# Multi ágensű robotrendszerek *Bevezetés*

Multi-ágensű mobilrobotrendszerek

# Robotrendszerek felosztása

- A robotrendszerek felosztását többféle szempontból kategorizálhatjuk:
  - Felosztás **mobilitás** szerint
    - Statikus vs. **Mobil**
  - Felosztás méreteik szerint
    - Ipari méretek/mikro/nano-robotok
  - Felosztás a felhasználásuk szerint:
    - Ipar/egészségügy/veszélyes környezetben való munka/felderítés-feltérképezés/szórakoztató ipar/…
  - Felosztás az intelligencia szintjük alapján
    - Dump-vezérlő nélküli robotok/egyprocesszoros,-/többprocesszoros rendszerek
  - Felosztás az ágensek száma szerint
    - Egy ágensű / **több ágensű** rendszerek (**MAS** – Multi Agent Systesm / **MAR** – Multi Ágensű Rendszerek)
  - ….

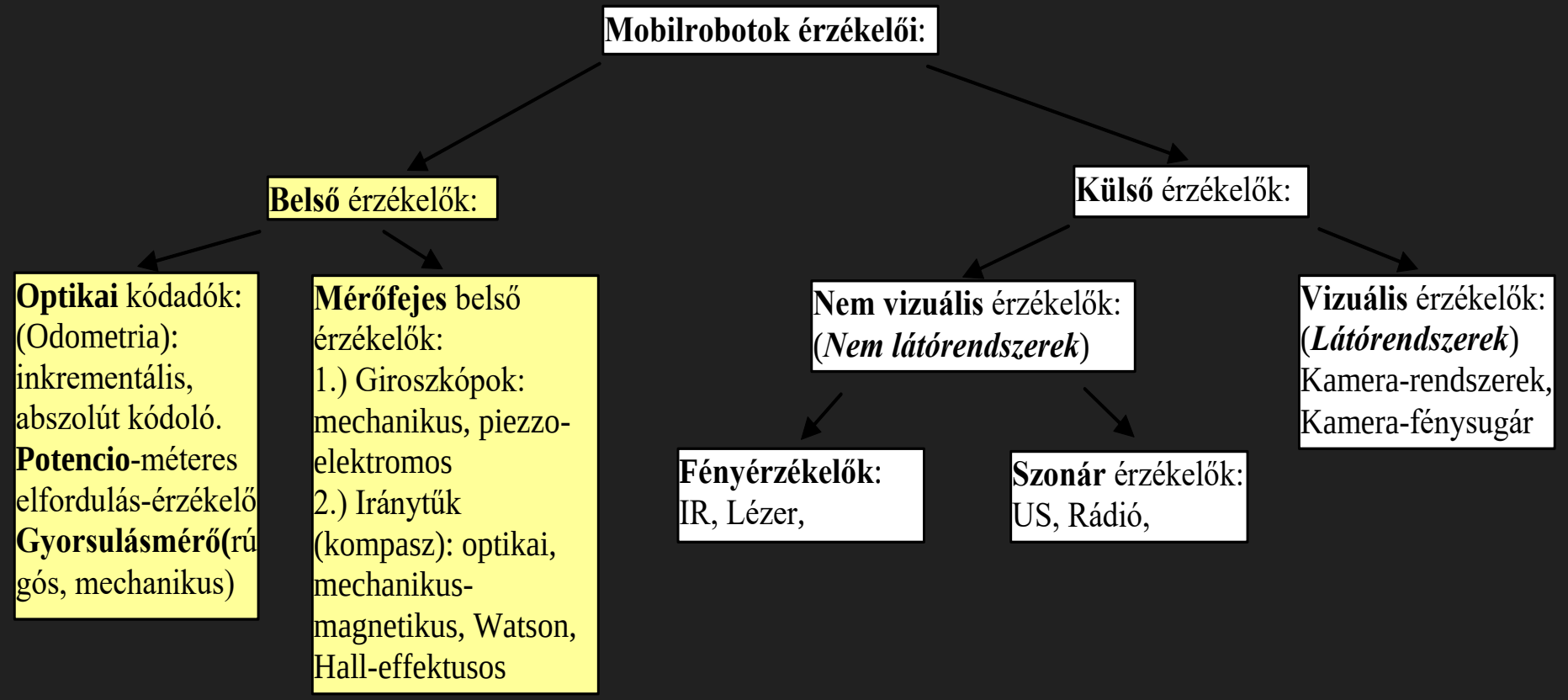
# A mobilrobot rendszerek további felosztása

- Kötött pályán mozgó mobilrobot rendszerek
  - Sínpálya/kötélpálya/indukciós vonal/festékvonal/fénysáv (IR, Lézer)/, …
- Szabadon mozgó mobilrobot rendszerek
  - Terület feltérképezése/terület értelmezése (felosztása)/Start-Cél kijelölés/pályatervezés/navigáció

# Szabad területen mozgó mobilrobotok (MR)

- Hardver felépítése
  - Test/járószerkezet (kinematikák-vezérlő paraméterek)
  - Motorok kiválasztása, megbecslése
  - Elektronika
    - Vezérlő elektronika: dump/1 µP/multi-proc. / proc. elhelyezése (on-board, off-board
    - Tápellátás/motor hajtások
    - Kommunikáció (kötött/RF/opt)
  - Szenzorrendszer (külön tárgyalva)
    - Belső szenzorok
    - Külső szenzorok

# Szenzorrendszerek felosztása



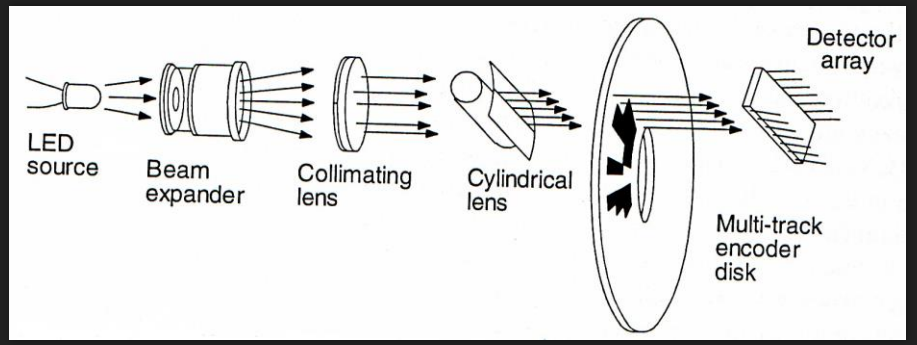
# Legfontosabb érzékelők - odométer

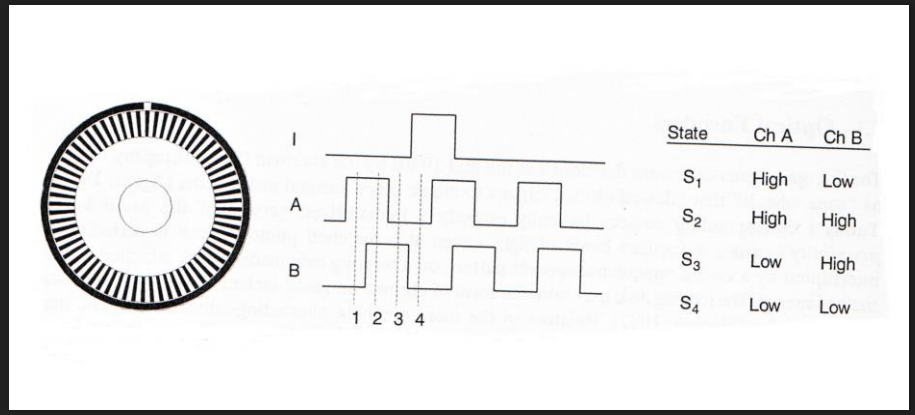
**Optikai inkrementális kódadó**: Az odometria széles körben használatos a kerékkel felszerelt mobilrobotokkal kapcsolatos navigációs eljárásoknál, metódusoknál. Egyik legismertebb kivitelezési formája, a keréken (vagy a keréktengelyen) elhelyezett kódtárcsa, egy inkrementális optikai kódadó. A kódtárcsa felbontása, többek között meghatározza a helyzet-meghatározás pontosságát is, de egyben tájékoztatást is ad a mobilrobot sebességéről. Ember-közelibbé téve, azt lehet mondani, hogy a kódtárcsa úgy működik, mintha egy ember becsukott szemekkel lépked egy bizonyos irányba, és számolja lépteit (gyorsaságát és mennyiségét). Vagyis ennek megfelelően egy *diszkrét* mozgást végző mobilrobot abszolút ($x$) pozícióját, - egyenlőre még *hibamentes* környezetet feltételezve-, a következőképpen számolhatjuk: 1.: **diszkrét** környezetben; 2.: **analóg** környezetre;

$$\vec{x} = \sum \vec{\delta}_{(i)};$$

$$\vec{x} = \int_{t_0}^{t_f} \frac{d\vec{x}}{d\vec{t}} dt;$$

| RENDSZERES hibák | SZTOCHASZTIKUS hibák |
|---|---|
| Különböző kerékátmérő | Egyenetlen talaj |
| A valós kerékátmérő különbözik a névlegestől | Váratlan akadály a talajon |
| A valós tengelytávolság különbözik a névlegestől | Kerékcsúszás:<br>▶ csúszós padló<br>▶ gyorsulás, kanyar<br>▶ külső erőbehatás<br>▶ kerék és a padló érintkezése nem egyenletes,… |
| Rossz kerék-geometria (összefutás/átmérő) | |
| Véges kódtárcsa felbontás | |
| Véges mintavételezés a kódolóról | |



