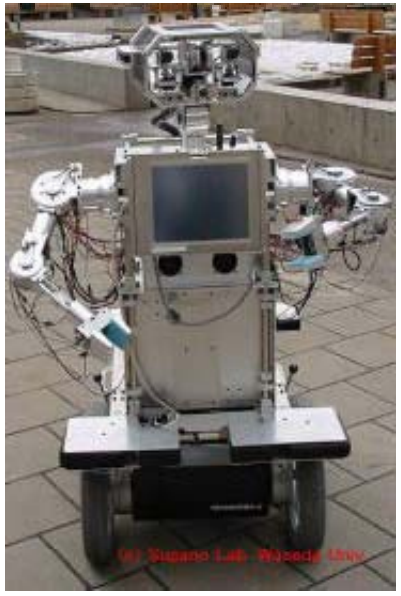
Lekcija 13: *Algoritmi izbjegavanja prepreka i pretraživanja grafova*

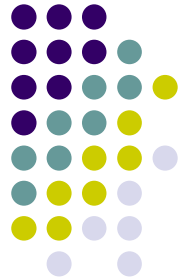


Prof.dr.sc. Jasmin Velagić
Elektrotehnički fakultet Sarajevo

Kolegij: Mobilna robotika

2012/2013

13.1. Algoritmi izbjegavanja prepreka



- U mnogim slučajevima, globalna mapa prostora nije dostupna u trenutku kada robot započinje svoje kretanje prema cilju.
- Planiranje zasnovano na metodi potencijalnih polja je primjer lokalnog postupka planiranja kretanja koji se može koristiti u ovoj situaciji.
- Nažalost, lokalni planeri na temelju potencijalnih polja ne mogu garantirati pronalaženje putanje do cilja.
- Moguće je, uz određene okolnosti, razviti metode planiranja kretanja koje osiguravaju dolazak robota u ciljnu tačku uz prisustvo neizvjesnosti, odnosno uz ograničeno poznavanje prostora.
- Ovi algoritmi su poznati i pod imenom ***hibridni algoritmi*** planiranja putanje.

13.1.1. Bug algoritam

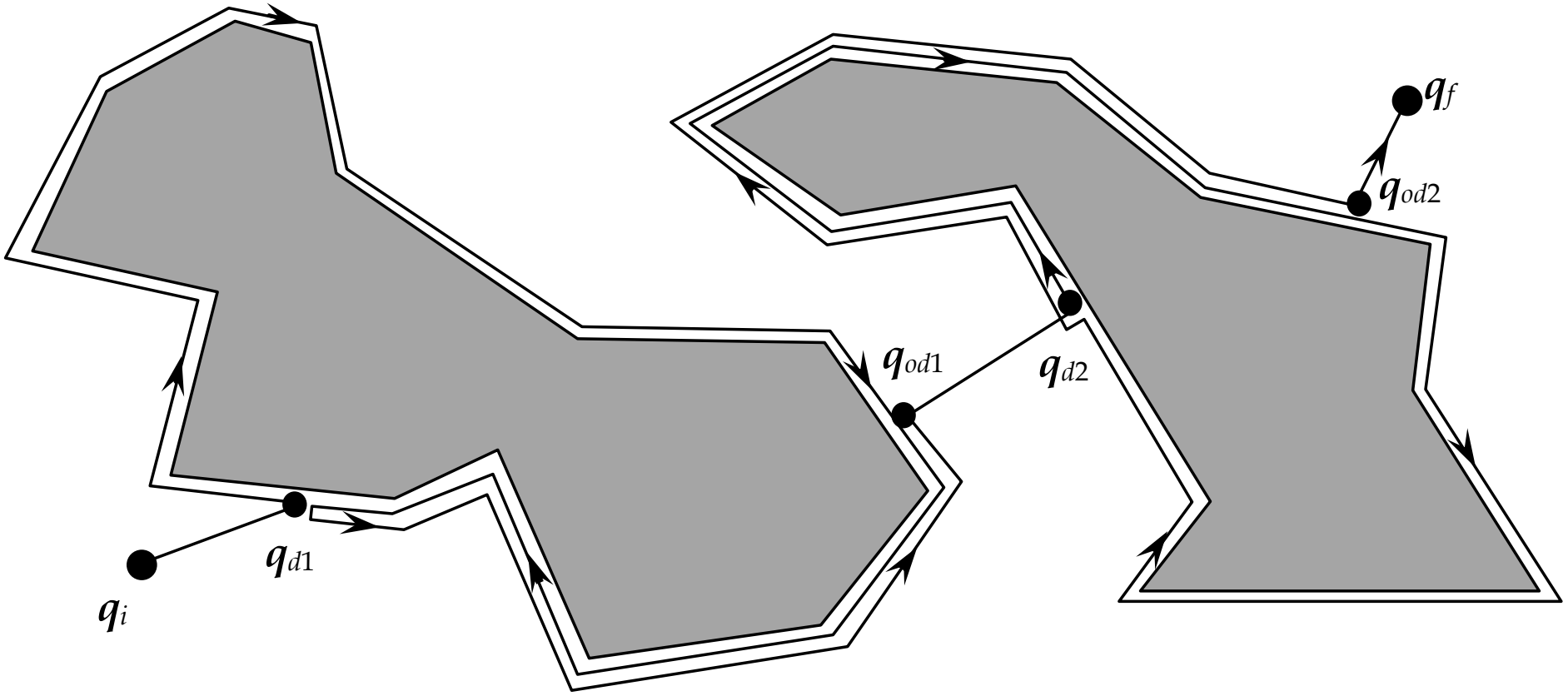
- Najpoznatiji iz grupe hibridnih algoritama je ***Bug algoritam***.
- Bug algoritam (Lumelsky i Stepanov, 1988; Lumelsky i Skewis, 1990) je jedan od najjednostavnijih **algoritama izbjegavanja prepreka** robota tokom kretanja ka ciljnoj tački.
- Osnovna ideja se sastoji u **slijeđenju kontura svake prepreke koja se nađe na putu kretanja robota i obilaska oko prepreke**.
- Ovaj algoritam se koristi za planiranje putanje od početne do ciljne tačke (njihove koordinate poznate) ako je **poznat smjer kretanja prema cilju**, **odometrija robota perfektna, idealan kontaktni senzor, beskonačna memorija za pohranu informacija**.

