

Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Pásztor Gábor Dávid

**Pályatervező algoritmus fejlesztése autószerű robotRA**

Konzulens

Csorvási Gábor

Kiss Domokos

BUDAPEST, 2018

Tartalomjegyzék

[Összefoglaló 3](#_Toc332798843)

[Abstract 4](#_Toc332798844)

[1 Bevezetés 4](#_Toc332798845)

[1.1 Formázási tudnivalók 4](#_Toc332798846)

[1.1.1 Címsorok 4](#_Toc332798847)

[1.1.2 Képek 4](#_Toc332798848)

[1.1.3 Kódrészletek 4](#_Toc332798849)

[1.1.4 Irodalomjegyzék 4](#_Toc332798850)

[2 Utolsó simítások 4](#_Toc332798851)

[Irodalomjegyzék 5](#_Toc332798852)

[Függelék 5](#_Toc332798853)

Hallgatói nyilatkozat

Alulírott **Pásztor Gábor Dávid**, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot/ 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 hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2018. 11. 03.

...…………………………………………….

Pásztor Gábor Dávid

Összefoglaló

Ide jön a ½-1 oldalas magyar nyelvű összefoglaló, melynek szövege a Diplomaterv Portálra külön is feltöltésre kerül.

Abstract

Ide jön a ½-1 oldalas angol nyelvű összefoglaló, amelynek szövege a Diplomaterv Portálra külön is feltöltésre kerül.

# Bevezetés

A mobil robotika az egyik legdinamikusabban fejlődő mérnöki terület jelen korunkban. Napjaink legkorszerűbb járművei az úgynevezett autonóm járművek, melyek képesek önállóan, emberi beavatkozás nélkül navigálni az adott környezetben anélkül, hogy az térben lévő akadályokba beleütköznének, nagy mértékben segítve ezzel a közlekedés hatékonyságát és biztonságosságát. Mivel a jövő para önálló mozgásra / közlekedésre képes járművek és robotok fejlesztése irányába halad elsősorban, ezért magam is ehhez a területhez tartozó témát választottam. Ezeknél a járműveknél kulcsfontosságú kérdés a megfelelő útvonal megtervezése, mind lokális, mind globális környezetben, amely figyelembe veszi a jármű mozgásállapotát és a környezet adatait, amelyek rendelkezésre állnak külső forrásból vagy különböző szenzoros mérések alapján.

Feladatom az útvonaltervező algoritmusok alapjainak megismerése, és egy ilyen algoritmus mélyebb áttanulmányozása implementálása és integrálása a tanszéken található programkönyvtárba, amelybe az eddigi évek hasonló témában készült projektjei vannak összegyűjtve. Dolgozatom első részében bemutatom a robotikai alapfogalmakat, és mozgástervezési módszereket, illetve az alkalmazott pályatervezési megközelítéseket. Ezután egy speciális pályatervező algoritmus: az Orientation-Aware Space Exploration Guided Heuristic Search (OSEHS) mélyebb ismertetésével foglalkozom. Ez az algoritmus a mozgástervezés hatékonyságát hivatott növelni szűk terekben történő navigáció (pl.: parkolási feladatok) során. Az algoritmust a mélyebb megértés és a későbbi hasznosítás céljából a tanszéken fejlesztett programkönyvtárba is, amely különböző hasonló célú programokat tartalmaz.

# Robotirányítási alapok

## Kereken mozgó robotok

Dolgozatomban a szárazföldi mobil robotok pálya tervezésesével foglalkozom. A kerekek segítségével robotok esetén a hajtott kerekek, kialakítása, elhelyezkedése és szabadsági foka határozza meg a robot helyzetváltoztatási lehetőségeit. Nincs általános, minden helyzetben jól működő konfiguráció, ezért ezt minden esetben a robot tervezése során feladatspecifikusan, a különböző elrendezések előnyeinek és hátrányainak figyelembevételével kell megválasztani.

A robotok elrendezését két nagy csoportban tárgyaljuk:

### Anholonom (*nonholonomic*) robotok:

Egy mechanikai rendszert akkor nevezünk anholonomnak, ha egy adott állapotában (konfigurációjában) létezik olyan elmozdulás/elfordulás irány, amely nem adható meg a lehetséges beavatkozások hatásainak algebrai kifejezéseként. Anholonom rendszerek legtipikusabb példája a személyautó, amely az aktuális orientációjára merőleges irányba nem tud elmozdulni. [1] A dolgozat további részében az ilyen robotokra koncentrálok, és az implementált algoritmust is anholonomikus robotkonfigurációra tesztelem.