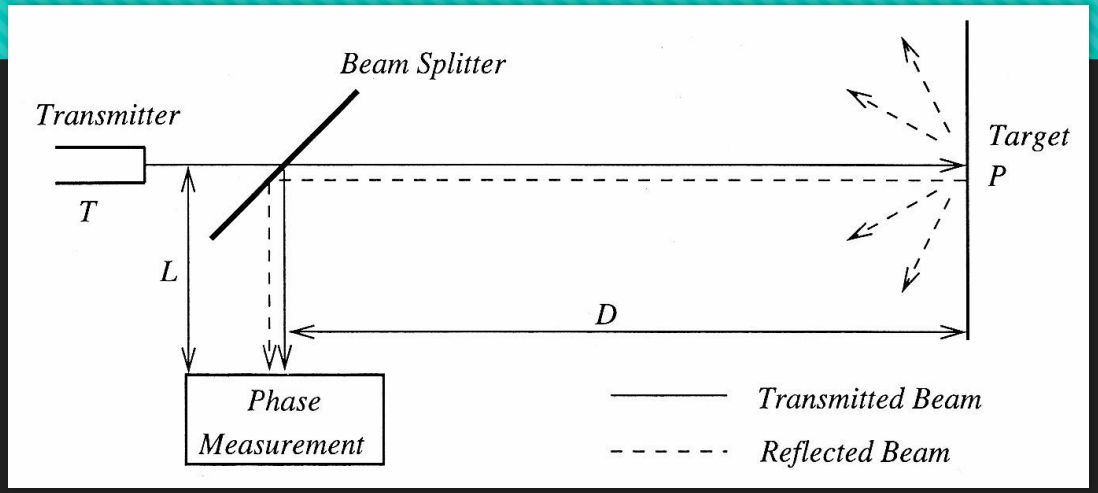



# Távolságmérés – IR / Lézer

## Phase Shifting

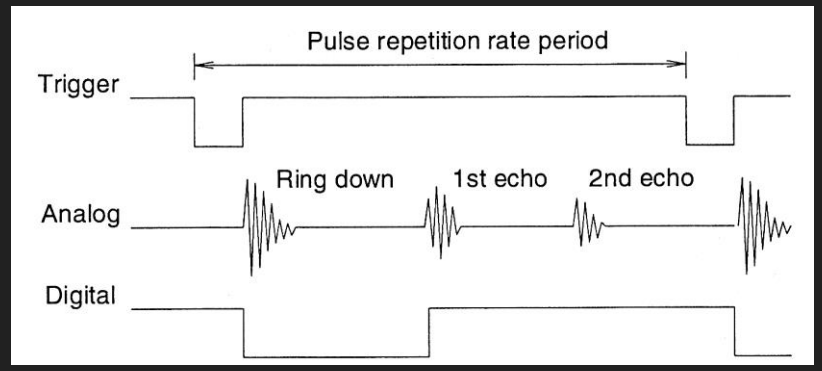
$$D = L + \lambda \frac{\theta}{2\pi}$$

**Theta** – a mért fáziskülönbség

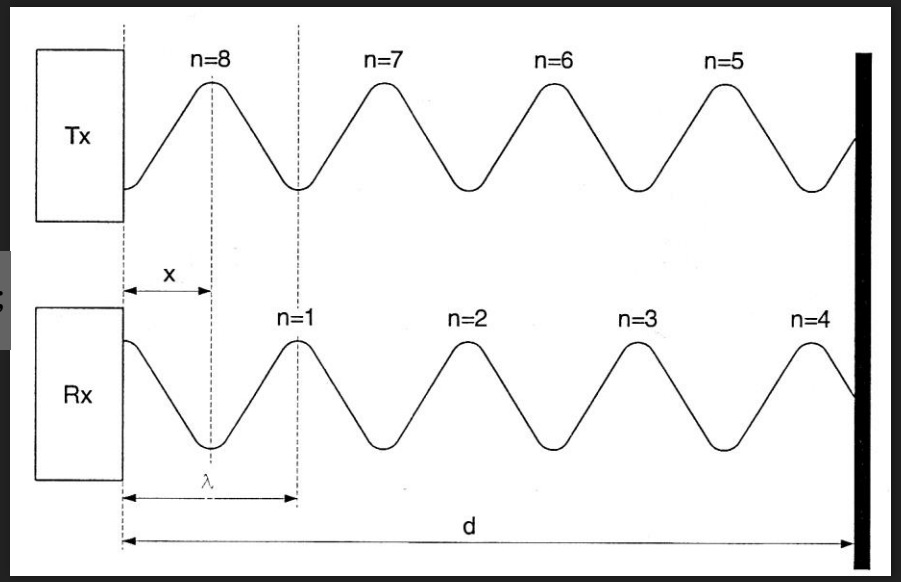



## TOF

$$d = \frac{1}{2} v.t;$$




$$\phi = \frac{4\pi d}{\lambda} \Rightarrow d = \frac{\phi\lambda}{4\pi}; (ha\ \lambda = \frac{c}{f}) \Rightarrow d = \frac{\phi c}{4\pi f};$$



# MR pályatervezések alapjai

„ Hol vagyok? Hova megyek? Hogy jutok el oda?”

- Pozícionálás/ lokalizálás (relatív (odometria) vs. abszolút pozícionálás)
  - Odometria/Tri-anguláció/Tri-lateráció/mesterséges vs. természetes markerek/modell illesztéses pozícionálás.
  - Térképkészítés – szenzortérkép/geometriai térkép/topológiai térkép (grág-térkép)/hibatérkép
  - Pályatervezés
  - Navigáció

# Lokalizáció

- **„Pozícióink megállapítása egy globális referenciaponthoz képest”**

- Annak megállapítása, hogy hol vagyunk a környezetünkben. (általában ismert pozíciójú markerektől állapítjuk meg távolságunkat, mely alapján kiszámoljuk pillanatnyi helyzetünket a környezetben és a mérések (számolások) alapján elhelyezzük magunkat az adott környezet előre megadott (ismert környezet) térképében. = LOKALIZÁCIÓ ).

- **Globális lokalizáció** – Elhelyezkedésünk megállapítása a környezetben. (Pontatlan, de a tájékozódás szempontjából nagyon fontos. Magába foglalja a környezet ismeretét.)

- **Lokális** lokalizáció – Pontos elhelyezkedésünk a közvetlen közelünkben lévő akadályokhoz képest.

- A lokalizációval kapcsolatos kulcskérdés az érzékelőktől kapott adatokból egy modell felállítása, és utána a modellt beillesztése a környezetről kapott térképbe. Az illesztésnél *három kategóriát* emelhetünk ki:

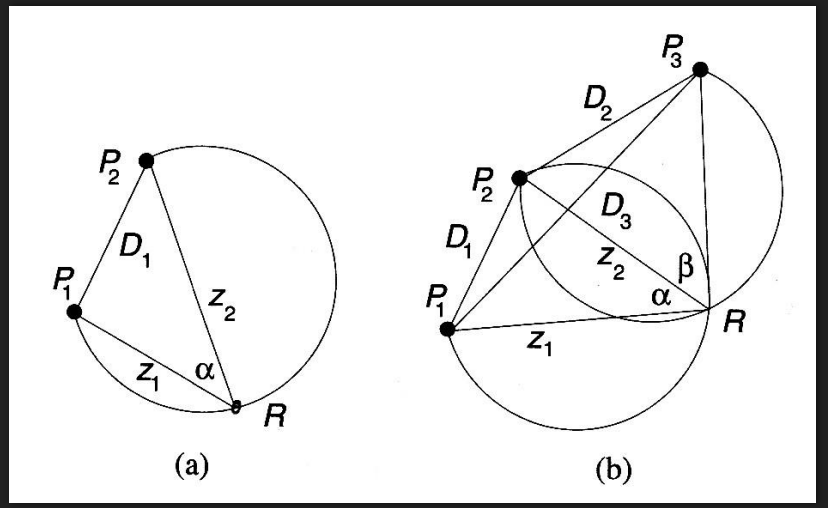
- ***Adat – adat illesztés***: közvetlen összehasonlítás az érzékelők mérési adatai, és a térkép alapján kiszámolt adatok között (markerektől mért távolságok).

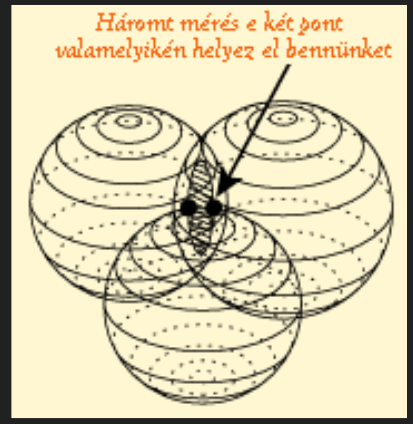
- ***Adat – modell illesztés***: A mérési adatok összeegyeztetése a térképben tárolt absztrakt modellekből kiszámolt adatokkal (akadályoktól mért távolságok).

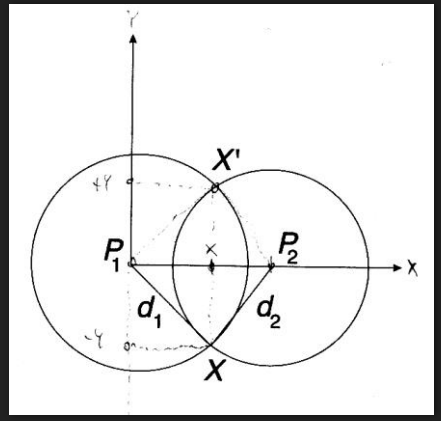
- ***Modell – modell illesztés***: A mérésekből kapott adatokból összeállítunk egy modellt, és ezt a modellt hasonlítjuk össze a térképben szereplő (és a memóriában tárolt) geometriai, absztrakt modellek adataival.

