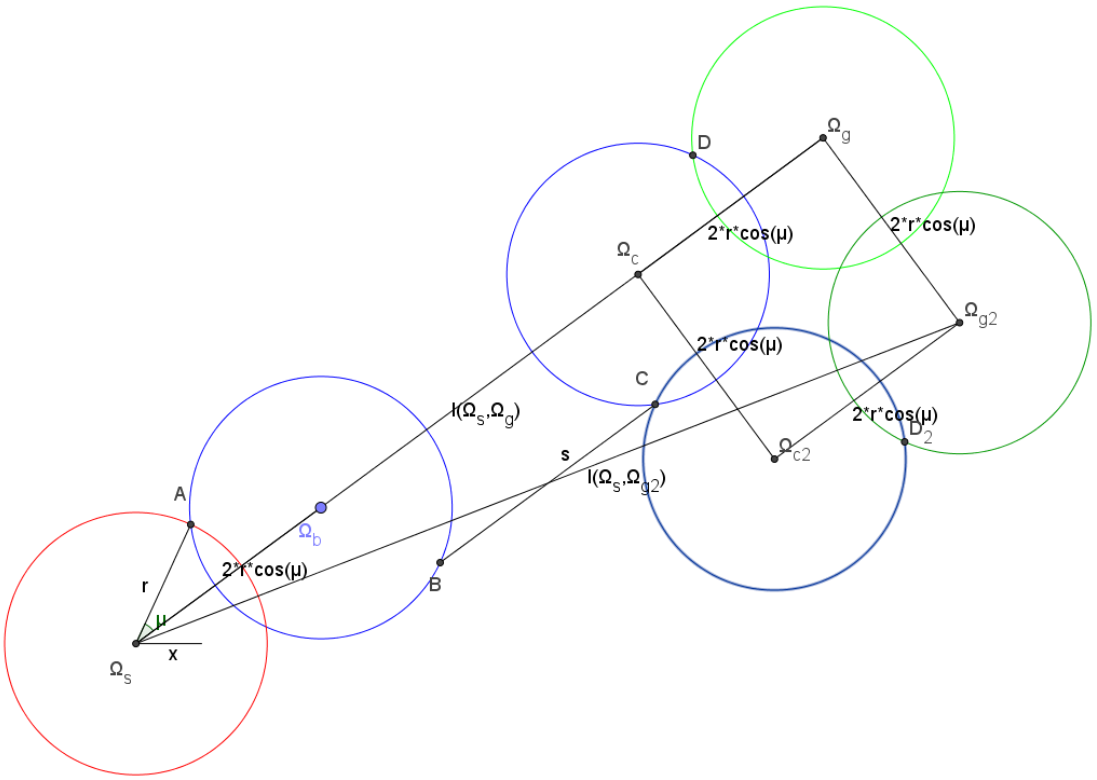
4.2.6 CSC|C osztály

Ez az osztály a C|CSC megfordításából adódik, vagyis keresésekor elég a start- és a célkonfigurációt felcserélni. A konfiguráció cserélés következtében a haladási

irányok is felcserélődnek, ezért fontos, hogy a pálya pontjainak a sorrendjét is meg kell cserélni.

4.2.7 C|CSC|C osztály

Ez az osztály hasonló a C|CSC-hez csak az utolsó CCTurn előtt van még egy 90°-os hajlásszögű CCTurn (4.6. ábra). Tervezése is hasonlós az említett osztályhoz, szintén két csoportra lehet bontani aszerint, hogy az utolsó körnél van irányváltás vagy sem. Irányváltás mentes esetben, a körök középpontjai egy egyenesen helyezkednek el, köszönhetően annak, hogy a második és a harmadik hajlásszöge 90°. A másik eset pedig a korábbiakhoz hasonlóan visszavezethető az előbbire (Ω_s Ω_g Ω_{g2} derékszögű és ismert két oldala).



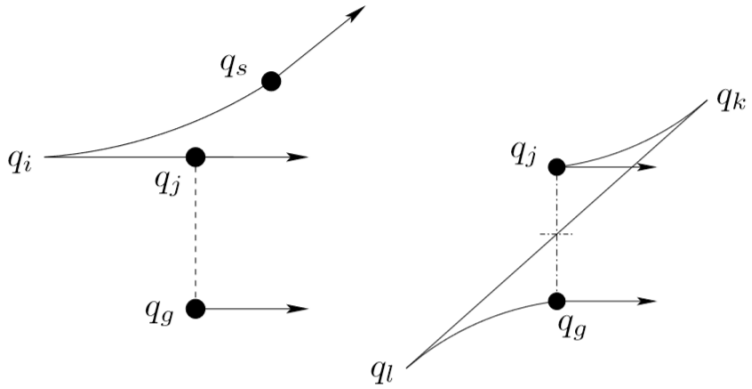
4.6. ábra. C|CSC|C pálya

4.2.8 Topológiai pálya

Ahhoz, hogy egy lokális pályatervező algoritmussal közelíteni lehessen egy globális pályát, szükséges hogy biztosítsa az úgynevezett topológiai feltételt. A későbbiekben az approximációs résznél részletesen bemutatom a közelítéses pályatervezést, előnyeivel és hátrányaival, valamint kitérek a topológiai feltétel pontos jelentésére, most viszont csak a megvalósított pályatervezést írom le.

A pálya két részből áll, az első felében a mobil robot beáll a célkonfiguráció irányába, a második felében pedig a 4.7. ábrán látható módon a két párhuzamos konfiguráció közti görbét tervezi. Az első, reorientációs rész két hasonló paraméterű klotoiddal kezdődik (a két szélén nulla, a kapcsolódásnál pedig a maximálisnál kisebb görbülettel) és végül egy egyenes szakasszal záródik. A második, párhuzamos szakasz hasonló paraméterű klotoidokkal kezdődik és záródik, köztük egy egyenes található.

A két résznél a klotoidok paramétere eltérő, számolásuk viszonylag komplex. A levezetésük részletesen megtalálható az [3] irodalomban, a topológia feltétel bizonyításával egyetemben. A dolgozatomban ennek ismertetésétől eltekintek, viszont megjegyezném, hogy az implementálás ennek függvényében is tartalmazott kihívásokat, ugyanis a leírás csak egy speciális esetet mutatott be, az általánosat nekem kellett kitalálni.



4.7. ábra. Topológia pályatervezés [3]

5 T*TS lokális pályatervezés

A T*TS lokális pályatervezést Kiss Domokos dolgozta ki [7]. Ez az algoritmus szintén folytonos görbületű pályatervezést valósít meg, mint a CCRS, és ahhoz hasonlóan ez is egy korábbi algoritmusból (C*CS, [7]) fejlődött ki.