### Holonom robotok

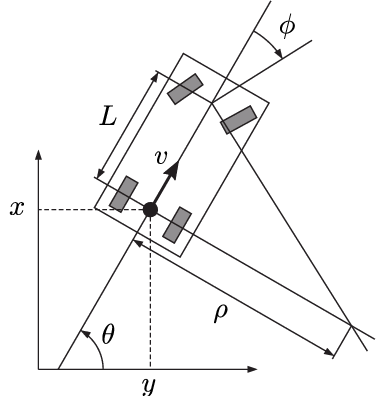
A holonomikus rendszerek viszonylag ritkák a gyakorlatban, hiszen ezen meghajtások sokszor bonyolult és drága mechanikát követelnek meg, illetve sokszor a helymeghatározáshoz használt odometria is pontatlan, ezért ezen konstrukciók bonyolultabb helymeghatározási módszereket követelnek meg.

Mechanikailag a legegyszerűbb, és legelterjedtebb konstrukciók az omnidirekciós (*omni* = minden, *direction* = irány) hajtású robotok. Ilyen lehet például egy lánctalpas megoldás, vagy ezen a hajtások egyik legérdekesebb kivitelezése Mecanum (Ilon) kerékkel oldható meg, melyet Bengt Erland Ilon talált fel a svéd Mecanum AB cég elnökeként. Ilyen omnidirekciós konstrukció például a KUKA Robotics omniMove szállítójárműve, amely kiválóan tud manőverezni olyan szűk és akadályokkal teli környezetben, mint egy raktár. [kép]

### Az autószerű robot modellezése

A jármű kinematikai modelljeként egy nemlineáris-egynyomvonalú modellt, úgynevezett „bicikli modellt” szoktak használni. Ebben a modellben a jobb első és bal első kerekek - a hátsó kerekekhez hasonló módon – egyetlen kerékként vannak reprezentálva. Ezen megközelítés mögött az az ötlet húzódik, hogy sok esetben a jármű hosszirányú kiterjedését nem szükséges figyelembe venni a kinematikai és dinamikai leírás során, mert az nem befolyásolja az oldalirányú vagy fordulási stabilitást. Az alábbi modellnek létezik 2, illetve 3 szabadságfokkal rendelkező változata is attól függően, hogy csak az első vagy a hátsó kerekek is mozgathatók.

Nyilvánvalóan a 2 szabadságfokú modell közelíti jobban egy autó mozgását. Fontos továbbá, hogy a két hajtott kerék tengelye egybeessen, a kerekek csúszásának elkerülése érdekében.

  
Autószerű robot modellezése [3]

A jármű aktuális mozgásállapota egy vektorral fejezhető ki a konfigurációs térben, ahol q az adott konfigurációt, x, y pozíciót, és θ a robot orientációját jelöli. Ez a mozgás egyenlet az ábra alapján is könnyen levezethető:

ahol L az autó két tengelyének távolsága, ϕ a kormányszög, v pedig a hátsó tengely középpontjának, mint a robot referenciapontjának tangenciális sebessége. [2] A fordulási sugár, pedig L és ϕ segítségével számolható a következő egyenlet szerint:

## Konfigrációs tér

A konfigurációs tér a robotmozgások leírására használható. A következő jelöléseket szokás alkalmazni:

* ***W* Munkatér** (workspace): az a környezet, amelyben a pályatervezés zajlik Descartes koordinátákkal leírva
* ***q* Konfiguráció:** Olyan n-dimenziós vektor mely egyértelműen jellemzi a robot statikus állapotát.
* ***C* Konfigurációs tér** (configuration space)**:** Egy lehetséges környezetben a robot összes lehetséges állapota.
* ***Cfree***  **Szabad konfiguráviós tér:** A konfigurációs tér akadályoktól mentes része
* ***Cobs*  Konfigurált akadályok halmaza:** Cfree komplemetere

A mozgástervezés során figyelembe kell vennünk különböző globális illettve lokális kényszereket, korlátokat. A globális kényszerek, azok melyek magából a környezetből adódnak: ezen korlátozások segítségével határozható meg Cfree. A lokális korlátozások pedig, a robottól származó kinematikai (pl.: sebesség, fordulási szög) és dinamikai (pl.: gyorsulás, szögsebesség) paraméterek, melyeket a robot konfigurációs változóinak differenciálegyenletével írhatunk le, ezért szokás őket differenciális korlátoknak is nevezni. A klasszikus mozgástervezési paradigmákban az ütközésmentes pályát csak a kinematikai korlátok figyelembevételével generálják. [4] Munkám során csak a kinematikai kényszereket veszem figyelembe az útvonaltervezés során.