# A lokalizáció módszerei

- Tri-anguláció (3 markertől)
- Bi-lateráció (2 markertől)
- 3-4-5 – laterációk illetve angulációk (GPS)



# Munkaterület

## Ismert

- Az akadályok pozíciója, mérete, mozgása ismert

## Ismeretlen

- Az akadályok pozíciója, mérete, mozgása nem ismert

# Munkaterület

## Sztatikus

- Álló akadályok

## Dinamikus

- Mozgó akadályok - ágensek

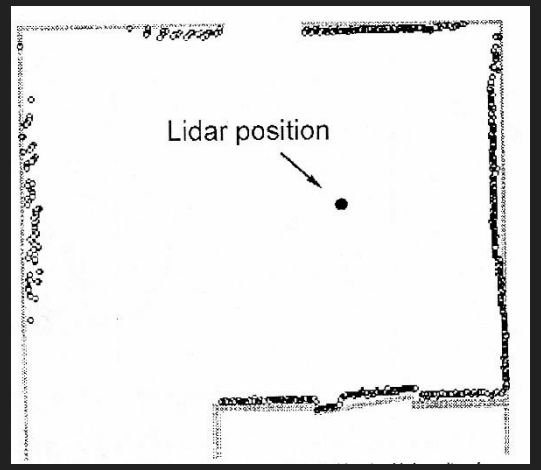
# Ismeretlen munkaterület feltérképezése

- Szenzortérkép

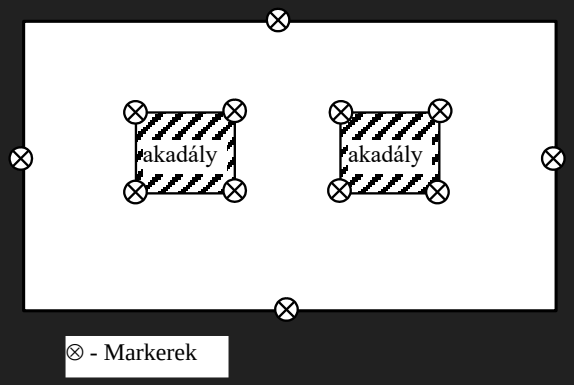
A robot megállapítja pozícióját, majd az érzékelői segítségével „körbenéz" és bejelöli az adott pozícióból érzékelt akadályokat, továbblép és újra ismétli ⇒ **lokális** térképek ⇒ illesztés: globális térkép ⇒ **Érzékelők adataiból készített** térkép.

- Az érzékelők adataiból készülhetnek **metrikus – geometriai** térképek, vagy, pl. az odometria segítségével **topológiai** térképek (az odométerről jövő adatokból: utak = gráfok élei, kereszteződések = csomópontok).
- Az egész környezet térképének összeállítása a lokális térképek **illesztésével** lehetséges. Az illesztés két alapon működhet:
  - -**ikonszerű** illesztés: az egyes lokális térképek (amik a pozícióváltásokkal keletkeznek a környezetről) közös pontjaikat keresik, és ez alapján illesztik egymás mellé az „ikonokat" → *globális* térkép.
  - -**jellegzetességek** alapján történő illesztés: A pozícióváltás után keletkezett lokális térképek jellegzetes pontjait keresik, majd ezeket illesztik egymáshoz → *globális* térkép.

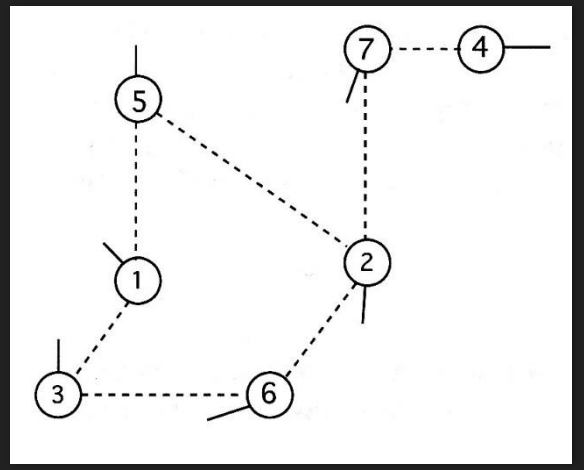
# Térképtípusok



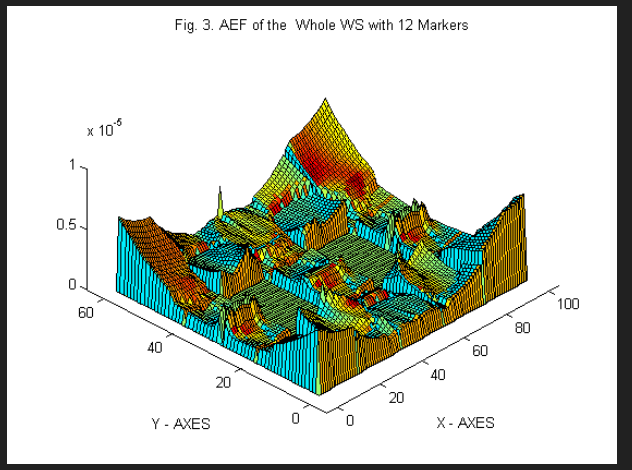

Szenzortérkép




Metrikus-geometriai térkép



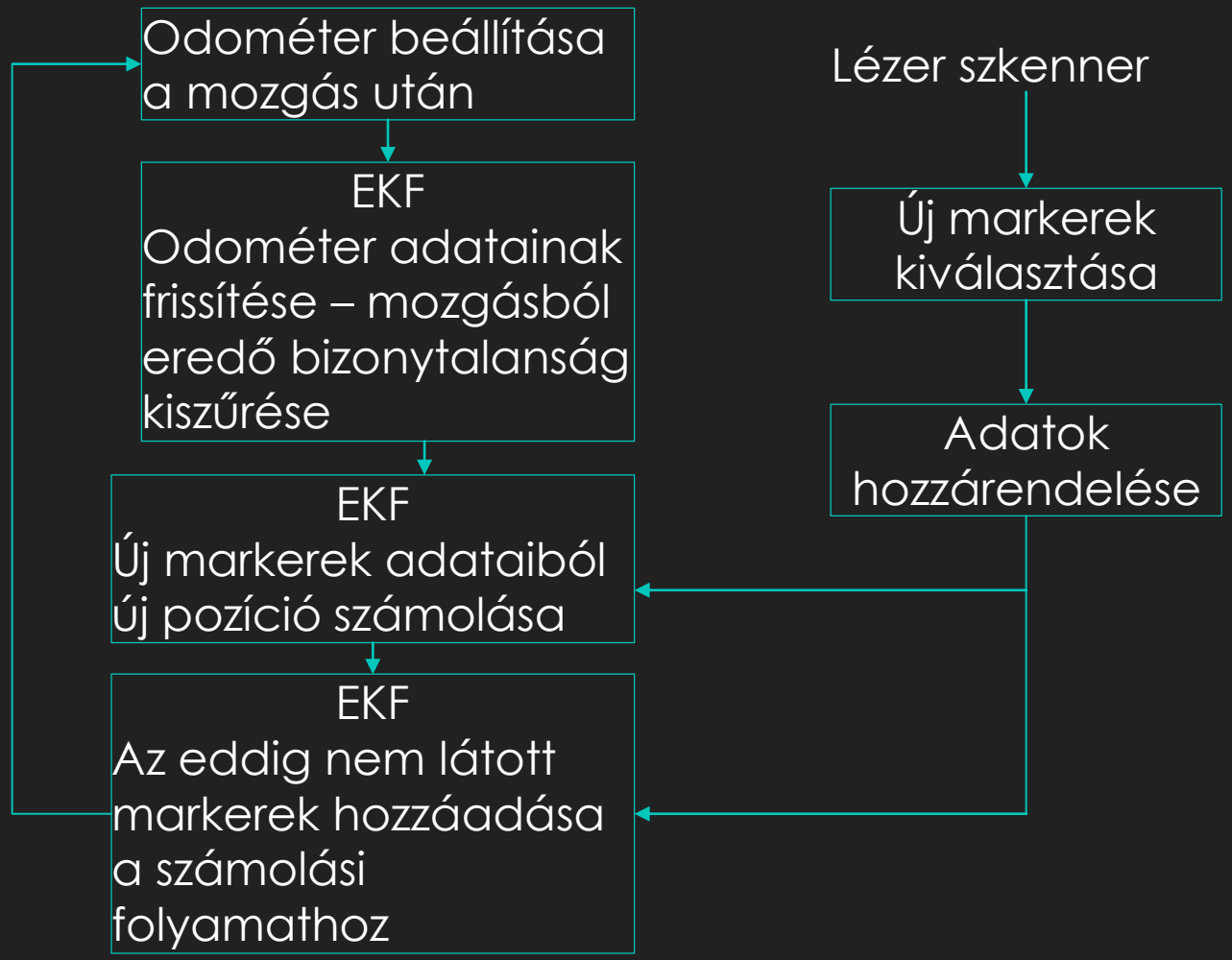

Topológiai-Gráf térkép




Hiba térkép

# SLAM

- Egyidejű lokalizálás és térképkészítés
- Folyamat:
  - Markerek kiválasztása/adatok hozzárendelése-értelmezése/pozíció meghatározása/→ új pozíció meghatározása → új markerek kiválasztása.
- Eszközök
  - Mobilrobot
  - Távolságmérő
    - Odométer (megtett út) / lézer / IR – (markerek távolsága)
- Módszerek
  - Kálmán filter / EKF



# Munkaterületek értelmezése

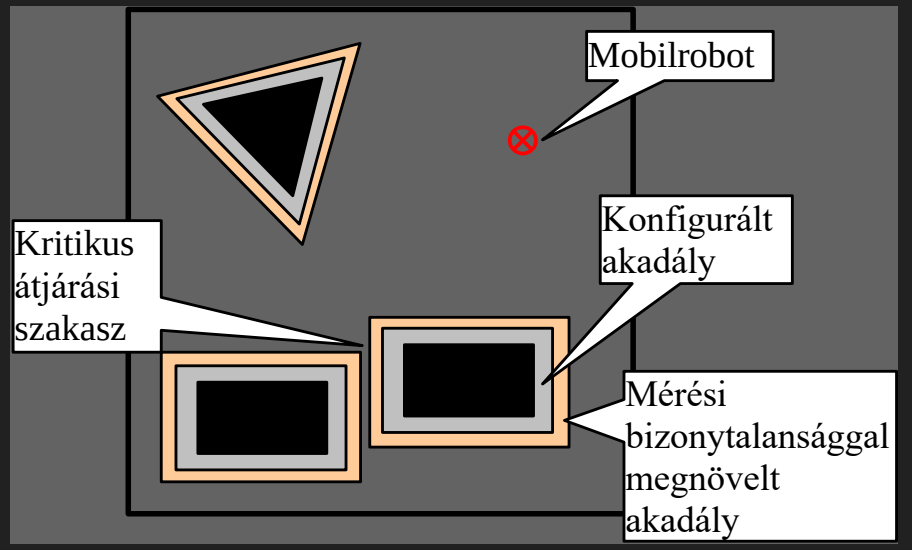
- Általános négyzethálós felosztás (pixel/voxel)
- Négyes fa (quadtree) felosztás
- Bináris szegmentáció
- Pontos felosztás