Bug algoritam

- Pri tome se pretpostavlja razumno okruženje koje sadrži konačan broj prepreka i postojanje linija koje će presjecati prepreku konačan broj puta.
- Najviše su u upotrebi dvije verzije Bug algoritma: **Bug1** i **Bug2**.
- Kod **Bug1** algoritma robot prvo **u cijelosti obilazi oko prepreke, zatim se počinje udaljavati od prepreke iz tačke koja je najbliža ciljnoj tački** (slika na sljedećem slajdu).
- **Ovaj pristup je veoma neefikasan, ali garantira robotu dostizanje bilo kojeg ostvarivog cilja.**

Bug algoritam

- Bug 1 algoritam

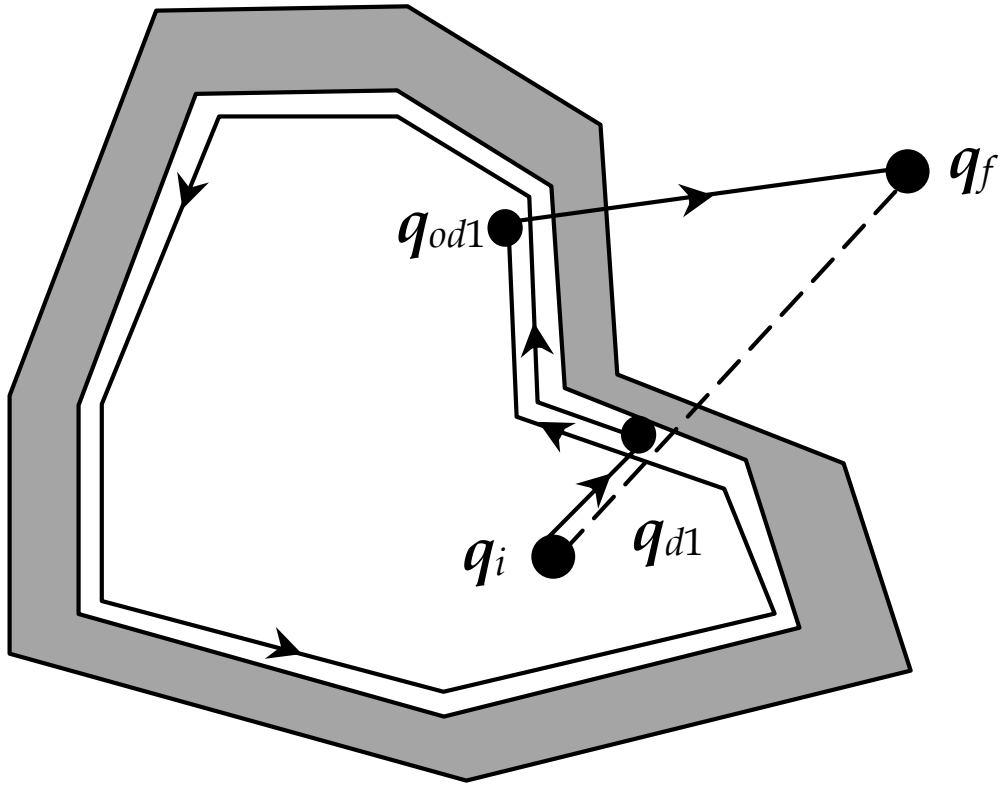


q_{od} –tačka odvajanja od prepreke

q_d – tačka prvog dodira sa preprekom

Bug algoritam

- Bug 1 algoritam
- Postoje situacije, odnosno okoline, u kojima primjenom Bug1 algoritma nije moguće dosegnuti, odnosno pristići u ciljnu konfiguraciju.