## Mozgástervezők csoportosítása.

A mozgástervezési módszereket alapvetően kétféle megközelítés alapján csoportosíthatjuk: megkülönböztethetünk *globális tervezőket,* illetve *lokális- vagy reaktív tervezőket.*

A két csoport közti szignifikáns különbség, hogy a globális tervezők esetében a tervezés a teljes konfigurációs térben történik és ennek megfelelően a kezdeti és a célállapot is egy konfigurációval van reprezentálva. Ebben az esetben a pályatervezés „offline” történik, még a mozgás végrehajtása előtt: először egy geometriai pályát tervezünk, majd ezt időparaméterekkel ellátva kapjuk meg a mozgás trajektóriáját, amit a robot később végrehajt. Ennek a módszernek a legfőbb hátránya, hogy egy előre nem ismert akadály felbukkanása esetén az egész pályát újra kell tervezni.

A reaktív akadályelkerülés esetén a tervezési és végrehajtási fázis nem különül el a mozgás során. A szenzorokkal érzékelt környezetet és a jelenlegi konfiguráció alapján történik a következő mozgásállapot meghatározása „online”. Itt azonban fennáll a veszélye annak, hogy az algoritmus egy lokális minimumhelyen ragad. Fontos még megjegyezni, hogy mivel ebben az esetben nem használunk térképet és az „online” tervezés miatt szükség van az adatok gyors feldolgozására, így a robot alakját figyelmen kívül hagyva, azt körökkel közelítjük.

Jól látható, hogy a két módszer jelentős mértékben eltér egymástól és más-más feladatokban tudnak hatásosabban teljesíteni, ezért általában a kettőt együtt alkalmazzák az optimális cél elérése érdekében: reaktív akadálykerülés a lokális minimumok csapdájának elkerülésével. Az általam bemutatott OSEHS algoritmus is ezen elv mentén tervezi az útvonalat.

# . Pályatervező algoritmusok

Mivel korábbi tanulmányaim során nem találkoztam még hasonló mélységben pályatervező algoritmusokkal, ezért a munkám néhány széles körben elterjedt útvonaltervező algoritmus megismerésével kezdődött. Ez azért is volt kulcsontosságú, mert az általam bemutatott módszer nagy mértékben épít más algoritmusokra is (A\*, Reed-Shepp), illetve a teszteknél is össze tudtam vetni a hatékonyságát más módszerekkel.

## Reed-Shepp

Az autószerű robotok mozgástervezésére akadályoktól mentes környezetben A. Reeds és L.A. Shepp dolgozott ki 1990-ben egy metódust. A Reed-Shepp út alatt olyan utat értünk, amely 2 pontot összekötő legrövidebb utat értjük az euklideszi síkban, ahol az út görbületére megszorításokkal élünk, előírjuk az út kezdeti és befejező tangenseit és feltételezzük, hogy az út mentén haladó jármű csak előre vagy hátra haladó mozgást tud végezni. Ha a tolatási manőver tiltott akkor Dubins-utat kapunk. [Wiki link]

A szerzőpáros átfogó cikkében [cikk link] bebizonyította, hogy egy ilyen útnak összesen 48 különböző megoldása van, melyek 5 különböző egyenes vonalú vagy köríven történő alap mozgásformából tevődnek össze. [oldal link] Egy ilyen út látható az alábbi ábrán is: [ábra]

Mivel a Reed-Shepp tervezés során a fordulás mindig a legkisebb fordulási sugárral történik, amely nem túl gyakori valós vezetési szituációkban, az eredmény szuboptimális megoldásra vezet az esetek többségében. Azonban ez a módszer nagy mértékben növeli a mélységi kereséssel előállított utak hatékonyságát, így én alkalmaztam az OSEHS algoritmus implementálása során.