# Munkaterület felosztás-összefoglalás

| | *Jellemzők* | *Előny* | *Hátrány* |
|---|---|---|---|
| ***Négyzethálós*** | Egyforma négyzetek<br>Nagy memóriaigény | Egyszerű számolás | Sok számolás, mivel sok négyzet. Van átlapolás. |
| ***Négyes-fa*** | Nem-egyforma négyzetek | Kevesebb négyzet, kevesebb számolás | Ha az akadályok falai nem párhuzamosak a munkaterület határaival. Átlapolás. |
| ***Bináris szegmentáció*** | Nem feltétlenül egyforma négyzetek | Kevesebb négyzet, kevesebb számolás. Ha nem kellenek egyforma négyzetek → nincs átlapolás | Ha egyforma négyzetek → átlapolás. |
| ***Pontos*** | Teljesen különböző nagyságú szabad területek. | Nincs átlapolás | Nehezebb a felosztás. |

# Mobilrobot ábrázolása a munkaterületen

- Pontszerű ábrázolás alapjai
- Konfigurált akadályok, konfigurációs tér
- Hibatér



# Pályatervezési alapok

- **A pályatervező algoritmusoknak biztosítani kell:**
  - Robot és környezete szempontjából: a strukturáltságot (a környezet strukturális felépítése, a robot felépítése, alakja, képességei)
  - Az útvonal akadálymentességét
  - Komplexitást: az algoritmus konvergenciáját (vagyis, hogy véges időn belül megtalálja az utat, amennyiben az létezik)
  - Optimalitást: A feltételeknek megfelelően a legrövidebb, leggyorsabb és legbiztonságosabb út megtalálása.
  - Idő – tér komplexitást: olyan legyen a munkaterület felbontása, és a számolási algoritmus bonyolultsága, hogy a számolásokat végző processzor és környezete (háttértár) elegendő legyen a kiszámolásra.

# Lokális – Globális pályatervezés

- *A* pályatervező algoritmusokat általában ***két*** *nagy csoportra* oszthatjuk, de mindezek mellett találunk *egyéni* pályatervező eljárásokat is, melyeket általában a kifejlesztőkről (*Lee algoritmus*), vagy az algoritmus jellegzetességéről neveztek el:

- ***Lokális*** pályatervező algoritmusok: általában a relatív pozícionálásból, illetve az ultrahangos pozícionálásból (virtuális erők) kapott érzékelők adataiból kiindulva építi fel a bejárandó pályát a *START (S) és CÉL (C)* pozíciók között. Általában **ismeretlen** környezetben navigálunk, és a **lokális** pályatervező eljárások, általában csak a robothoz legközelebb álló akadállyal (pályatervezéssel) foglalkoznak ⇒ **lokális**.

- ***Globális*** pályatervező algoritmusok: Ezek az eljárások általában ismert területeken futnak. Számítási igényük nagy, és a számítások szempontjából két nagy csoportra oszthatók:

  - ***Off Line***: ami az előkészítő fázis, kiragadja a megfelelő adatokat az előre megadott munkaterület-térképből. Némely esetben tanuló *„learning"* fázisnak is emlegetik.

  - ***On Line***: amit az angol irodalom *„query"* vagy *„real-time"*, néven ismer. Ez a végrehajtó fázis, ami általában valamilyen gyors algoritmus az optimális (közel optimális) útvonal kiválasztására.

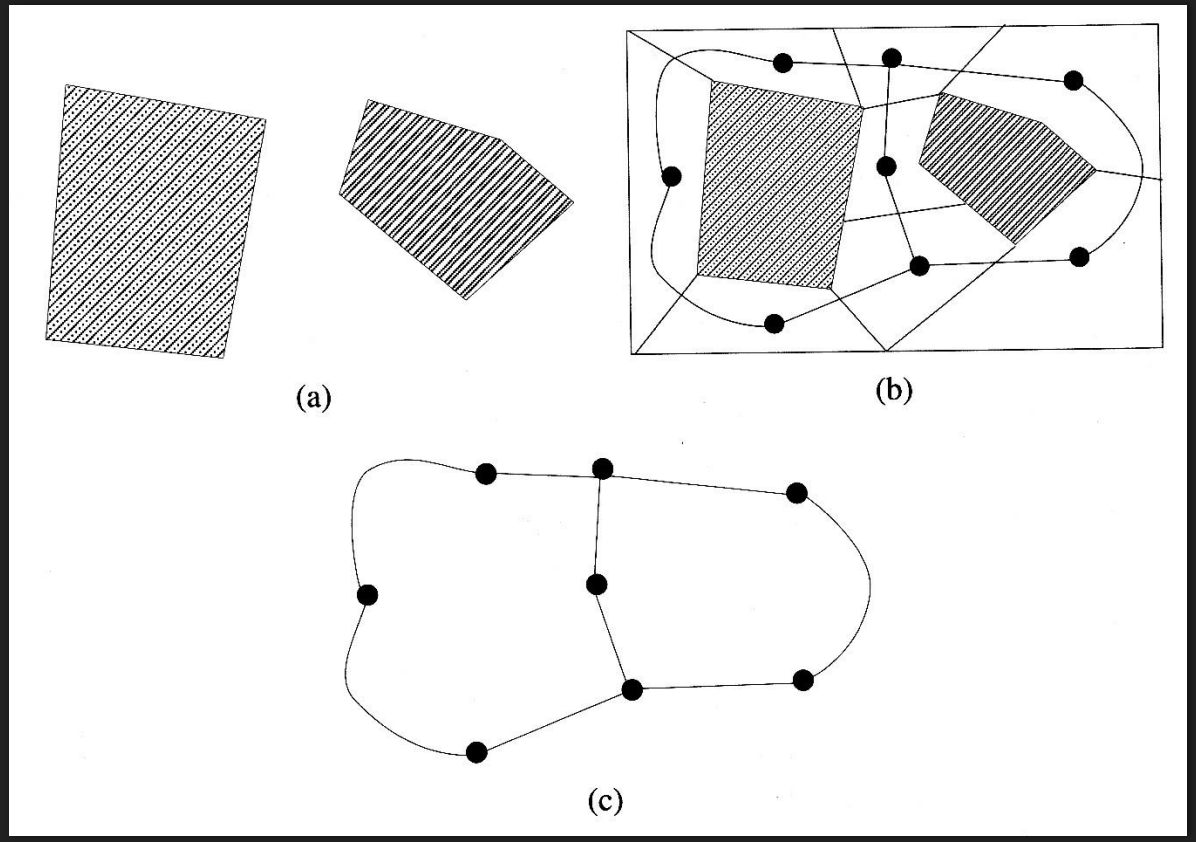
- Amennyiben ***kategorizálni*** szeretnénk a globális pályatervező eljárásokat, akkor három nagyobb kategóriát lehetne felállítani: az elsőbe sorolhatók a ***Reeds-Shepp*** féle, kibővített paraméterekkel dolgozó pályatervező eljárások, a másodikba a ***hullámterjedéses*** módszeren alapuló pályatervező eljárások, míg a harmadikba a ***véletlenszerű*** *(P-PPL)* pályatervező eljárások.

# Gráf alapján történő pályatervezés

1. Megalkotjuk a munkaterület *gráf-térképét* (beleépítjük a geometriai metrikus térképbe):

   - Az előzőekben tanult valamilyen módszerrel *felosztjuk* a munkaterületet: szabad tér – akadályok
   - Megkeressük mindegyik szabad szegmens középpontját ⇒ *csomópontok*
   - Összekötjük a csomópontokat ⇒ *gráf élek*
   - Kivesszük a térkép metrikus, geometriai részét, marad a tiszta gráf

2. Megszámozzuk a csomópontokat, amik valójában a szabad területek számozása, (a számozásban legyen valamilyen rendszer : balról jobbra növekedjen), Majd ez alapján a számozás alapján történik az útvonal felépítése az *S*-től *C*-ig.



# Láthatóság alapján történő útvonalkeresés (visibility graph)

*Ez* a módszer adja a legrövidebb utat az *S*-től a *C*-ig.