A pályatervező nevében két olyan jelölés is van, amivel az Olvasó idáig nem találkozhatott a dolgozatomban, először ezek feloldásával kezdem az algoritmus bemutatását. Korábban a körívre és a CCTurn-re is a „C” jelölést használtam, és a tárgyalt pályatervezőtől függött, hogy melyiket értettem alatta. Az egyértelmű jelölés érdekében került bevezetésre a „T” mint CCTurn elem, és a „C” maradt a körív azonosítására.

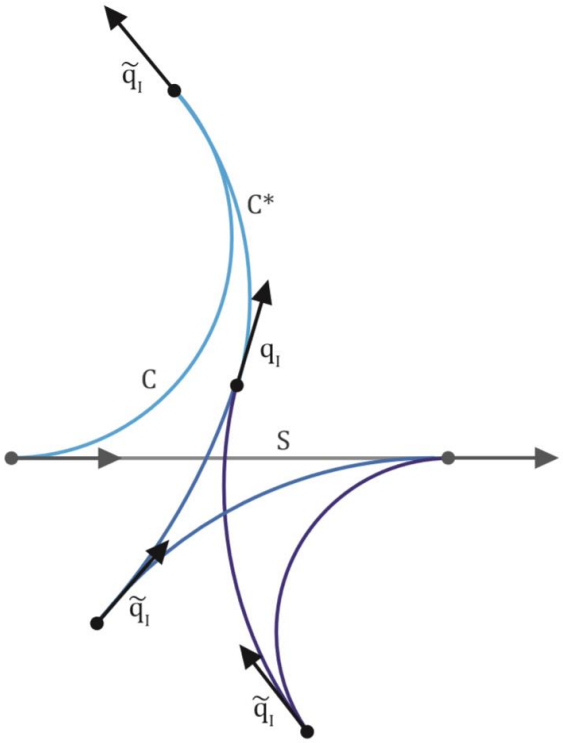
A körív és CCTurn pályatípusok görbületének nagyságára nem lehet következtetni a jelölésükből (legalábbis az általam használtakból). A kinematikai korlátozásoknál a görbületnek csak felső korlátja van, alsó nincs, ezért az tetszőlegesen kicsi lehet, akár nulla is, ami tulajdonképpen az egyenes. Ezt jelöli a „*”, hogy a körív vagy CCTurn helyett az egyenes is megengedett. Összefoglalva, a T*TS két pályatípust rövidít, a TTS-t és az STS-t.

Az algoritmus tervezésekor a szerző kiemelt célja volt, hogy az emberi vezetéshez hasonló pályákat kapjon. Ezt a következő tulajdonságokkal próbálja biztosítani. A célkonfigurációba érkezés utolsó szakasza az embereknél legtöbbször egy egyenesen történik, nem pedig köríven, ezért a tervező utolsó eleme mindig egy egyenes szakasz.

A másik tulajdonság abból következik, hogy sokkal kényelmesebb kisebb és lassabb kormánymozdulatokkal vezetni, ezért ellenben a CCRS pályatervezővel, ahol a CCTurn-beli kanyarívek görbülete mindig maximális, a T*TS nem követeli meg a köríveknél a maximális görbület használatát, tetszőleges kisebb érték lehet helyette. Ennek a szigorításnak a feloldásával, látszólag még messzebb kerülünk a pályahossz szerinti optimális megoldástól, mint CCRS, de a végső célunk nem az önmagában történő lokális pályatervezés, hanem a felhasználásából kapott globális pályatervezés.

5.1 Pályatervezés lépései

A maximálisnál tetszőlegesen kisebb görbület megengedésének köszönhetően, több megoldás is adódik ugyanannál a start és célkonfigurációnál. Erre látható példa a következő ábrán. A megoldások egy osztályt alkotnak, amik közül a mobil robot számára egy alkalmasat kell választani. A későbbiekben ez jelentős előnnyel fog járni a CCRS-sel szemben, amikor a globális pályatervező közelítésekor használjuk az algoritmust, ugyanis több lehetőségnél nagyobb az esély arra, hogy van legalább egy ütközésmentesen bejárható pálya.



5.1. ábra. CCS pálya lehetőségek azonos start- célkonfigurációnál [1]

Az 5.1. ábrán látható, hogy a q_i konfigurációból többféle lehetőségünk van, hogy az S egyenesre jussunk. Az ábra köríveket használ (CCS pályatervezés) CCTurn-ök helyett, de értelemszerűen az utóbbiakkal tervezve, hasonlóan több megoldáshoz jutunk.

Tervezéskor az első lépés, hogy a startkonfigurációból többféle CCTurn-t indítunk tetszőleges paraméterekkel, amelyek kisebbek, mint a kinematikai korlátozásokból adódó maximumok. Előre meghatározott felbontással, egyenletesen mintavételezem a következő paramétereket:

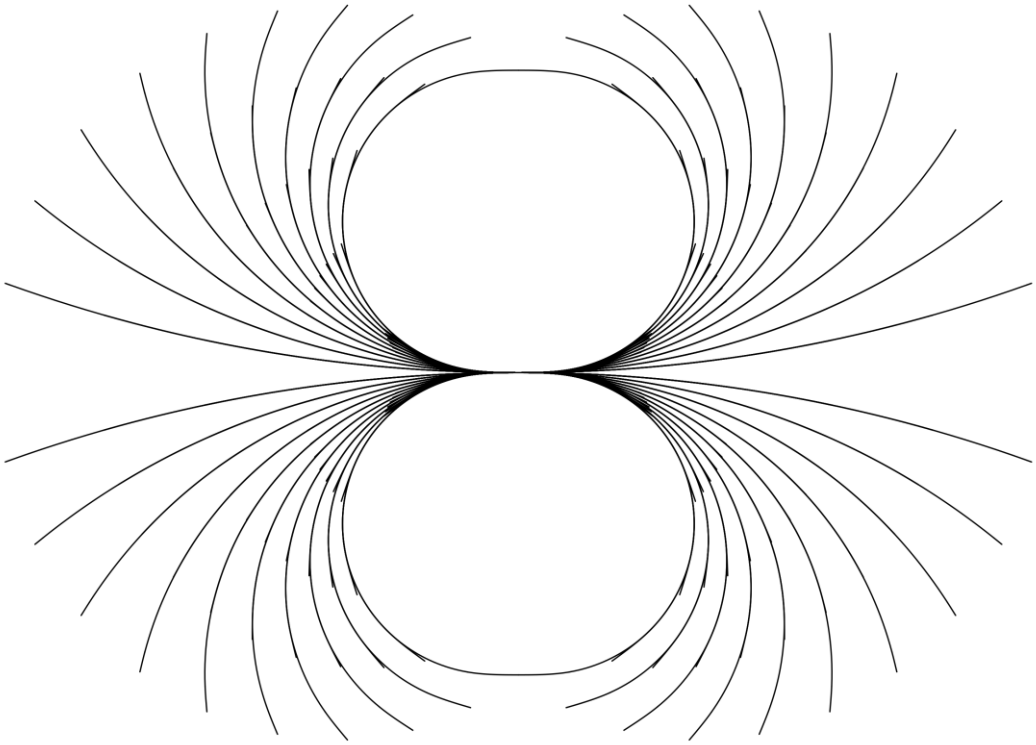
$$l \in [0, l_{max}], \qquad l_{max} = \frac{\pi}{\kappa_{max}} + \frac{\kappa_{max}}{\sigma_{max}} \tag{5.1}$$