## RRT

A Rapidly-exploring Random Tree (RRT) algoritmus különösen alkalmas többdimenziós terek gyors feltérképezésére, de autószerű robotoknál is gyakran alkalmazzák, hiszen egyszerűen figyelembe vehetők a jármű differenciális korlátai az alkalmazása során.

Ez az algoritmus véletlenszerűen generált minták segítségével épít fel egy fa struktúrát a tér felderítéséhez, melynek gyökere a kiindulási konfiguráció. Ha a random generált új *q* ϵ Cfree (tehát az adott konfigurációval a kocsi nem ütközik akadályba), akkor ezt az új állapotot felvesszük a fába. [kép]

Mivel ez az algoritmus rendkívül széles körben elterjedt így több különböző fajtája is létezik, ilyen többek között az RRT\* mely folyamatosan, futási időben optimalizálja a keresést, vagy az RRT-Connect két különböző struktúrát épít egyszerre, a teljesítmény növelése miatt. [forrás] Ezek az algoritmusok a keresési metódus véletlenszerűsége miatt a gyakorlati, online pályatervezési feladatok során kisebb hatékonysággal alkalmazhatóak.

## Hybrid A\*

Az A\* algoritmus gráfok bejárására használható, mely egy adott kiindulási csúcstól keres utat a célig csúcsig költségekkel súlyozott gráfokban; úgynevezett informális kereséssel ami azt jelenti, hogy a keresés mindig a legkisebb költségű csúcs felé halad. A felhasznált költségfüggvény hasonló heurisztikát/becslést alkalmaz az általam bemutatott OSEHS algoritmushoz.

Ezen algoritmus egyik változatát, a Hybrid A\*-ot alkalmazzák leggyakrabban anholonom robotoknál, mely egy rácsháló segítségével deríti fel a rendelkezésre álló utat. Bár a hasonló módszerek sokszor kompromisszumra kényszerülnek teljesítmény terén, amikor rácsháló segítségével diszkretizálják a konfigurációs teret. A Hybrid A\* és a hasonló elven működő algoritmusok esetén is az optimális heurisztika meghatározásának bonyolultsága gyakran összemérhető magával az útkeresési feladatéval. Bár ezen módszerek segítségével nagyban növelhető a valós idejű végrehajtás teljesítménye, figyelembe véve, hogy a különböző térképen történő navigáláshoz gyakran különböző felbontású rácshálók generálása szükséges. [kép]

# . Az OSEHS algoritmus

## Probléma felvetés

Az útvonaltervezés során a következő problémát kell megoldani: adott bemenetként a kiindulási konfiguráció *qstart*, és a cél konfiguráció *qgoal*, melyek *C* részei, ezentúl ismerjük még a környezet geometriai leírását is. A feladatunk, hogy találjunk *qstart* és *qgoal* olyan konfiguráció sorozatot melynek minden eleme része *Cfree*-nek. Ez a trajektória lesz az algoritmus kimenete.

Az alább ismertetett módszer 2 lépésből tevődik össze. Az első rész a *Felderítés* (*„Space Exploration”*), amely körök segítségével térképezi fel a munkateret. A Felderítés kimenete egy átlapolódó körökből álló folyosó a kiindulási és az elérni kívánt pozíció között („Circle-Path”). A Kör-folyosó információt szolgáltat a későbbiekben az akadálymentes térrészek elrendezéséről és méreteiről, továbbá a mozgásállapotok kiterjesztésének feltételezett irányáról is. A második szakasz a Heurisztikus Keresés („Heuristic Search”), amely a kinematikai korlátozások figyelembevételével keres utat Cfree-ben, miközben figyelembe veszi a Felderítés során előállt információt a körök sugarainak felhasználásával. A céltól vett becsült távolság és a sugarak figyelembevételével a kereső algoritmus egy hatékony tervezési megoldást szolgáltat anélkül, hogy a munkateret felosztanánk rácsokra (lásd Hybrid A\*).

Az általam bemutatott OSEHS algoritmus a feltérképezés során a körök orientációját is eltárolja ezzel segítve az állapotokkiterjesztését szűk folyosókon. Az algoritmusnak létezik egy olyan változata, mely ezeket az információkat nem használja fel (lásd: [link]), ezzel azonban ront a teljesítményben olyan forgatókönyvek esetén, mikor a járműnek keskeny hely áll rendelkezésre a megfelelő manőverek végrehajtásához (pl.: parkolási szituációk). Tekintve, hogy a mai korszerű vezetéstámogató rendszerek közül a park-pilot rendszerek egy széles körben használt és alkalmazott eszközök, úgy gondolom, továbbá a városi közlekedésben gyakran van szükséges keskeny helyen való pontos vezérlésre én is ezzel a változattal ismerkedem meg mélyebben.