Bug algoritam

- Bug 1 algoritam – pseudo kod

Algoritam: Bug1

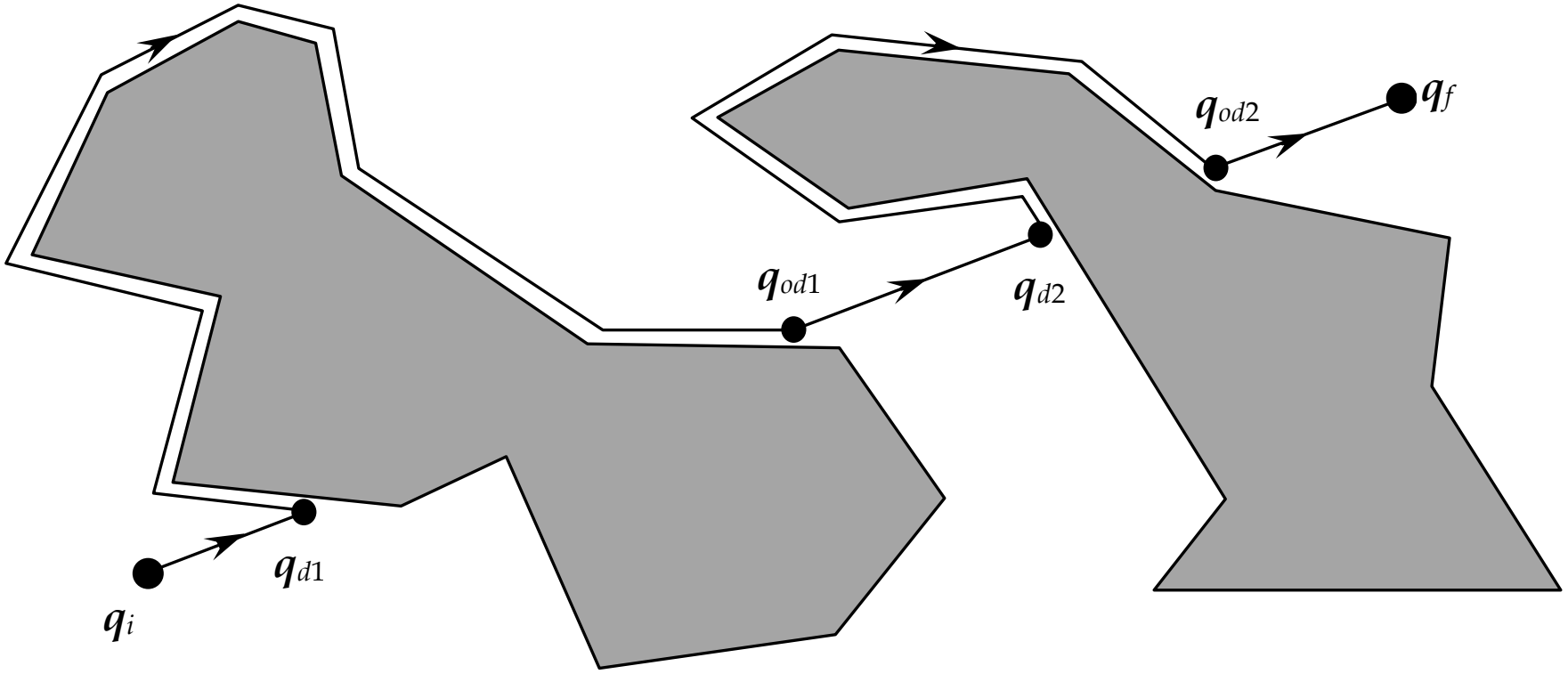
1. $q_i = q_{od_{i-1}}$ ← početna tačka;
2. q_f ← ciljna tačka;
3. **petlja**
4. **ponavljaj**
5. kretanje od $q_{od_{i-1}}$ prema q_f
6. **dok** q_f nije dosegnut ili robot pristigao u q_{d_i}
7. **ako** cilj (q_f) je dosegnut **onda**
8. izlaz
9. **kraj**
10. **ponavljaj**
11. slijeđenje granice prepreke
12. **dok** q_f nije dosegnut ili robot ponovo pristigao u q_{d_i}
13. odrediti q_{od_i} na perimetru koji je najbliža cilju
14. idi do q_{od_i}
15. **ako** se robot ranije kretao ovom stazom ka cilju **onda**
16. q_f nije dohvatljiv i izlaz
17. **kraj**
18. **kraj petlje**

Bug algoritam

- Sa **Bug2** algoritmom robot **započinje slijediti konturu prepreke, ali se počinje od nje udaljavati trenutno kada osjeti da se može pravolinijski kretati ka cilju po pravcu *m* koji spaja početnu i ciljnu tačku**.
- Na ovaj način se općenito poboljšava Bug algoritam kojim se sada značajno skraćuju ukupno kretanje robot do cilja, što se vidi sa slike na sljedećem slajdu.
- Međutim, ostaje i dalje donekle **izražen problem neefikasnosti, posebno u pogledu optimalnosti putanje**.

Bug algoritam

- Bug2 algoritam



Bug algoritam

- Bug2 algoritam – pseudo kod

Algoritam: Bug2

1. $q_i = q_{od_{i-1}}$ ← početna tačka;
2. q_f ← ciljna tačka;
3. **dok** je istina **izvršavaj**
4. **ponavljaj**
5. kretanje od $q_{od_{i-1}}$ prema q_f
6. **dok** q_f nije dosegnut ili robot nije pristigao u tačku dodira q_{d_i}
7. skrenuti desno (ili lijevo)
8. **ponavljaj**
9. slijeđenje granice prepreke
10. **dok**
11. cilj (q_f) je dosegnut **ili**
12. robot ponovo pristigao u q_{d_i} **ili**
13. robot ponovo pristigao na pravac m u tački m tako da
14. $m \neq q_{d_i}$ (robot nije dosegao tačku dodira)
15. $d(m, q_f) < d(m, q_{d_i})$ (robot blizu cilja) i
16. **ako** robot se kreće prema cilju, tada on neće udariti u prepreku
17. postaviti $q_{od_{i+1}} = m$
18. povećaj i
19. **kraj petlje**