$$\kappa \in [0, \kappa_{max}] \tag{5.2}$$

Ahol l_{max} maximális görbületű 90°-os hajlásszögű CCTurn hosszát jelenti. Bevezetése azért volt célszerű, hogy a végtelen sok lehetőség közül, a hossz szerint is megadjunk egy felső határt. A CCTurn-t a programomban azonban a hossza szerint nem lehet konstruálni, ezért át kellett térnem a hajlásszög szerinti paraméterezésre, ami figyelembe véve az előbbi hosszúságra előírt korlátozást, a következő tartományra adódott:

$$\delta \in [0, \delta_{max}], \qquad \delta_{max} = \kappa \cdot \left[\frac{\pi}{\kappa_{max}} + \frac{\kappa_{max} - \kappa}{\sigma_{max}}\right] \tag{5.3}$$

A kétszeresen egymásba ágyazott iteráció során először a görbület értékét választottam ki, majd annak függvényében a hajlásszöget a lehetséges tartományból. Azonos paraméterekkel megterveztem mind a négyféle CCTurn-t. A következő ábrán erre látható példa, görbület szerint tíz, hajlásszög szerint pedig öt mintát véve.



5.2. ábra. T*TS mintavételezés

Az eredeti pályatervező algoritmusban ennél a mintaszámnál egy nagyságrenddel többet használok, ami a futási időben jelentős lassuláshoz vezet, de nagyobb valószínűséggel kapok ütközésmentesen bejárható pályát, ami miatt összességében megéri többet várni. Az algoritmus optimalizálásaképpen, már az első elemnél megvizsgálom, hogy volt-e ütközés, és ha igen, akkor annál a CCTurn típusnál az adott görbületnél már nem érdemes nagyobb hajlásszögre tervezni, így az iterációban ugorva, a minták számát csökkenteni tudom.

A második CCTurn-nél már csak egy megoldást kaphatunk, létezésének szükséges feltétele, hogy a célkonfiguráción átmenő egyenes, az első elem utolsó konfigurációjából indított maximális görbületű CCTurn célkonfigurációjának a megfelelő oldalán legyen. Az elem startkonfigurációja és a célkonfigurációjának az orientációja adott, tehát ismert a CCTurn hajlásszöge, a tervezéshez csak a görbület nagysága hiányzik, amit bináris keressél általában pár lépés alatt megkapok.

Az első és a második CCTurn ismeretében az utolsó szakasz könnyen számolható. Utolsó lépésként az ütközésmentesen bejárható pályák közül kiválasztom a legrövidebbet.

5.2 Topológia pálya

Az eddig ismertetett pályatervezés topológia értelemben nem konvergens, ennek hiányában pedig nem alkalmas approximációs pályatervezésre. A CCRS-hez hasonlóan a T*TS algoritmusnál is plusz pályatervező biztosítja ezt a tulajdonságot a szükséges esetekben, az $e\bar{e}s$ pályatervező. A „*e*” tulajdonképpen a kis hajlásszögű CCTurn-t jelöli, amikor a körív hossza nulla. A második elemnél a komplementer arra utal, hogy paramétereiben megegyezik az elsővel, csak az iránya ellentétes.

Az első két elem görbülete azonos, csak irányuk ellentétes. Ebből a megkötésből adódik, hogy a T*TS-sel ellentétben mindig pontosan egy megoldást kapunk. [7] irodalomban megtalálható a bizonyítása, hogy az ilyen típusú pálya rendelkezik a topológia tulajdonsággal, illetve a paraméterek számításához szüksége egyenleteket is tartalmazza, aminek közlésétől most is eltekintek. Megjegyzem, hogy a klotoidok használata miatt, hasonlóan a CCRS lokális pályatervezőjéhez, az egyenletek Fresnel integrálokat tartalmaznak, aminek inverz megoldása számítás igényes feladat.

Összefoglalva, a T*TS lokális pályatervező algoritmus, az emberi vezetéshez hasonló pályákat tervez, és rendelkezik a topológia tulajdonsággal, ami alkalmassá teszi, hogy approximációs pályatervezésre használjuk. A későbbiekben összehasonlítom a CCRS algoritmussal, de előtte bemutatom a globális pályatervező algoritmust, illetve az approximációs módszert, amelynek segítségével össze lehet kapcsolni a különböző globális és lokális tervezőket.

6 Globális pályatervezés

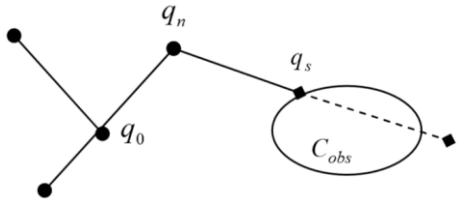
A globális pályatervező algoritmusok alkalmasak arra, hogy az akadályokat tartalmazó környezetben ütközés mentes pályát találjanak. Ebben a fejezetben bemutatom a tanszéki C++ könyvtárban lévő globális tervező algoritmusokat, amiket a teszt során alkalmaztam.

Először a széles körben használt RRT-t, majd ennek a módosított változatát a RTR-t [9] ismertetem. Ezek után kitérek arra, hogy az általam implementált lokális pályatervező algoritmusokat, hogyan lehet összekapcsolni a globális tervezőkkel, hogy végeredményben egy olyan algoritmust kapjak, ami alkalmas a lokális és globális korlátozások betartására. Legvégül az ütközés detektálásról írok, ami az egyik legtöbb számítást igénylő része a globális pályatervezésnek.