## Felderítés - Space Exploration

A felderítés mindig a munkatérben kerül végrehajtásra, ami az általam vizsgált autószerű robotok esetében a 2 dimenziós Descartes-féle koordináta rendszer. Ebben az algoritmusban a szabad térrész feltérképezéséhez használt alakzat a kör, melynek a leggyorsabban és legegyszerűbben számítható az ütközése más alakzattal, és a távolsága egy adott ponttól. A körök kiterjesztése a start pozíciótól a cél felé történik, mélységi kereséssel, számításba véve mind a céltól való euklidészi távolságot, mind az orientáció beli különbséget az aktuális kör célkonfiguráció között, az előre és hátra irányok megkülönböztetésesével. Az algoritmus a következő: [algo]

*Sopen* egy tároló mely a felderítendő köröket tartalmazza, *Sclosed* pedig azokat amelyeket a későbbiekben még ki kell terjesztenünk. A kezdő kört *qstart* pozíciójában hozzuk létre a robot beírható körének sugarával (ez a robot paramétereiből adódik). Minden egyes iterációs ciklusban kiválasszuk a még nem feltérképezett körök közül, azt amelyiknek az összköltsége (*„TotalCost”*) a legkisebb. Egy kör összköltsége a következőképpen határozható meg:

A heurisztikus távolság meghatározásánál a kör irányát is figyelembe vesszük. Legyen az euklideszi távolsága az *ci* és *cj* körök középpontjának, továbbá ugyanezen körök közötti abszolút szögkülönbség a intervallumra korlátozva. Figyelembe véve a jármű kialakításából adódó *rmin* minimális fordulási sugarat, meghatározhatjuk, hogy az adott orientációkülönbség áthidalásához szükséges táv egyenlő -rel. Két kör közötti távolság egyenlő a fentebb részletezett két érték maximumával:

Ha a kiválasztott kör nem létezik a már feltérképezettek között (*NotExist(chosen,Sclosed)*) akkor kiterjesszük. A kiterjesztést az *Expand(chosen circle)*függvény segítségével történik úgy, hogy adott lépésközzel mintavételezzük a kör pontjait és a mintavételi pontokban megpróbálunk gyerek-köröket létrehozni. A generált körökhöz két-állapotú irány-információt rendelünk (Előre/Hátra), annak függvényében szülő-körhöz képesti szögkülönbségük benne van-e a intervallumban. [kép]

Egy újonnan generált kör sugara megegyezik a körhöz legközelebbi akadály távolságának és a jármű belső körének sugarának a különbségével. A redundancia és a futási idő csökkentése miatt figyelünk rá, hogy ne hozzunk létre végtelen kis sugarú köröket, illetve olyanokat, amelyek egy bizonyos mértéknél jobban fedik az előző köröket (pl. 50%) és az orientációik megegyeznek. Fontos odafigyelni, hogy a körök, az orientációjuk figyelembevételével egy háromdimenziós (*x,y,θ*) térben értelmezettek, így a megfelelő működés érdekében a különböző irányba lévő de egymást fedő köröket nem szabad eldobni. Így sokkal több kör generálódik, de a későbbi lépéseket nagy mértékben megkönnyíti az állapotok felderítését a többletinformáció a kívánt mozgásirányról. [kép]

A kiterjesztett „gyerek körök” minden adata (pl.: pozíció, orientáció, heurisztikus távolság…) ebben a függvényben kerül kiszámításra. Végül pedig miután az új körök kigenerálódtak, el kell őket tárolni *Sopen*-be. Ez az iteráció addig folytatódik amíg van feltérképezendő kör (*Sopen* *!= 0*), vagy amíg nincs átlapodó körünk a cél körrel. A cél kört a kiindulásihoz hasonló módon határozzuk meg.

A cél elérése után a kör-folyosó előállítható az ősök visszavezetésével a kezdeti állapotig mélységi keresés által létrejött fa-struktúrában.