1.Amennyiben *S* és *C* között láthatóság van, akkor az útvonal az *S*-t és *C*-t összekötő egyenes

2.Amennyiben nincs láthatóság:

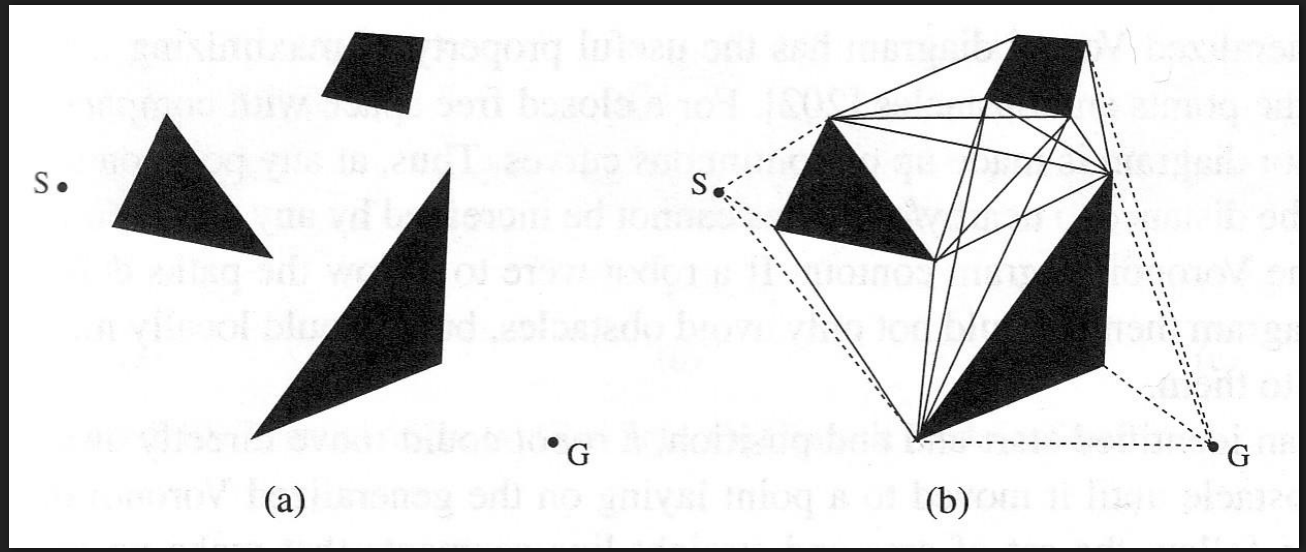
*A.* *S*-ből kiindulva megyünk (egyenest húzunk) a legközelebb eső akadály látható sarkaiig,

*B.* Majd a sarkakból újra vizsgáljuk a *C* láthatóságát.

1.Ha van: megyünk egyenesen *C*-be

2.Ha nincs: megyünk a következő legközelebbi akadály sarkaiig (egyenes vonalban)

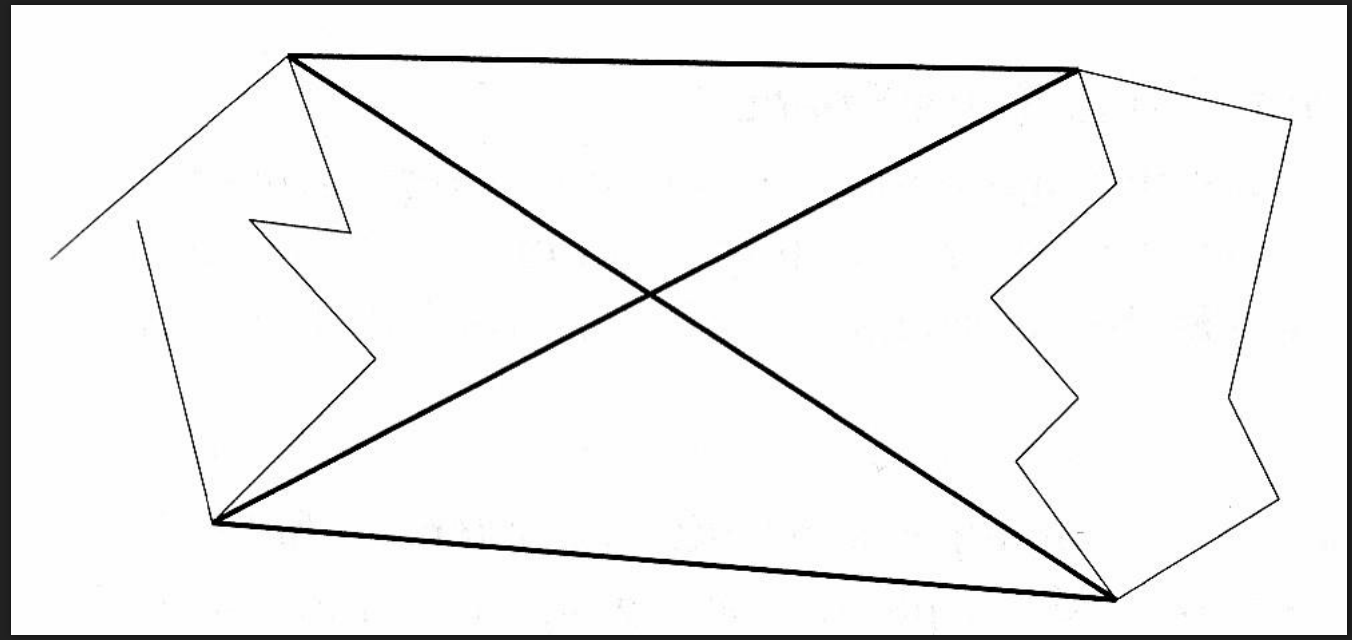
*C.* Újból vissza az *A* pontba addig míg *B.1* nem teljesül.



# Tangenciális gráf (tangent graph)

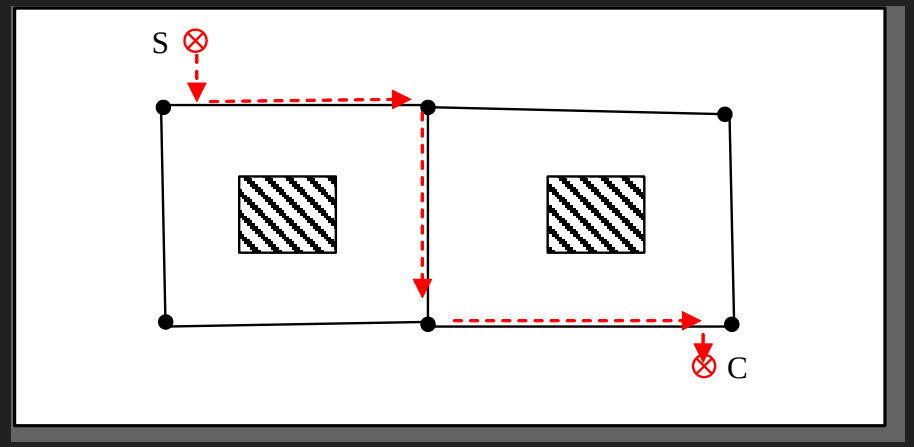
A láthatósági gráf effektívebb kihasználása. A láthatósági gráfnál az akadályok minden sarkát, minden sarokkal összekötjük, ami felesleges lehet az akadályok belső sarkainál.

Bebizonyított, hogy két akadály között a láthatóság 4 láthatósági vonallal (4 szélső sarok) meghatározható $\Rightarrow$ tangenciális egyenesek.



# Voronoi diagram

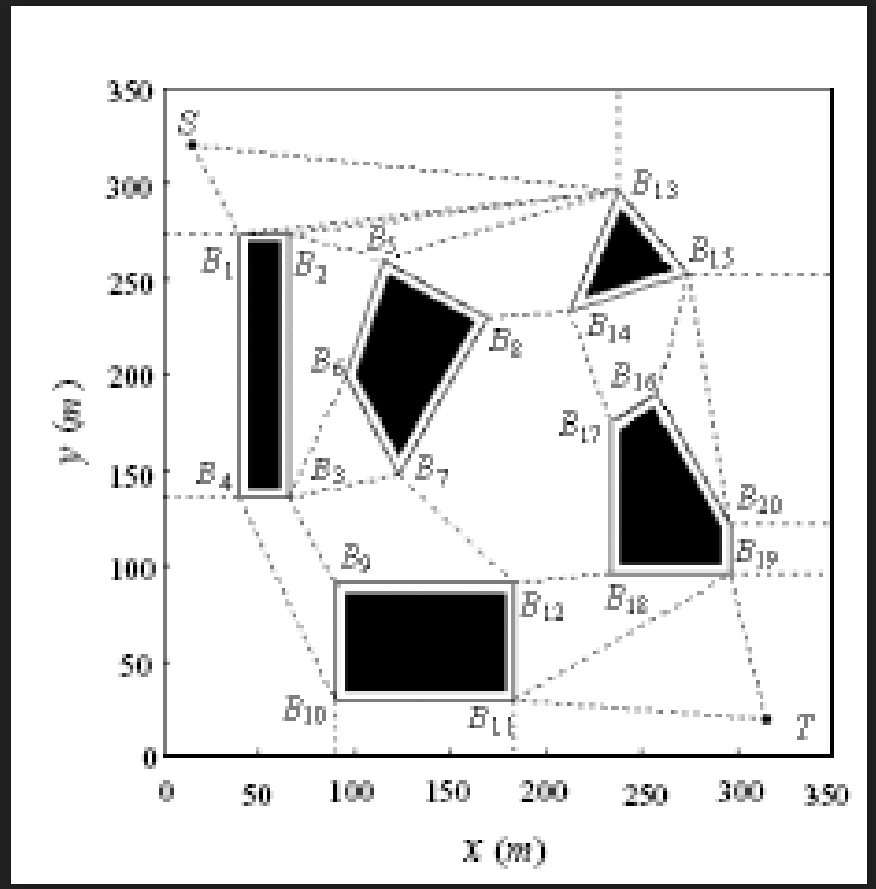
*Az* eddig ismert módszerek egy közös problémája, hogy a mobilrobot némely esetekben túl közel kerül az akadályokhoz. Ezt küszöböli ki a Voronoi-diagramm, (ami legjobban a gráf alapján történő útvonaltervezéshez hasonlít). *Az általános definíció* alapján a Voronoi- diagramm azon pontok halmaza, melyek egyforma távolságra vannak, a két egymáshoz legközelebb álló akadályoktól (beleértve a munkaterület határait is). Itt az akadályok, illetve akadály és munkaterület határa, közötti *középvonalakról* van szó. Ha az általános definíciót veszzük, akkor a diagramm matematikailag nehezen leírható görbékből tevődik össze. Ezt elkerülvén „egyszerűsítjük" a diagrammot, ahol lehet *linearizáljuk*, vagy *ívekre és parabolákra* próbáljuk illeszteni – ezeket az illesztéseket a későbbiekben szegmenseknek nevezzük, megszámozzuk, és nagy szerepük lesz az útvonal felépítésében.