6.1 RRT algoritmus

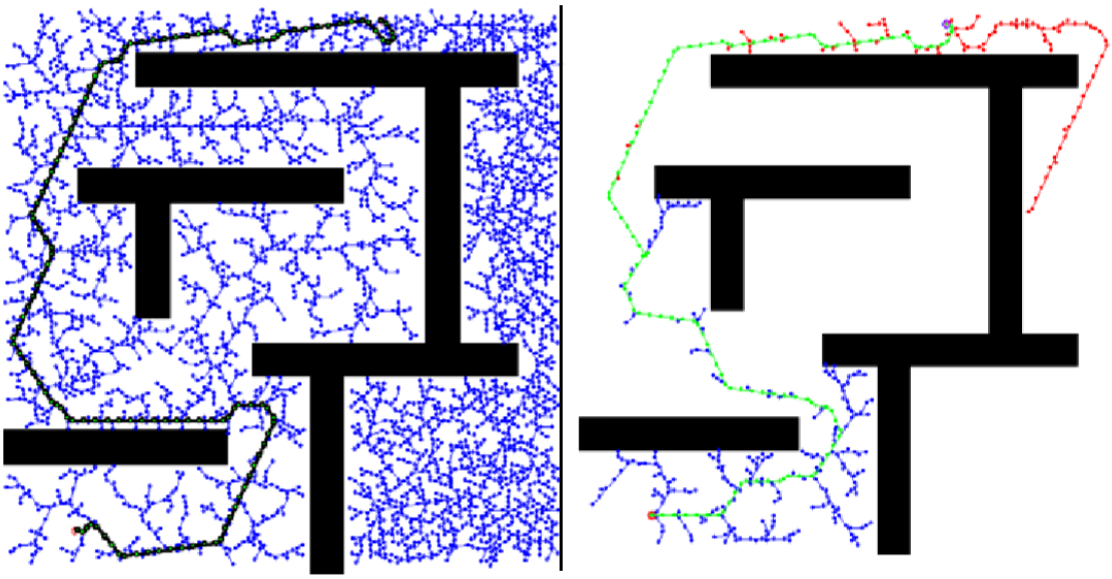
A RRT algoritmus (jelentése: Rapidly Exploring Random Trees) véletlen mintavételezve a konfigurációs térből, egy fát épít, melynek csúcsaiban szabad konfigurációs pontok vannak [10]. A cél, hogy a kezdeti konfigurációból, úgy növesszük a fát, hogy az elérje a célkonfigurációt, ami után a fában egy start és cél közti útkereséssel meghatározzuk a pályát a konfigurációs térben. A gyakorlatban diszkrét felosztást használunk, aminek következtében megengedhető, hogy fában lévő cél pont és a tényleges között kis eltérés legyen.

Az első lépés az algoritmusban, hogy véletlenül kiválasztunk egy pontot és megkeressük, hogy a fában melyik konfigurációhoz van a legközelebb. Ezután megpróbáljuk összekötni a kiválasztott ponttal. Gyakran előfordul, hogy az összekötés során akadályba ütközünk, ilyen esetben csak az akadályig lévő pontot építjük a fához. Ezt szemlélteti a következő ábra:



6.1. ábra. RRT pont beszúrás ütközés esetén [11]

Bizonyos szituációkban célszerű lehet több fát növeszteni. A leggyakoribb implementációknál általában két fát szoktak használni, egyet a startból és egyet a célkonfigurációból. A 6.2. ábrán ennek az előnyei láthatóak, egy fa esetén 4791, két fa esetén pedig 542 csomópontra van szüksége az algoritmusnak, hogy utat találjon a két konfiguráció között.



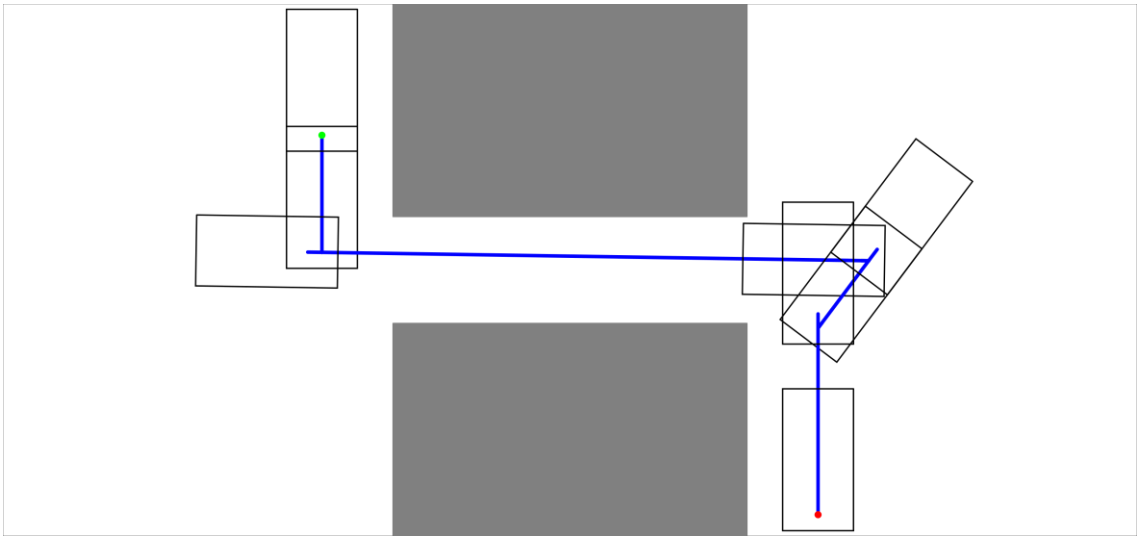
6.2. ábra. RRT útvonal keresés egy illetve két fa esetén [11]

A konfigurációk közti interpoláció miatt a RRT alapvetően csak omnidirekcionális robotok esetén használható, azonban ha a csomópontok összekötésénél a lokális tervezőt használjuk, akkor alkalmassá válik, hogy autószerű robotok számára is megfelelő pályát találjon.

6.2 RTR algoritmus

Az RTR pályatervező az RRT algoritmuson alapszik [9], az implementációjának részletes leírása megtalálható a [1] dolgozatban. Az RTR algoritmus omnidirekcionális robotok számára készült, egy konfigurációból a következőbe való eljutást három mozgással valósítja meg: célkonfigurációba való fordulással, az adott pontig egyenesen történő haladással, majd végül a célirányba történő fordulással (rotate, translate, rotate).