## Heurisztikus Keresés.

Mielőtt elkezdődne a keresés a körök által meghatározott úton, szükség van egy előfeldolgozásra, hogy meghatározhassuk a lehetséges mozgásirányokat és azokat a régiókat, ahol manőverezésre lehet szükség. Ha egy kör irányvektora a leszármazottja felé mutat akkor előre mozgás szükséges, egyéb esetben pedig hátra mozgás. Azonban a kinematikai korlátok figyelembevételével látható, hogy nem mindig tudjuk követni a körök által meghatározott utat a korlátos fordulási sugár miatt. Ezekben az esetekben további manőverezésre lehet szükség -egymás utáni előre és hátra mozgásokra is- annak érdekében, hogy követni tudjak körök által kijelölt utat. Az előfeldolgozás után a körök három kategória szerint vannak elosztva, a mozgásirányoknak megfelelően: *Előre (Forward), Hátra (Backward), Kétirányú (Bidirectional).* A Heurisztikus Kereső algoritmus a Felderítéshez hasonló elven működik, azzal az alapvető különbséggel, hogy megfelelő méretű geometriai alakzatok helyek, lehetséges konfigurációkat keres, ahogy azt a következő algoritmus is jól mutatja: [algo]

Az állapotokat a körök pozíciójához hasonlóan összköltség alapján a sorrendezzük, és amíg lehetséges minden iterációs ciklusban kiválasztjuk a minimális költségűt a még feltérképezetlen állapotok közül (*MapNearest(chosen\_state)*). A kiválasztás után meghatározzuk az adott állapothoz legközelebb álló kört a kör-folyosó mentén, és az állapothoz rendeljük azt.

Az állapotok heurisztikus távolsága meghatározható a kör-folyosón hátralévő út segítségével. A távolságbecslést a következő összefüggés segítségével számítjuk:

ahol *dq,c* a *q* aktuálisállapot és a következő *c* kör távolsága, *dc*  pedig ezen *c* kör távolsága a célban meghatározott pozíciótól a kör-folyosó mentén. A gyakorlatban a c-kör mindig az állapothoz legközelebb eső körhöz képesti következő az út mentén. Így könnyedén biztosítható, az állapotok folyamatos haladása a kijelölt cél irányába. A kör-folyosóban lévő köröket célszerű 2 irányú láncolt listában tárolni, nem csak úgy, hogy a gyermektől a szülő irányába legyen hivatkozás eltárolva. Ha így teszünk akkor mindig tudjuk, melyik a következő kör az útvonal mentén.

A kiválasztott állapot kiterjesztése (*Expand()*) különböző primitív mozgások segítségével történik. Az előfeldolgozás során előállt az állapotok további kiterjesztéséhez javasolt mozgásirány a Space Exploration rész orientáció-információi alapján. Ezeket az információkat két féle módon használhatjuk fel a továbbiakban: különböző primitív mozgások alkalmazásával két irányú mozgások esetében vagy a javasolt irányoknak megfelelő súlyozások alkalmazással a kiválasztás során. Én az implementáció során az utóbbi megoldás mellett döntöttem ezért, ha a körök orientációjának ellentétes irányba terjesztem ki a gyerek állapotot előre, vagy hátra irány esetén, akkor ezt a heurisztikus távolság kiszámításánál „büntetéssel” veszem figyelembe. Kétirányú körök estén pedig nem változatok a heurisztikus távolságok arányán. Ennek az az előnye, hogy a kódban könnyebben implementálható és a robosztusabb megoldást biztosító paraméterek megtalálása is egyszerűbb feladat. A másik lehetőség pedig, hogy „Bidirectional” esetben manőverező mozgásokkal haladunk az alap előre, hátra és forduló mozgások helyett. Ilyenkor megfelelő „csúcsos” manőverek segítségével próbáljuk meg beállítani a tovább haladás megfelelő szögét, de ezen apró manőverek alkalmazása a hétköznapi közlekedési szcenáriókban, jobban elbonyolítja az algoritmus, mint amennyi teljesítmény növekedést tudunk elérni a segítségével.