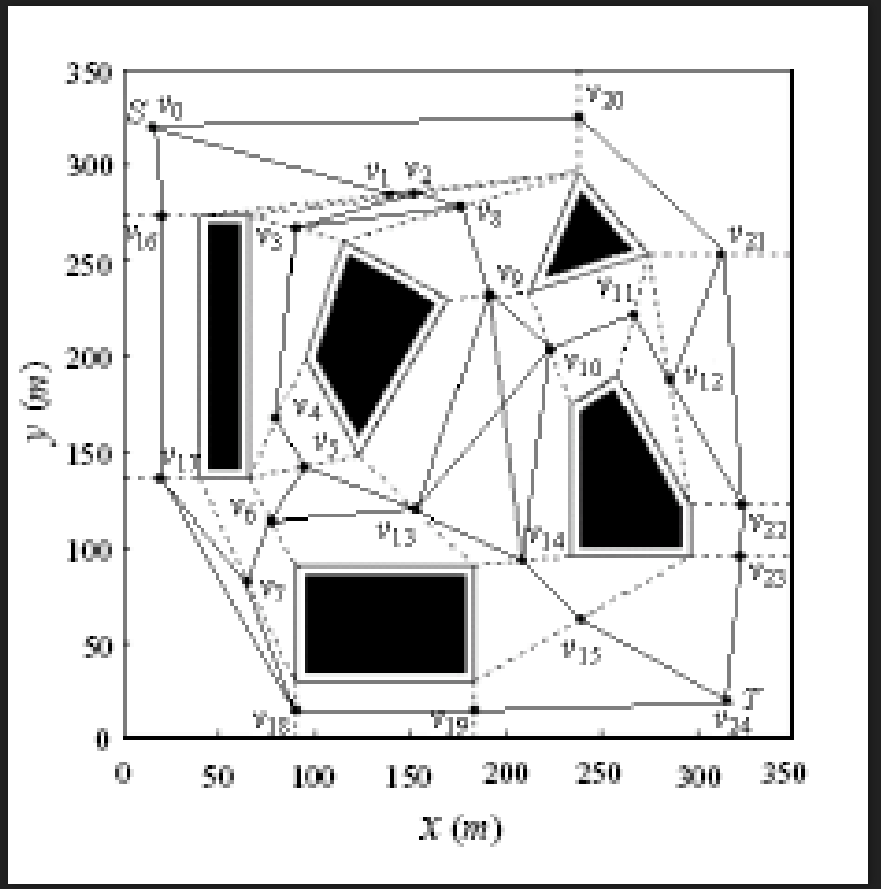



*Útvonal felépítése*: amennyiben adottak az *S* és *C* koordináták, akkor a mobilrobot első lépésként megkeresi az *S-hez* legközelebb eső Voronoi-diagramm szegmensét, majd a szegmenseken haladva eljut a *C-hez* legközelebb eső ponthoz, majd innét letérve a diagrammról eljut a *C-be*.

*Megjegyzés*: *A **biztonság*** szempontjából a Voronoi-diagramm adja a legbiztonságosabb utat: középvonalak = legtávolabb az akadályoktól.

# MAKLINK diagram



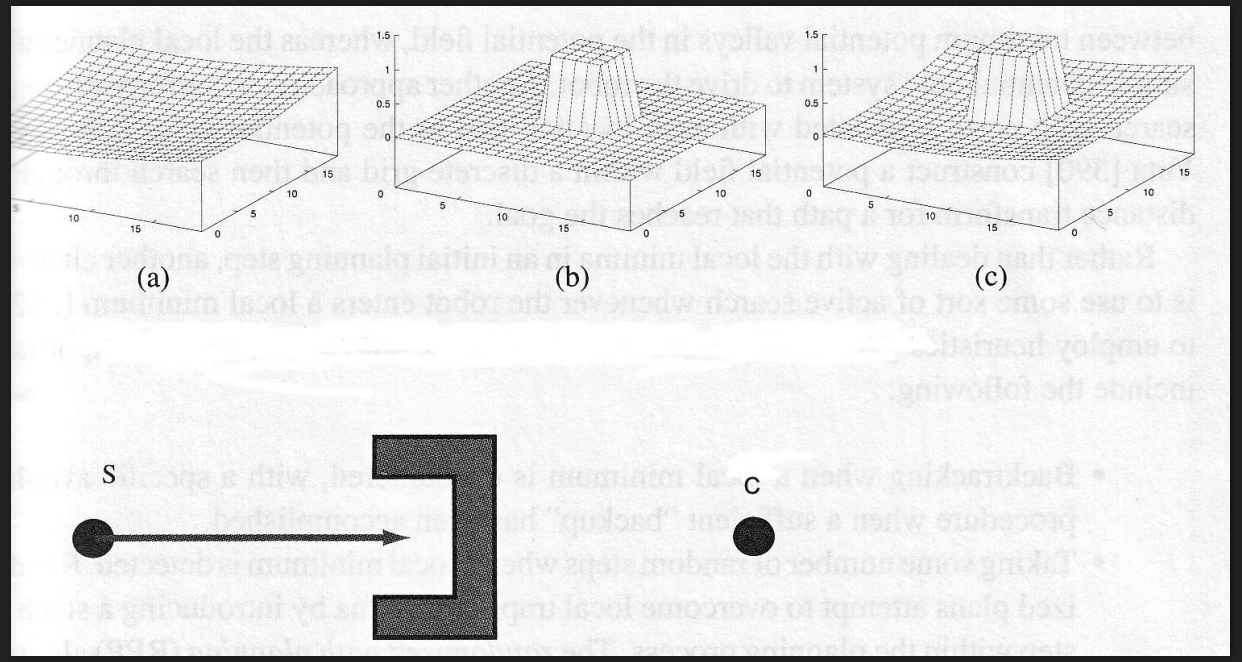



# Potenciálmezők (PF)

*Szintén,* többnyire lokális pályatervezésre használatos. Előnye, hogy dinamikus pályatervező eljárás = mozgó akadályok detekálása. Ismeretlen terület feltérképezésére is kitűnően használható.

*A* modellben az akadályokat fogjuk fel, mint feltöltött töltéseket, és a mobilrobot is legyen egy ugyanilyen potenciállal és pólussal feltöltött töltés ($\Rightarrow$ taszítják egymást). Ez a taszító (*repulzív*) erő akadályozza meg az ütközést. A repulzív erő nagysága, matematikailag, az adott pontban lévő potenciál negatív gradiensével egyenlő.

2D-s környezetben:



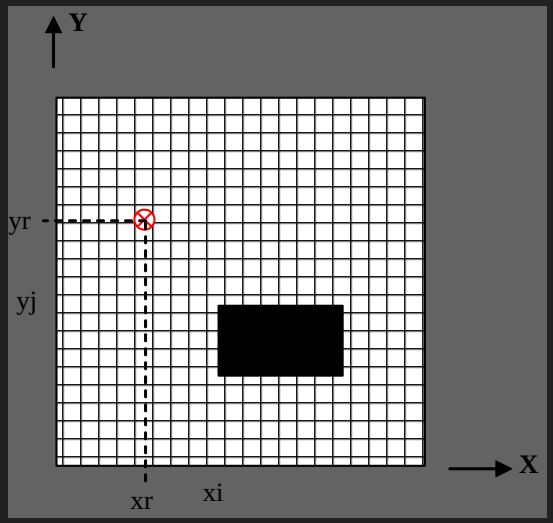

$$U(q) = U_{cél}(q) + \sum U_{akadály}(q);$$

# Virtuális erőtér (VFF)

A munkaterület négyzethálósan fel van osztva, és az egyes négyzetek „***erő-vektorai***" aszerint kerülnek kiszámításra, milyen messze vannak (vagy rajta vannak / félig, negyedig, …. / foglalva vannak) az akadályoktól.

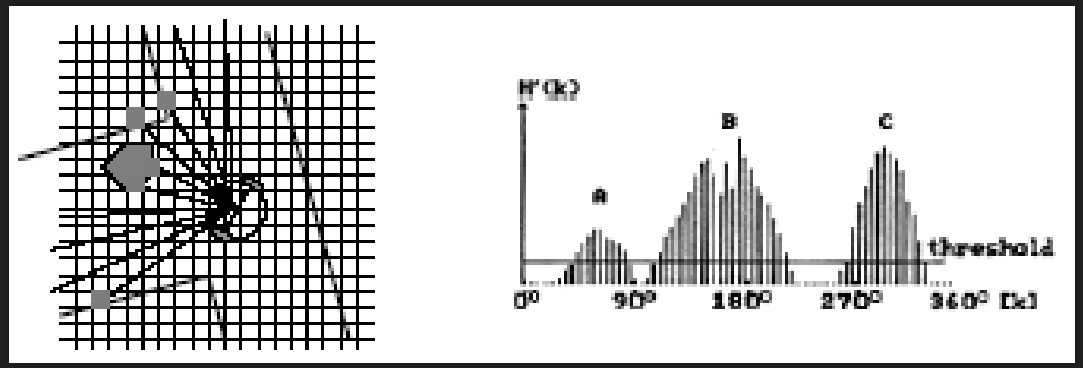
A négyzeteket reprezentáló ***erővektorok*** egyben megadják a vezérléshez szükséges paramétereket is (sebesség), ami hajtja tovább a cél felé a mobilrobotot. (Ebben hasonlít a potenciálmezőhöz.). A négyzethálós felosztás az erővektorokkal egy *vektortérre* szűkül. A fejlesztések az adaptív területfelosztások (BSP, Quadtree, …) felé irányulnak, hogy egyszerűbb legyen a vektortér. Ekkor viszont a nagyobb területeken nem mindegy milyen irányba indulunk, mert az akadály az egyik feléhez közelebb lehet (itt lassabban kell haladni), míg a másikhoz egész távol (ahol lehet gyorsan közlekedni).

Ennek kiküszöbölésére fejlesztették ki a ***VFH*** (Vector Field Histogram) eljárást.



# Vektortér hisztogram (VFH)

A mobilrobot körül egy *w* tartományban létrehozunk egy „ablakot”, melyet polárisan (sugárirányban: szög-szektorosan) még felosztunk. Ezzel pontosabban meg tudjuk határozni, hogy melyik irányban van az akadály, és ebből pedig azt, hogy merre induljunk tovább.