Lényeges különbség az RRT-vel szemben, hogy a mintavételezés nem a konfigurációs térben történik, hanem a sík pontjai közül választ. Összességében szűk folyósokat tartalmazó pályákon a RTR jobb eredményt ad, ezt a későbbiekben tesztekkel is alátámasztom.



6.3. ábra. RTR pályatervezés

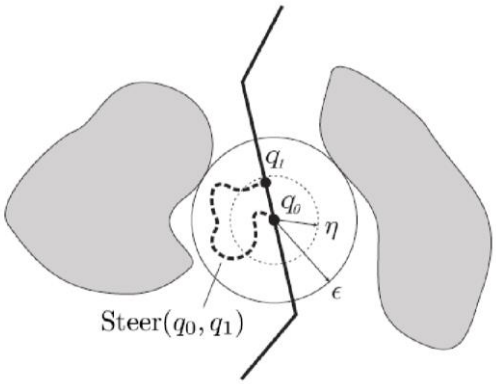
6.3 Approximációs pályatervezés

Approximációs pályatervezés során a globális algoritmussal kapott pályát közelítjük a lokális pályatervezővel. Erre akkor van szükség, a globális pályatervezésnél nem vettük figyelembe a mobil robot kinematikai korlátozásait (pl.: RTR vagy az interpolációs RRT). Az eljárás során a startkonfigurációból próbálunk lokális pályát tervezni a célkonfigurációba. Ha ez nem sikerül ütközésmentesen, akkor a cél helyett eggyel korábbi konfigurációt választunk a globális pályatervezőtől kapott konfigurációs listából és addig ismételjük folyamatot, amíg megfelelő eredményét nem kapunk. Sikeres lokális tervezés után, a pálya fenn maradó részére rekurzívan ismételjük az algoritmust.

Előfordulhat, hogy a listában két szomszédos konfiguráció között a lokális pályatervező nem talál ütközésmentes pályát. Ilyenkor a két konfiguráció közé az interpolációjukkal kapott köztes pontot kell beszúrni. Ha a lokális algoritmus teljesíti a topológia feltételt, akkor az algoritmus teljes és minden globális pályára talál megoldást.

6.3.1 Topológiai feltétel

A topológia feltétel szemléletesen azt jelenti, hogy a pályatervező két konfiguráció közti pályánál mindig talál rövidebbet, ha konfigurációkat eléggé közelítjük egymáshoz. A pontos matematikai feltétel a következő [7]:



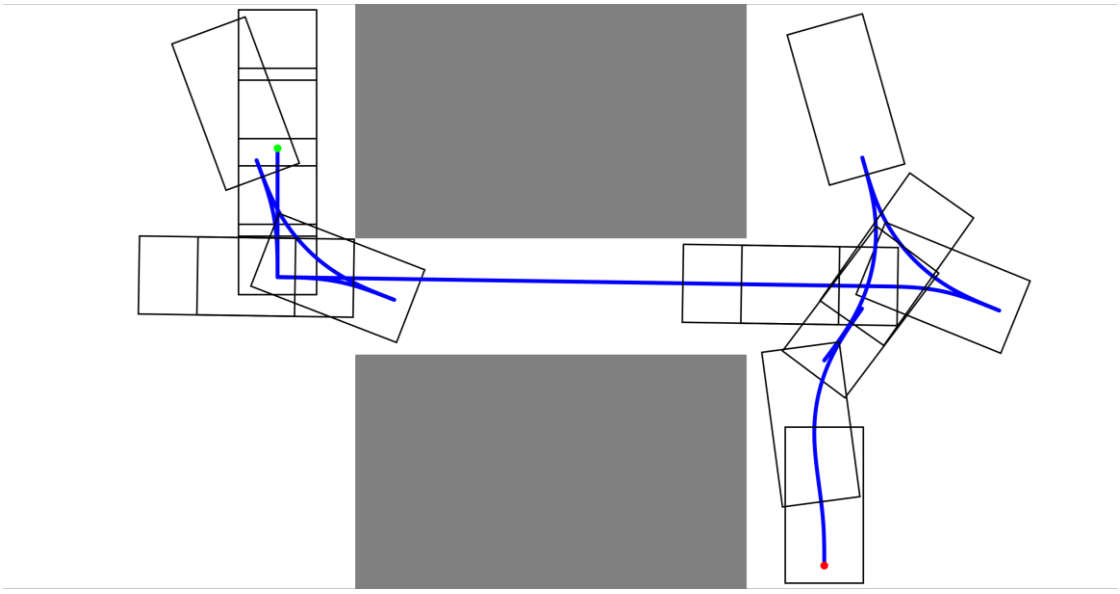
6.4. ábra. Topológiai feltétel [7]

$$\forall \varepsilon > 0, \ \ \exists \eta > 0, \ \ \forall q_0, q_1 \in \mathcal{C} \tag{6.1}$$

$$d_{\mathcal{C}}(q_0, q_1) < \eta \Longrightarrow d_{\mathcal{C}}\left(q_0, Steer(q_0, q_1)(\sigma)\right) < \varepsilon, \tag{6.2}$$

$$\forall \sigma \in [q, S], \tag{6.3}$$

ahol Steer(q_1, q_2) egy lokális pályatervező függvény q_1 és q_2 konfiguráció között, és d_c egy metrika C konfigurációs tér felett.



6.5. ábra. Approximációs pályatervezés

A 6.3. ábrán lévő RTR globális pálya T*TS lokális algoritmussal történő közelítése látható 6.5. ábrán. A pálya ütközésmentesen bejárható és teljesíti a kinematikai korlátozásokat is.