A primitív mozgások a jármű minimális fordulási sugarának figyelembevételével, adott lépésközzel generált alapvető mozgástípusok. A lépésköz megválasztása a körök sugarának segítségével történik, kihasználva a feltérképezés során nyert helyinformációkat:

,

ahol α, β együtthatók, melyek tapasztalati úton adódnak, rc és dc az adott kör sugara, illetve a céltól vett becsült távolsága, c pedig az állapothoz rendelt kör. Az smin pedig egy minimális lépésköz amely elhagyható abban az esetben, ha a felderítés során a kör sugarát egy adott értékben korlátoztuk, a végtelen ciklusok elkerülése érdekében. Mielőtt létrehozzuk az új állapotot meg kell győződni róla, hogy nem ütközik-e akadályba az adott konfigurációval a jármű, ugyanis az autószerű robot kinematikai korlátozásai miatt nem mindig tudjuk követni a felderítés során előállt kör-utat. Ha az újonnan létrehozott qnew ϵ Cfree, akkor qnew eltárolható az újonnan létrehozott állapotok közé. Mivel a keresőalgoritmus mindig a fa minimális elemét veszi, így az útvonaltervezés nagy lépésközökkel operál, ha nagy szabad hely áll rendelkezésünkre és messze van a cél.

Az új állapotok kiterjesztése addig folytatódik amíg el nem érjük a kijelölt cél konfigurációt vagy amíg van lehetséges új kiterjesztendő állapot. A fent részletezett módszer azonban módszer egyik alapvető hiányossága, hogy a célkonfiguráció szinte soha nem érhető el teljesen pontosan. Hasonló esetekben, például a Hybrid A\* algoritmusnál[hivatkozás] is Reed-Shepp tervezőt használnak, ha az algoritmus egy bizonyos távolságra megközelíti a célt. Az algoritmus hatékonyságának növelése érdekében én minden egyes iterációs ciklusban megpróbálok egy Reed-Shepp utat tervezni.

# Implementáció

A mozgástervezési alapfogalmakkal és az alkalmazott algoritmusokkal való részletes megismerkedés után az OSEHS algoritmust implementáltam, kezdetben egy egyszerű script segítségével, majd pedig a tanszéki útvonaltervező programkönyvtárba is.

## Python script

A Python egy általános célú, magas szintű, interpreteres programozási nyelv, mely azt jelenti, hogy nincs különválasztva a forrás és tárgykód: a megírt program máris futtatható, ha rendelkezünk a Python értelmezővel. A nyelvnek igen kiterjedt és széles körű standard könyvtára van, amit még kiegészítenek a mások által megírt publikus modulok. Az gyors kezelhetőség miatt választottam ezt nyelvet, és amiatt mert egyes modulok segítségével egyszerűen lehet vele ablakos környezetbe rajzolni.

Először az algoritmus Space Exploration részét implementáltam. Pythonban az implementáció és a debuggolás is nagyon egyszerűen kivitelezhető volt, mivel az eredményt egy Windows ablakban rajzoltam ki, ezért a pályaszerkesztés és az útvonaltervezés eredménye is valós időben hozzáférhető volt grafikus felületen, ami nagyban segítette a megértést és a fejlesztést. Ezután az állapot kiterjesztést (*Heuristic Search: Expand()*) is hozzáadtam a script-hez, azonban itt nem vettem figyelembe a kinematikai megszorításokat, és ezért a köríven való fordulást sem, hiszen elsődleges célom az állapotkiterjesztés folyamatának a megértése volt. A program kimenete az alábbi ábrákon látható: [kép]

## C++

# Validáció

Irodalomjegyzék

1. Dr. Tevesz Gábor: Robotirányítás rendszertechnikája elektronikus jegyzet (2017.12.08)
2. Csorvási Gábor: *Pályatervezési és pályakövető szabályozási algoritmusok fejlesztése robotautókhoz* (2014.11.15)
3. <https://www.researchgate.net/figure/Model-of-a-differential-drive-left-and-a-car-like-robot-right_fig1_282050458>
4. Chao Chen, Markus Rickert, Alois Knoll: Combining Space Exploration and Heuristic Search in Online Motion Planning for Nonholonomic Vehicles
5. Chao Chen, Markus Rickert, Alois Knoll: Path Planning with Orientation-Aware Space Exploration Guided Heuristic Search for Autonomous Parking and Maneuvering
6. Wikipedia: Evaluation strategy, <http://en.wikipedia.org/wiki/Evaluation_strategy> (revision 18:11, 31 July 2012)
7. http://planning.cs.uiuc.edu/node822.html
8. Wikipedia: Dubins path, <https://en.wikipedia.org/wiki/Dubins_path>   
   (revision: 22 June 2018, at 10:05)
9. Reeds, J.A. and L.A. Shepp: Optimal paths for a car that goes both forwards and backwards (1990)

Függelék