Bug algoritam

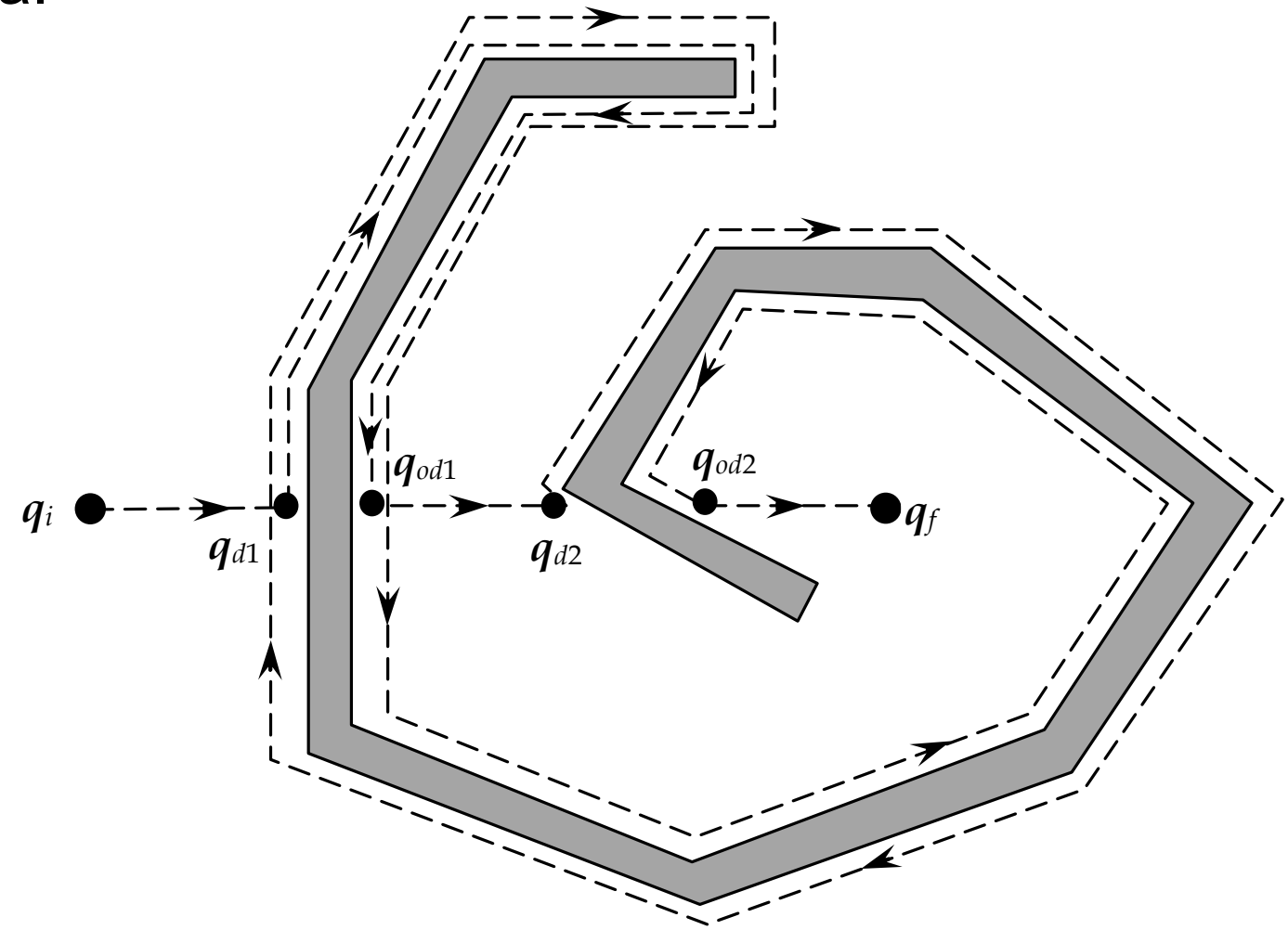
- Iz navedenog se može zaključiti da je Bug2 efikasniji od Bug1 algoritma, međutim, nije uvijek tako.
- Metrika – duljina:

$$L_{Bug1} \leq d(q_i, q_f) + \frac{1}{2}\sum_{i=1}^{n} p_i$$

$$L_{Bug2} \leq d(q_i, q_f) + \frac{1}{2}\sum_{i=1}^{n} n_i p_i$$

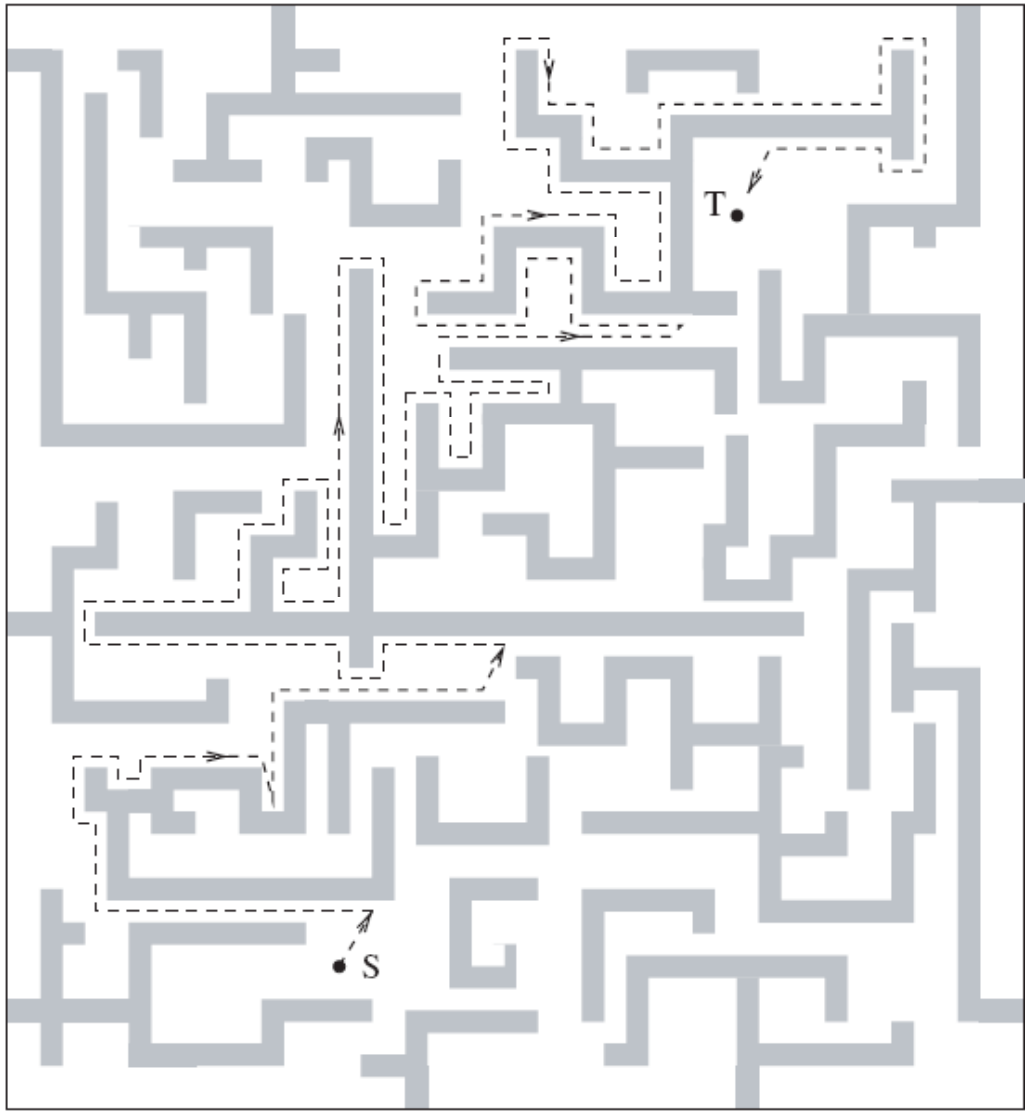
Bug algoritam

- Duljina pređenog puta može biti značajno dulja ako pravac *m* presjeca granice prepreka više puta.