6.4 Ütközés detektálás

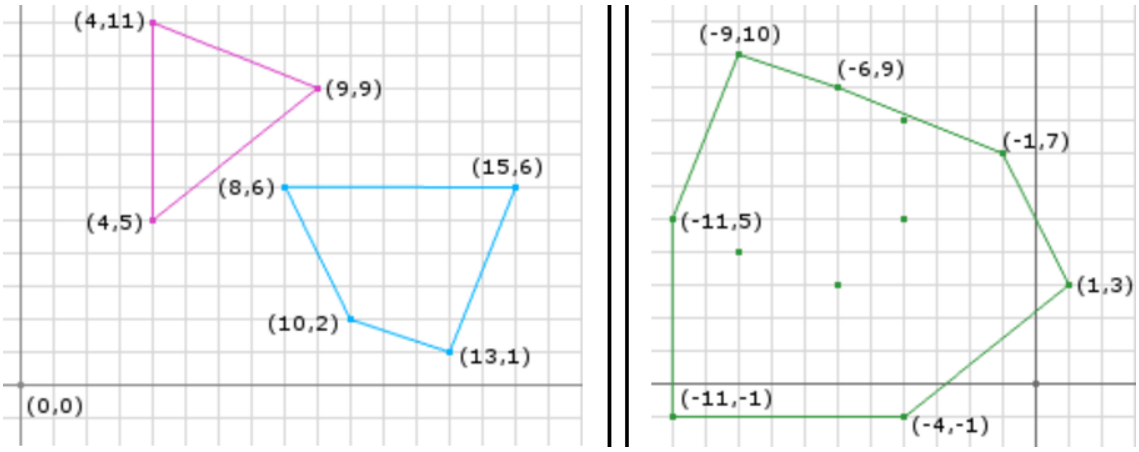
A globális pályatervező algoritmusok egyik legtöbb számítást igénylő része az ütközés detektálás, ezért az algoritmus optimalizálást itt érdemes kezdeni. A C++ könyvtárban lévő ütközés detektálás a következőképpen működött két poligonra. Először megvizsgálta, hogy az egyik poligon tartalmazza-e a másik csúcsait. Ha igen akkor ütköztek, egyébként mindkét poligon összes oldalát páronként kell megvizsgálni, hogy van metszéspontjuk vagy nincs. Előbbi esetben ütköznek, utóbbiban pedig nem. Ezt a megoldást a már implementált metszéspont számoló függvények miatt volt kézenfekvő használni, azonban a Minkowski ütközés detektáló algoritmus sokkal gyorsabb futási időt eredményez, ezért szerettem volna rá áttérni.

Az algoritmushoz [12] két fogalmat kell bevezetnem, a Minkowski összeget és különbséget:

$$A \oplus B = \{a + b | a \in A, b \in B\} \tag{6.4}$$

$$A \ominus B = \{a - b | a \in A, b \in B\} \tag{6.5}$$

Ez poligonoknál azt jelenti, hogy A és B-beli pontokat páronként vektoriálisan össze kell adni vagy kivonni. Kivonásra példa a következő ábra:



6.6. ábra. Minkowski különbség [12]

Utolsó lépésben a különbségként kapott poligonról kell eldönteni, hogy tartalmazza az origót vagy sem, ha igen, akkor ütköznek egyébként nem. Áttérés után, átlagosan a pályatervező algoritmusok 30%-kal lettek gyorsabbak, ami jó eredménynek számít.

7 Pályatervezők összehasonlítása

Ebben a fejezetben összehasonlítom a különböző lokális és globális pályatervezők összekapcsolásából létrejött algoritmusokat. Konzulensem tanácsára több olyan mérőszámot is bevezettem, ami jól jellemzi a kapott pályák minőségét és az algoritmusok performanciáját. A tesztekből egy publikáció is készült [13].

Globális pályatervezők közül a bemutatottakat használtam az RTR-t és RRT-t, utóbbi esetében nem csak az approximációs módszerrel kapcsoltam össze a lokális tervezővel, hanem közvetlenül is. Az elvárásom a CCRS és T*TS lokális pályatervezővel kapcsolatban, hogy az előbbinek kisebb számítási igénye lesz, de minőségileg rosszabb pályát talál, mint a T*TS.

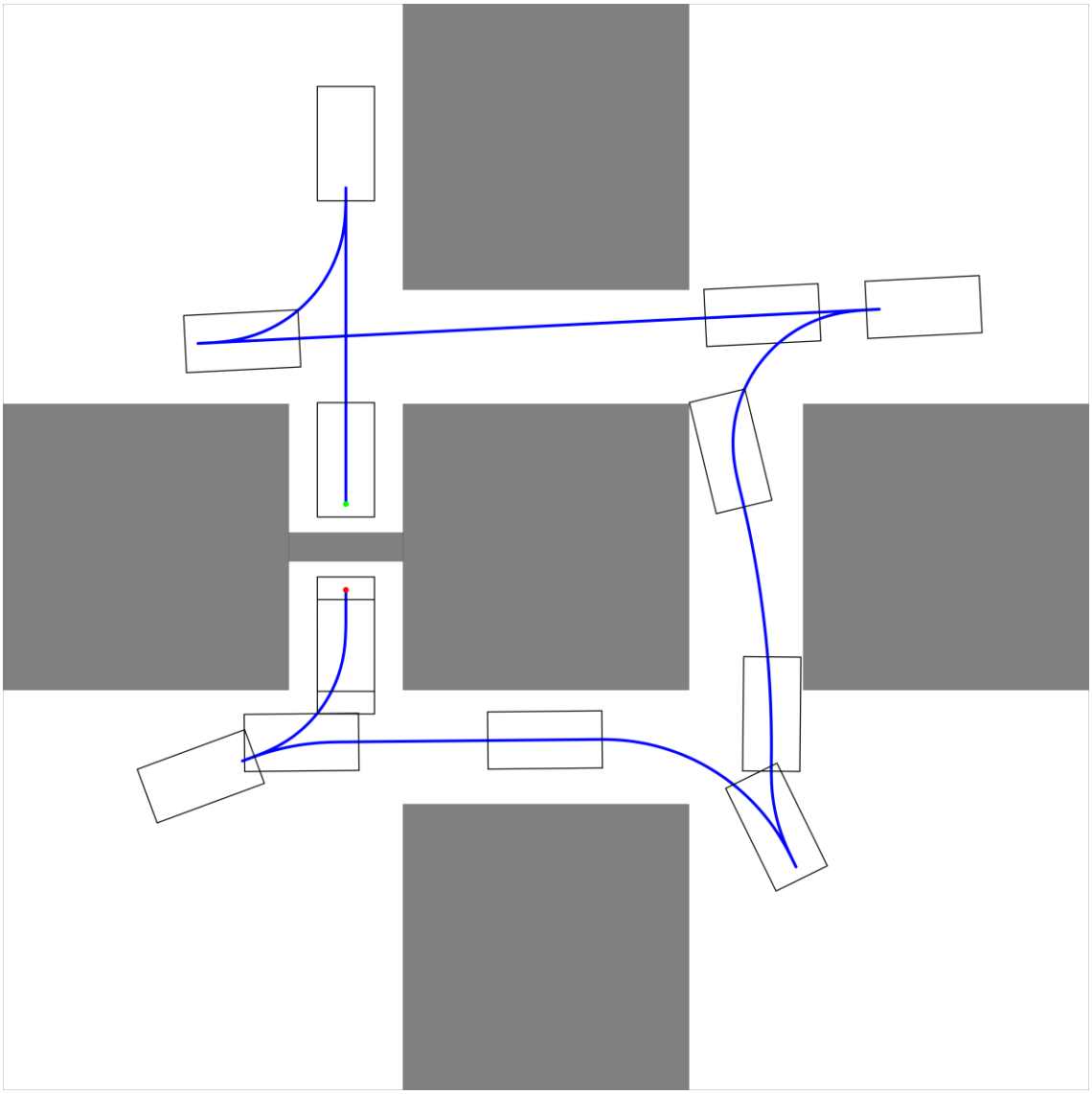
Talán az egyik legfontosabb kérdés az algoritmusoknál, hogy milyen gyakran talál pályát. Az eredményekből jól látszik, hogy a T*TS mindig sikeresebb, mint a CCRS, köszönhetően annak, hogy sokkal több pályamintából tud választani. Ezek után a futási időt nem önmagában, hanem a sikerességi ráta függvényében érdemes összehasonlítani. A pályáktól függően (a függelékben megtalálhatóak az összehasonlítás során használt pályák) az eredmények eléggé változatosak, de kijelenthetem, hogy az előzetes elvárásaimmal ellentétben a CCRS ebben a tekintetben sem tudta megverni a T*TS-t.