A mobilrobot ***haladási iránya*** a hisztogramnak megfelelően, a globális haladási iránynak megfelelően ⇒ egy *költségfüggvénynek* megfelelően kerül meghatározásra.

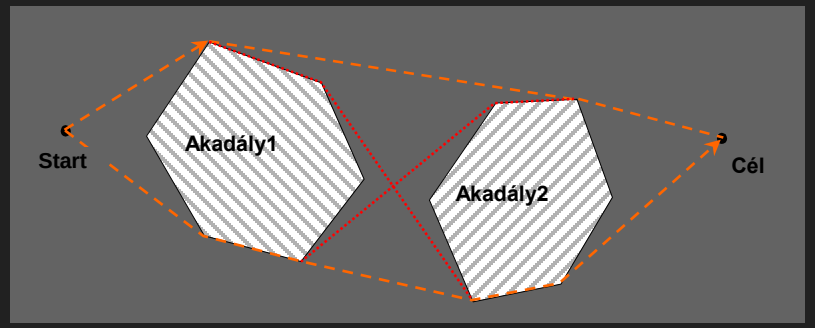
A költségfüggvény lehet:

*G = a*cél_irány + b*számolt_orientáció + c*előző_orientáció*

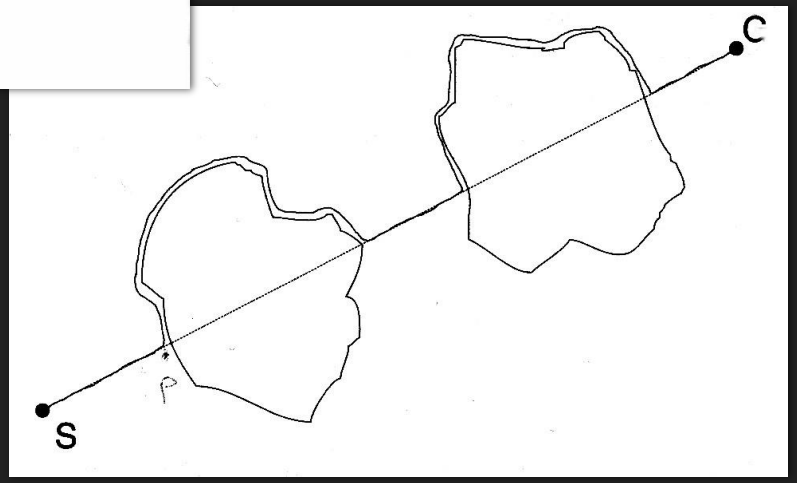
Ahol *a, b, c* = súlyozások, hogy melyik tényezőt milyen súlyozással vegyük figyelembe.

$$\beta_{i,j} = \arctan \frac{y_j - y_r}{x_i - x_r};$$

# BUG, BUG1, BUG2 algoritmusok




```
Visualize a direct path SG from the start S to the goal G
while the goal G is not achieved, do:
  begin
    while the path SG to the goal is not obstructed, do
      begin
        move towards the goal,
        if the path is obstructed then
          begin
            mark the current location as p
            circumnavigate the object until the robot eit
            (a) hits the line SG again and can move towar
             in which case you do so;
            (b) returns to where p
             in which case G is unreachable.
          end
      end
  end
```



# Reeds-Shepp algoritmusok (Geometriai primitívákon alapuló)

*Geometriai primitívákon* alapuló pályatervező algoritmusok. Ebben a bekezdésben a geometriai primitívák alatt *szakaszok* (S), *ívek* (C) és *megfordulást* jelentő szimbólumok (|) értendők. A megfordulás alatt irányváltoztatás értendő, az előre → hátra, illetve hátra → előre között. Az ilyen típusú pályatervező algoritmusok általában feltételeznek egy bizonyos mobilrobot-kinematikát (általában kerekes járművekről van szó, de még ezen belül is az egyes algoritmusok pontosításra szorulnak). Az egyik legismertebb kifejlesztői és képviselői az ilyen típusú algoritmusoknak, *Reeds és Shepp.*

*Reeds és Shepp* megállapította, hogy ha figyelembe veszzük a geometriai- és mozgás-primitívákat is, akkor 48 különböző variációval bármilyen pályát le tudunk írni.

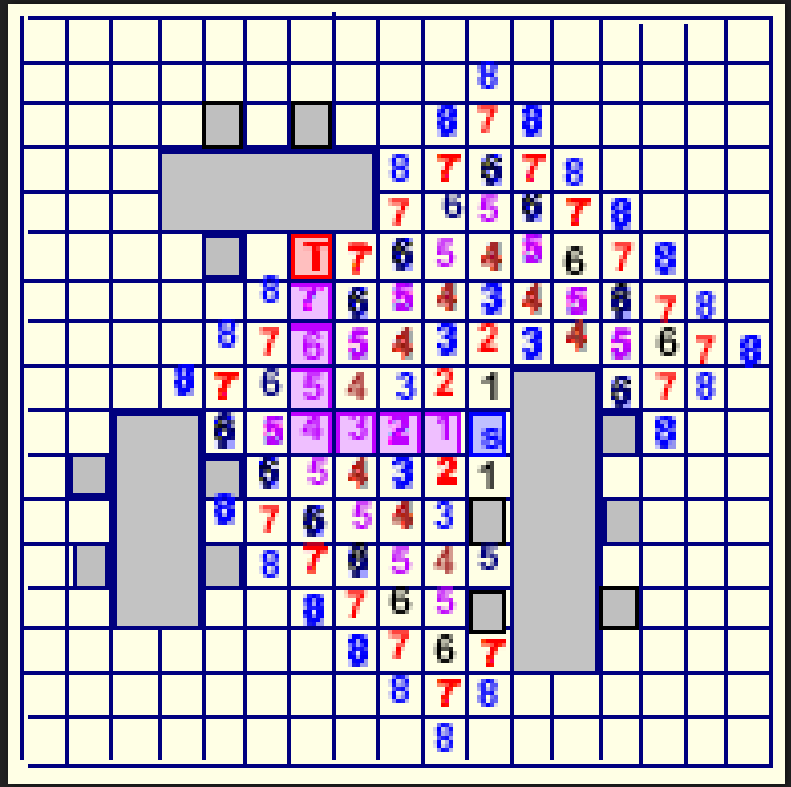
| variációk | α | β | γ | d |
|---|---|---|---|---|
| $C_\alpha\|C_\beta\|C_\gamma$ | [0,π] | [0,π] | [0,π] | - |
| $C_\alpha\|C_\beta C_\gamma$ | [0,β] | [0,π/2] | [0,β] | - |
| $C_\alpha C_\beta\|C_\gamma$ | [0,β] | [0,π/2] | [0,β] | - |
| $C_\alpha S_d\|C_\gamma$ | [0,π/2] | - | [0,π/2] | (0,∞) |
| $C_\alpha\ C_\beta\|C_\beta C_\gamma$ | [0,β] | [0,π/2] | [0,β] | - |
| $C_\alpha\ \|C_\beta C_\beta\|C_\gamma$ | [0,β] | [0,π/2] | [0,β] | - |
| $C_\alpha\ \|C_{\pi/2} S_d\ C_{\pi/2}\|C_\gamma$ | [0,π/2] | - | [0,π/2] | (0,∞) |
| $C_\alpha\ \|C_{\pi/2} S_d\ C_\gamma$ | [0,π/2] | - | [0,π/2] | (0,∞) |
| $C_\alpha\ S_d\ C_{\pi/2}\|\ C_\gamma$ | [0,π/2] | - | [0,π/2] | (0,∞) |

# Globális pályatervező eljárások

A globális pályatervező eljárásoknál az útvonal általában ismert környezetben (ahol ismert a munkaterület térképe: topológiai/metrikus) kerül meghatározásra. Az ilyen eljárások számítási igénye jóval nagyobb, mint a lokális eljárásoké. Az eljárások a számítás szempontjából két fázisra oszthatók, egy „*off-line*" (ami az előkészítő fázis, kiragadja a megfelelő adatokat az előre megadott munkaterület-térképből. Némely esetben tanuló „*learning*" fázisnak is emlegetik.) és egy „*on-line*" (némely esetben „*query*" vagy „*real-time*"), végrehajtó fázis, ami általában valamilyen gyors algoritmus az optimális (közel optimális) útvonal kiválasztására. Amennyiben kategorizálni szeretnénk a globális pályatervező eljárásokat, akkor három nagyobb kategóriát lehetne felállítani: az elsőbe sorolhatók a ***Reeds-Shepp*** féle, kibővített paraméterekkel dolgozó pályatervező eljárások, a másodikba a ***hullámterjedéses*** módszeren alapuló pályatervező eljárások, míg a harmadikba a ***véletlenszerű*** *(P-PPL)* pályatervező eljárások.
Az *első kategória* abban tér el a klasszikus Reeds-Shepp metódustól, hogy ebben az esetben az algoritmusnak figyelembe kell vennie a munkaterületen lévő akadályokat. A metódusok általában valamilyen munkaterületi felosztással kezdődnek, ahol először, a bejárhatóság szempontjából, megállapításra kerülnek az akadályok és a szabad területek. Az akadályok általában valamilyen konvex *poligon* formájában vannak képviselve a munkaterületen. Ezek után az „off-line" fázisban egy *gráf-rendszer* kerül megvalósításra, -az előző munkaterület-térkép alapján-, amely magába foglalja a szabad területeket (lehetséges útvonalakat) függetlenül a kiindulási- és cél-koordinátáktól. Itt a gráf-rendszer élei és csomópontjai követik a *konfigurált akadályok* határvonalait, illetve sarokpontjait, vagyis a végleges út szempontjából az útvonal falkövetéses, majd a végén célkövetéses lesz. Ebben a fázisban valójában még nincs is megállapítva kiindulási-, ill. cél-koordináta. A második fázis, a megvalósítási „*Real-time*" fázis, ahol az előző fázisban felépített adat-struktúrák alapján, általában három lépésben, megtervezésre kerül a végrehajtási útvonal. Az *első lépésben* a kiindulási- és cél-koordináták kerülnek meghatározásra. Ebben a lépésben, a munkaterületnek megfelelően, több lehetséges útvonal is kialakulhat a koordináták között. A *második lépés* általában valamilyen gráf-keresési módszer (pl.: A* algoritmus), amely megkeresi a két koordináta között lévő „legrövidebb" (vagy legkisebb számítási költséggel járó) utat. A *harmadik lépés*, pedig a kiválasztott útvonalra megtervezi a bejárandó, végleges végrehajtási útvonalat, ahol az algoritmusok nagy része az ívekből-szakaszokból összetevődő *Reeds-Shepp* metódust használja.