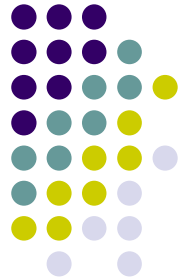


Bug algoritam

- Planiranje kretanja u složenoj okolini – Bug2



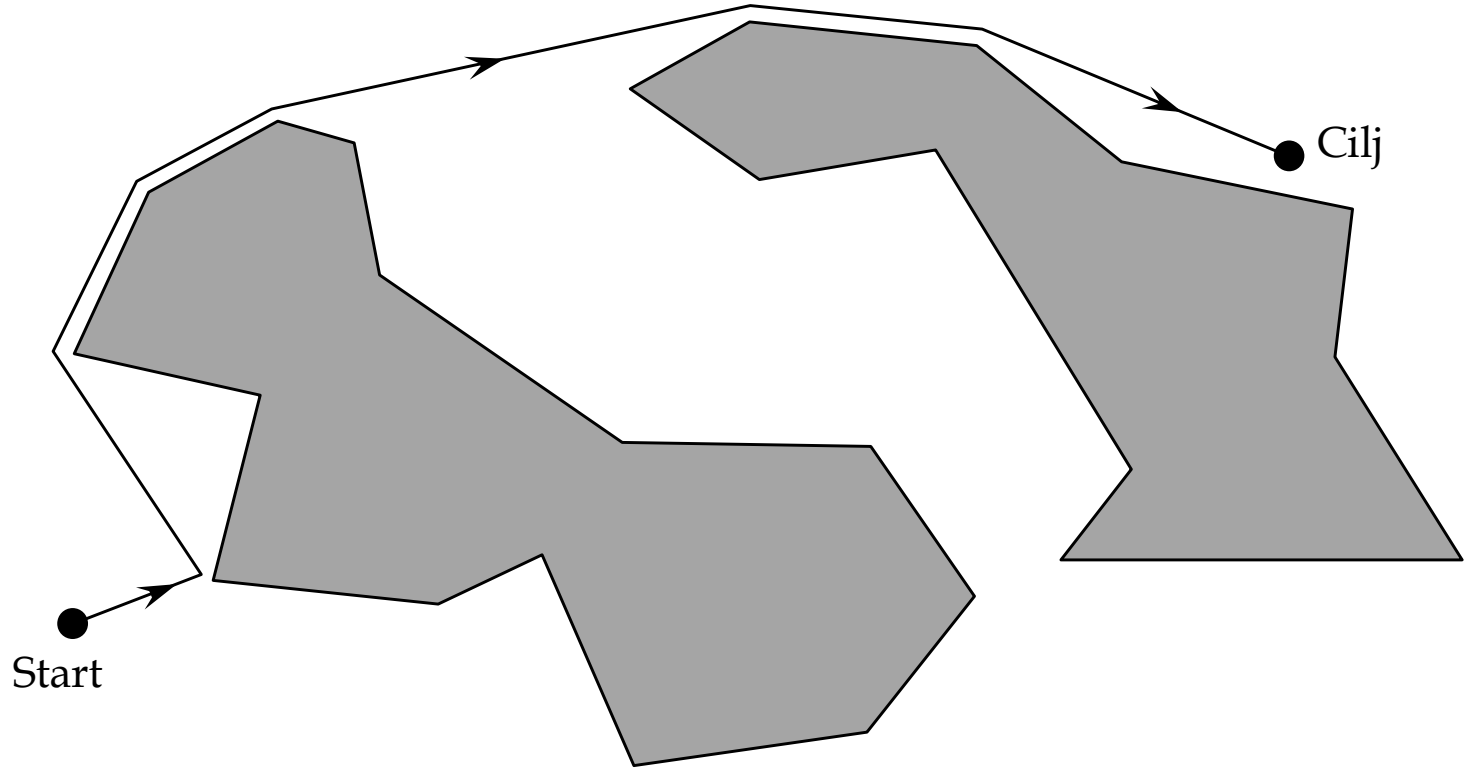
Bug algoritam



- Postoje brojne varijacije i proširenja Bug algoritma.
- Jedan od algoritama kojim se znatno poboljšava optimalnost Bug2 algoritma je ***tangecijalni Bug*** algoritam (Kamon i saradnici, 1996), koji dodaje opažanje okoline na temelju mjerenja senzora udaljenosti (infracrveni, ultrazvučni senzori) i lokalni prikaz okoline, poznat pod imenom ***lokalni tangencijalni graf*** (LTG).
- Ne samo da se robot može efikasnije kretati prema cilju korištenjem LTG-a, već se može **kretati prečicama kada obilazi prepreke duž njihovih kontura i usmjerava se prema cilju ranije**.

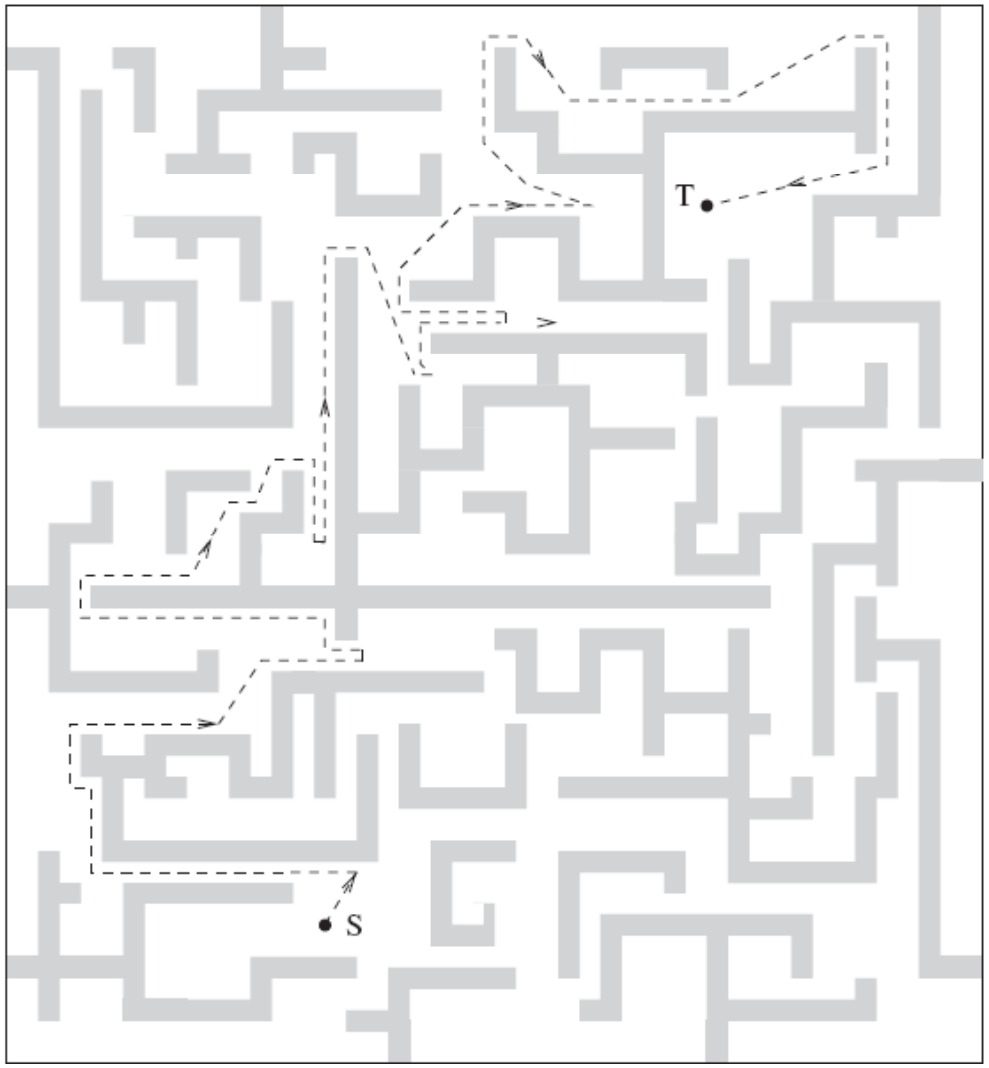
Bug algoritam

- Osim Bug algoritama koji koriste senzore dodira (Bug 1, Bug2) i senzore udaljenosti (TangentBug), razvijeni su i Bug algoritmi koji koriste vizualne informacije iz okoline, dobivene kamerom, takozvani **VisBug** algoritmi.



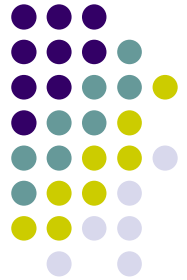
Bug algoritam

- Najpoznatiji iz ove grupe algoritama su *VisBug 2-1* i *VisBug 2-2* [Lumelsky, 2006].