A pályák minőségét három mérőszám alapján hasonlítottam össze. Az első a fordulások száma, ami nyilván egy fontos elem a komfort és a bejárási idő szempontjából is. A második az autó fordulásainak abszolút összege radiánban, a harmadik pedig az utazási idő, amit úgy kapok, hogy egyenesben a robot maximális 5 m/s sebességgel megy, kanyarban pedig a görbülettől függően lecsökkenhet akár 1 m/s-ra is. Az irányváltásokat fél másodperccel büntetem.

A pályaminőségi mérőszámokban a T*TS kiemelkedően jól teljesít a CCRS-sel szemben, köszönhetően annak, hogy a maximális görbületeknél kisebbet is megenged kanyarodáskor.

Összességében kijelenthetem, hogy a legjobb párost, az approximációsan összekapcsolt RTR+T*TS adja. (Approximáció során az algoritmus egy bizonyos

lépésszám után leáll, ezért nincs mindenhol 100%-os eredmény.) A következő ábrán egy ilyen pályatervezéssel készült, szép pálya látható.



7.1. ábra. RTR+T*TS

7.2. ábra. Pályatervezők összehasonlításának eredményei [13]

8 Összefoglalás

Összegzésképpen elmondhatom, hogy a célkitűzéseimet sikerült megvalósítanom a dolgozatom során. A tanszéki pályatervező C++ könyvtár megismerése után két olyan lokális pályatervező algoritmust is sikerült implementálnom, amelyek az előírt kinematikai feltételeket betartva, folytonos görbületű pályákat terveznek. A pályatervezők részletes összehasonlítása után választ kaptam arra a kérdésre is, hogy a különböző esetekben melyiket célszerűbb használni. Jövőbeli terveim között szerepel, hogy valódi robotokon is kipróbáljam az algoritmusokat és a futási időt optimalizáljam.

Köszönetnyilvánítás

Szeretnék köszönetet mondani konzulenseimnek, Csorvási Gábornak és Kiss Domokosnak a rendszeres konzultációkért és tanácsaikért.

Irodalomjegyzék

[1] G. Csorvási: Pályatervezési és pályakövető szabályozási algoritmusok fejlesztése robotautóhoz, Diplomaterv, December 2014

[2] S. M. LaValle, Planning Algorithms. Cambridge, U.K.: Cambridge University Press, 2006. Available online at http://planning.cs.uiuc.edu/

[3] D. Kiss–G. Tevesz, „A model predictive navigation approach considering mobile robot shape and dynamics," Periodica Polytechnica - Electrical Engineering, vol. 56, pp. 43–50, 2012.

[4] T. Fraichard, A. Scheuer: From Reeds and Shepp's to Continuous-Curvature Paths, IEEE Trans. on Robotics and Automation, Vol. 20, No. 6, December 2004

[5] E. Szádeczky-Kardoss: Pályatervezés autószerű járművekhez. Jegyzet a „Navigáció és pályatervezés " tárgyhoz, 2015

[6] International GeoGebra Institute, „GeoGebra" https://www.geogebra.org

[7] D. Kiss–G. Tevesz: „A Continuous-Curvature Steering Method for Efficient Path Planning of Car-like Robots," Journal of Robotics, (beküldve 2016. októberében, elbírálás alatt)

[8] D. Kiss–G. Tevesz: A steering method for the kinematic car using C*CS paths. In Proceedings of the 2014 15th International Carpathian Control Conference (ICCC) (konferenciaanyag). Velké Karlovice, Czech Republic, 2013. May, 227–232. p. ISBN ISBN: 978-1-4799-3527-7.

[9] D. Kiss–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 (konferenciaanyag). Sheffield, England, 2014. September.

[10] S. M. LaValle: Rapidly-exploring random trees: A new tool for path planning. Jelentés, 1998, Computer Science Dept., Iowa State University.

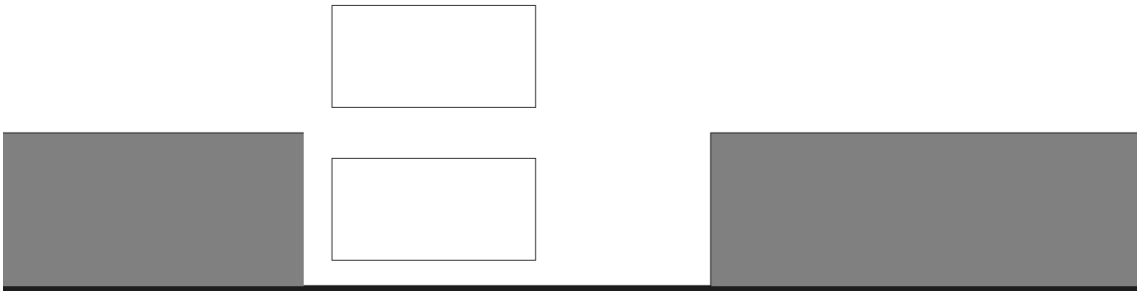
[11] I. Harmati: Ütközésmentes pályatervezési algoritmusok mobilis robotokhoz. Segédlet az „Autonóm robotok és járművek " tárgyhoz, 2008