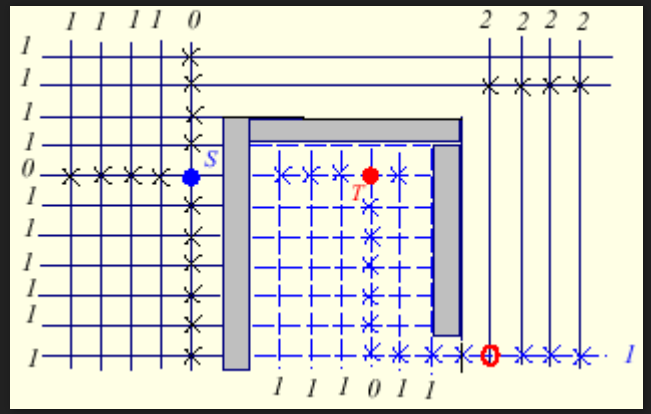
# Hullámterjedéses módszerek1

*A hullámterjedéses* módszerek szintén a munkaterület felosztásával kezdődnek. Alap kivitelben, itt a munkaterület négyzethálósan van felosztva, így az akadályok is négyszögekkel vannak képviselve a munkaterületen. A továbbiakban a négyzetek számozva vannak, kiindulva (legkisebb számozás) az indulási koordinátától (*S*), -körbe, mint ahogy a hullám terjed-, egészen a cél-koordinátáig (*C*). Az útvonal (keresés) a számozások alapján alakul ki (a cél felé tartva, mindig az eggyel nagyobb számozású négyzet a következő).



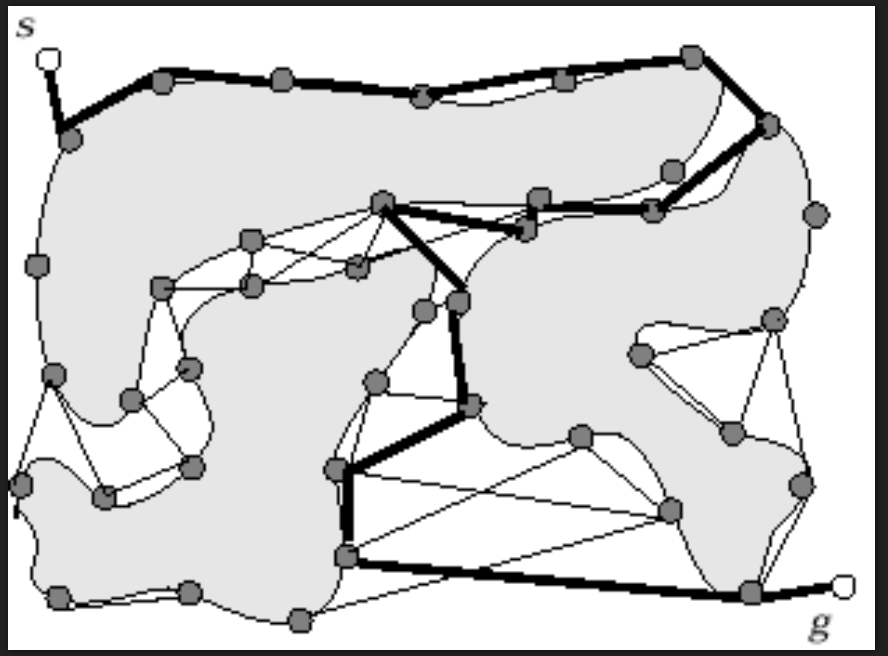
# Hullámterjedéses módszerek 2

A hatvanas évek végén (1968/69) a négyzetrács-szerkezetű területfelosztást, *vízszintes és függőleges vonalak hálója* váltotta fel. Ezekben az esetekben a kiindulás és cél koordinátákat jelölő vonalak voltak a „0" számmal jelölve, majd távolodva tőlük emelkedtek a számok is. Itt is külön hullámot indítottak a kiindulási és cél koordinátákból, majd a számozás és vonalak metszéspontjai alapján került megállapításra az útvonal.



# Véletlenszerű pályatervező algoritmusok

*A véletlenszerű* pályatervező algoritmusok hasonlítanak az első kategóriába sorolt, globális *Reeds-Shepp* metódushoz. Az előzőekben említettekkel összehasonlítva, talán ez a legáltalánosabb, de mindenesetre a legújabb metódus. Itt mindenképpen meg kell említeni *Lydia Kavraki és J-C. Latombe* munkáit, akik talán elsőként dolgozták ki és alkalmazták ezt a módszert.
Ezek az algoritmusok szintén két részre oszthatók, egy számításigényes *tanulási (learning)* és egy gyors *végrehajtó (query)* fázisra.
A *tanulási* rész a munkaterület felosztásával kezdődik, szabad területekre, illetve akadályokra. A továbbiakban azonban már eltér az előzőektől, itt ugyanis a szabad területeken belül, *véletlenszerűen*, *konfigurációs pontok* lesznek elhelyezve, majd a szomszédos konfigurációs pontok összekötésre kerülnek. Ezekből a konfigurációs pontokból és az összekötésekből kialakul egy gráf-rendszer (nem irányított). Ennél a metódusnál, általában már a gráf-rendszer elkészítésénél több szempontot is figyelembe lehet venni, mint pl.: csak a szabad területen belül szabad összekötni a konfigurációs pontokat, és már az összekötésnél figyelembe kell venni a robot kinematikai tulajdonságait (kanyarodási képesség). Minél több szempontot veszünk figyelembe a tanulási fázisban, annál gyorsabb lesz a végrehajtási fázisunk.
A *végrehajtási* fázisban a kiindulási és cél koordináták meghatározása után, elindul egy rövid keresés, ami ezeket a koordinátákat *„rávetíti"* a megfelelő (ami általában a legközelebbi) gráf összetevőkre (általában a ponthoz legközelebb eső él). Ezek után egy gráf-keresési algoritmus (A*, Dijsktra) megkeresi az összeköttetést a két pont között, majd a végleges útvonal illesztése következik. A megtalált összeköttetés nem minden esetben kell, hogy a legmegfelelőbb útvonal is legyen, ezért ezek a továbbiakban még különböző szempontok alapján még optimalizálhatóak.



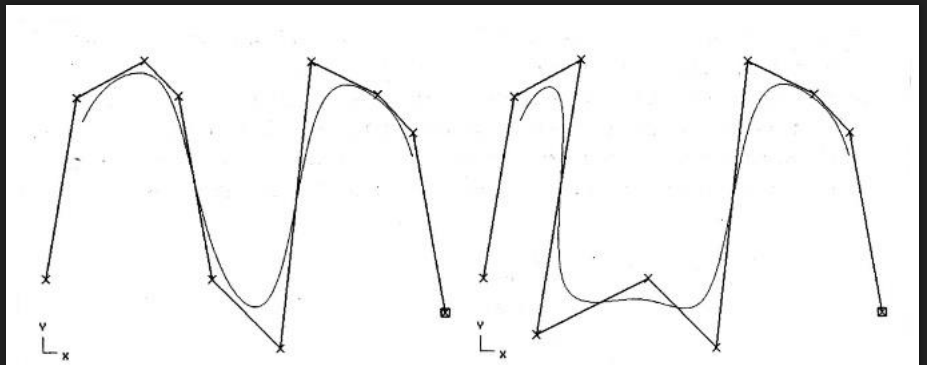
# Lágygörbéken alapuló pályatervezés

**Ez a pályatervezés a végrehajtó (query) részre vonatkozik.**

Ez a ***legújabb*** pályatervező algoritmusok közé tartozik. Általában a közel ***idő-optimális*** pályatervezést valamilyen lágygörbe segítségével érik el. A lágygörbék adják a legfolytonosabb utat, kizárják a hirtelen fékezéseket, gyorsulásokat.

Legismertebb görbék a pályatervezéshez:

- *Lagrange* interpolációs görbék: A görbe *átmegy* a vezérlőpontokon. Sajnos hajlamos az oszcillációra. Egy vezérlőpont koordinátájának megváltoztatása, sajnos az egész görbére hat → globálisan megváltozik az egész görbe alakja.
- *Bézier* approximációs görbék: A görbe nem megy át a vezérlőpontokon (csak az elsőn és az utolsón), de a vezérlőpontok konvex burkában marad.
- *Összetett* görbék (több görbeszegmensből építkező görbék) – SPLINE -ok: fontos az illeszkedési pontok folytonossága. A robotirányítás szempontjából legideálisabb a $C^3$ *(létezik a harmadik deriváltja)* folytonosságú, de már a $C^2$ folytonosságú is elégséges lehet. Előny: a *B-Spline* -ok ***lokálisan*** vezérelhetők ⇒ egy vezérlőpont megváltoztatása, csak a közvetlen környezetére hat.



*B-spline lokális vezérelhetősége: a 4. vezérlőpont áthelyezése csak a görbe első részét módosítja*

$$B_0(t) = \frac{(1-t)^3}{6};$$

$$B_1(t) = \frac{1+3(1-t)+3t(1-t)^2}{6};$$

$$B_2(t) = \frac{1+3t+3(1-t)t^2}{6};$$

$$B_3(t) = \frac{t^3}{6};$$