Rezultati sa *VisBug 2-1* algoritmom

Bug algoritam



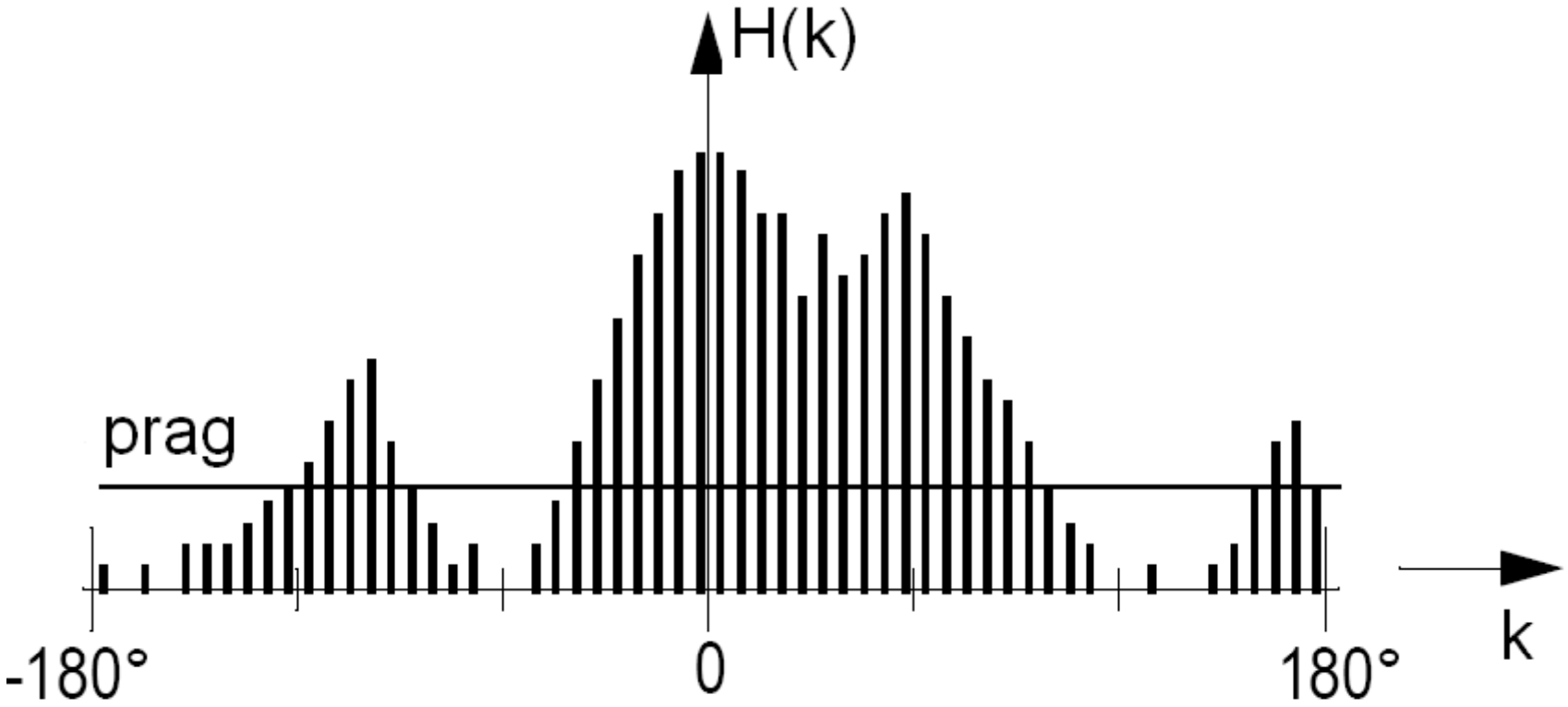
- **U mnogim jednostavnijim okolinama, tangecijalni Bug pristupi globalno optimiraju putanju**.
- **Nedostatak Bug2 algoritma je da ne uzima u obzir kinematiku robota**, što predstavlja važan segment, posebno kod neholonomskih robota.
- Osim toga, budući da se koriste senzorske vrijednosti, šum senzora može prouzročiti značajan utjecaj na performance realnog svijeta.
- Jedan od algoritama koji nadvladava neka od navedenih ograničenja je **VFH** (engl. Vektor Field Histogram) metoda.

13.1.2. VFH metoda

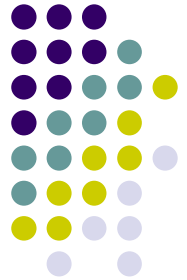
- Različite metode zasnovane na principima sličnim potencijalnim poljima su razvijene.
- Jedna od njih je algoritam ***virtualnih sila polja*** (VFF – Virtual Field Force) kojim se konstruira mreža zauzeća u on-line režimu i zatim pridružuju sile preprekama i cilju koje djeluju na robota.
- Ove sile usmjeravaju robota prema cilju uz izbjegavanje prepreka, ukoliko mu se nađu na putu.
- Za razliku od metoda potencijalnih polja, **VFF metoda razmatra sile samo u maloj, lokalnoj okolini oko robota**.
- **Ovom metodom se prostorna distribucija sile reducira na pojedinačni vektor, što je i glavni nedostatak ove metode**.

VFH metoda

- VFH algoritam (Borenstein i Koren, 1991) prevazilazi ovaj nedostatak konstruiranjem ***polarnog*** prikaza gustoće prepreke.