[12] Minkowski algoritmus: http://www.dyn4j.org/2010/04/gjk-distance-closest-points/, (2016. Október)

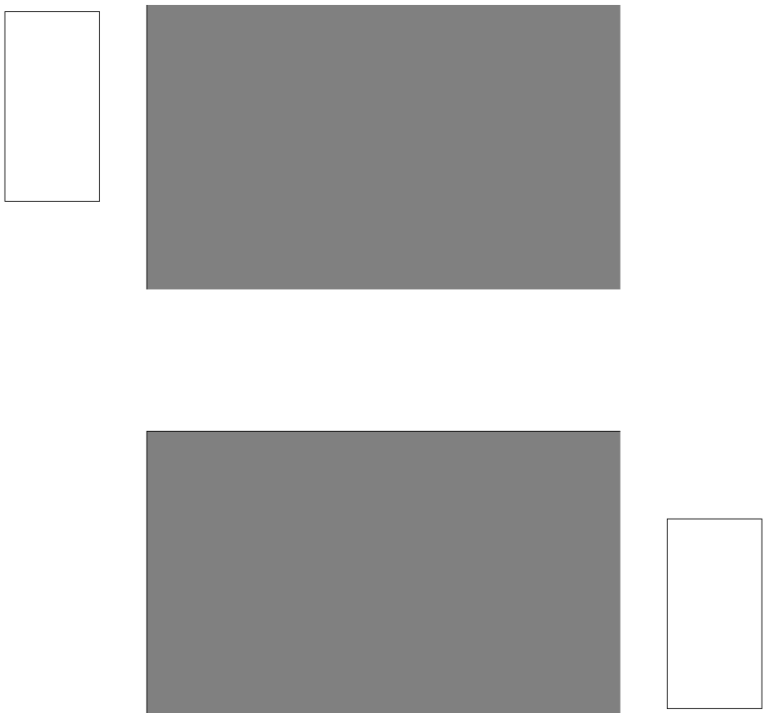
[13] D. Kiss–D. Papp: Effective Navigation in Narrow Areas: A Planning Method for Autonomous Cars, 15th IEEE International Symposium on Applied Machine Intelligence and Informatics (SAMI 2017), (beküldve 2016. októberében, elbírálás alatt)

[14] J. A. Reeds and L. A. Shepp: Optimal paths for a car that goes both forwards and backwards. Pacific Journal of Mathematics, 145:367–393, 1990.

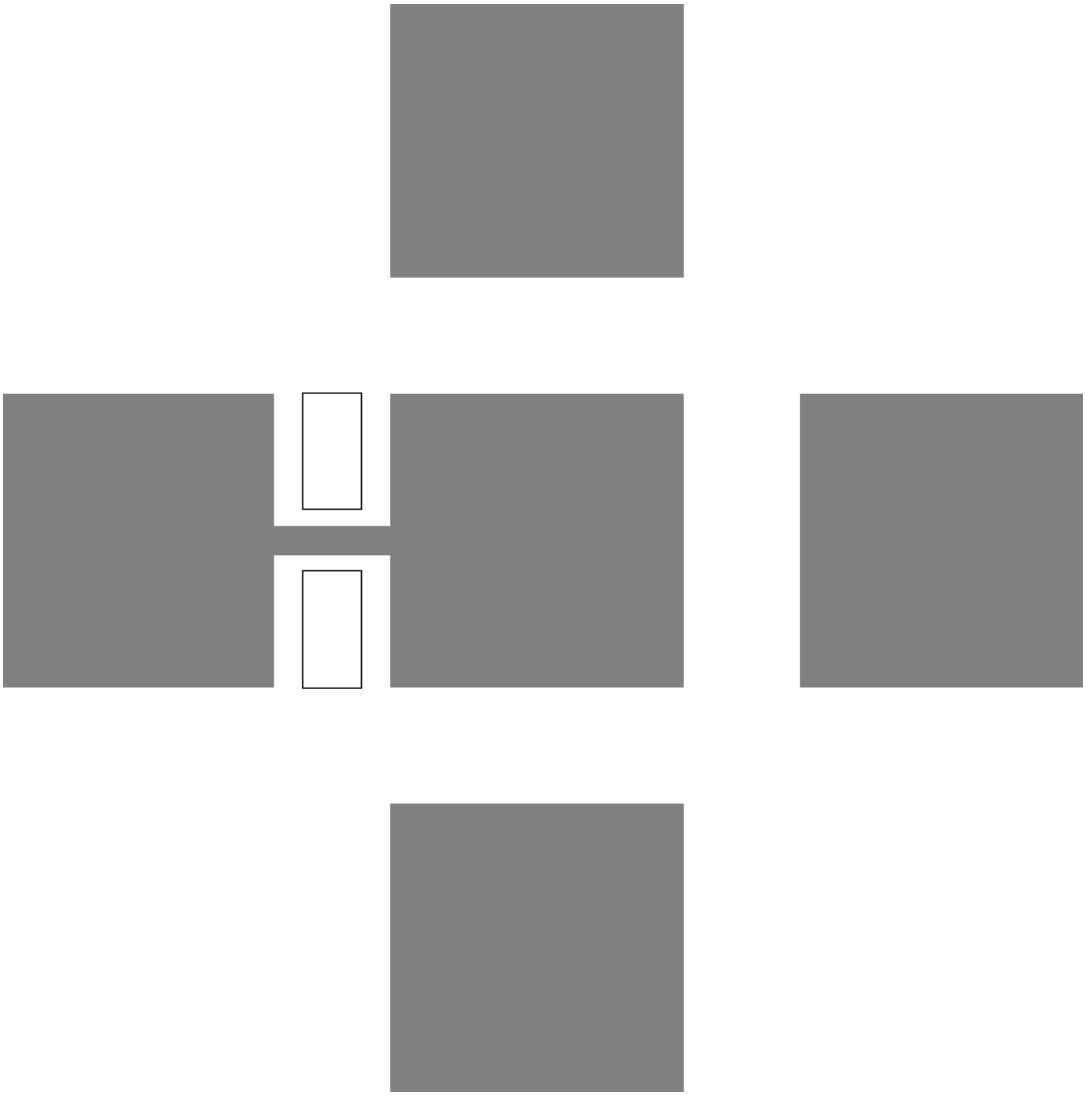
Függelék



8.1. ábra. Párhuzamos parkolás pálya



8.2. ábra. One corridor pálya



8.3. ábra. Three corridor pálya