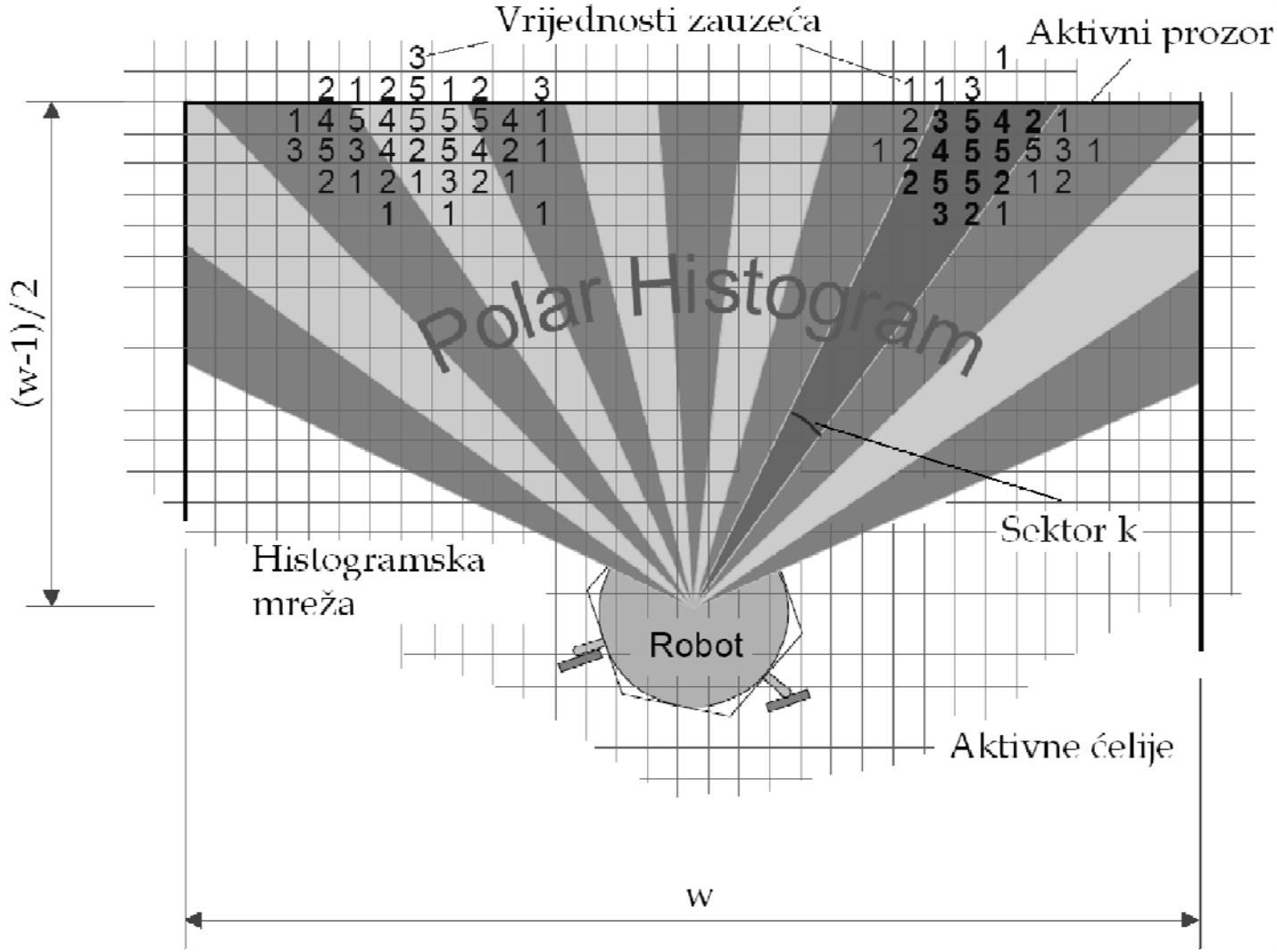
VFH metoda



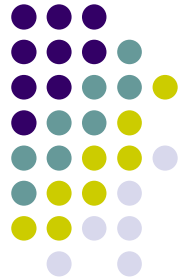
- Jednodimenzionalni histogram *H* sadrži *n* ugaonih sektora duljine *α*.
- Aktivni region *C** se preslikava u histogram *H* u kome svaki sektor *k* pohranjuje vrijednost h_k koja predstavlja gustoću prepreke u polarnom histogramu u smjeru koji odgovara sektoru *k*.
- **Sektori koji premašuju prag označavaju neprohodnost prostora**.
- Za transformaciju se koristi ***pomični otvor*** (prozor), dimenzija *w*×*w*, koji se kreće zajedno sa mobilnim robotom i pokriva dio histogramske mreže (slika na sljedećem slajdu).

VFH metoda

- Preslikavanje aktivnih ćelija u polarni histogram.



VFH metoda



- Sadržaj svake ***aktivne ćelije*** u histogramskoj mreži se tretira kao *vektor prepreke*, čija orijentacija β je određena u odnosu na ***centralnu tačku robota*** (centar mase ili centar osovine) VCP (engl. Vehicle Central Point):

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

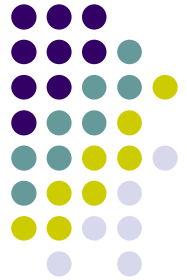
dok je amplituda vektora prepreke dana sa:

$$m_{i,j} = c^*_{i,j}(a - bd^2_{i,j})$$

VFH metoda

- Oznake u prethodnim jednadžbama su:
- $c^*_{i,j}$ - vrijednost sigurnosti pridružena aktivnoj ćeliji (i, j),
- $m_{i,j}$ - amplituda vektora prepreke pridruženog ćeliji (i, j),
- $d_{i,j}$ - udaljenost između aktivne ćelije (i, j) i VCP-a,
- x_r , y_r - koordinate trenutne pozicije centra robota VCP,
- x_i , y_j - koordinate aktivne ćelije (i, j),
- $\beta_{i,j}$ - pravac od aktivne ćelije (i, j) do VCP-a.

VFH metoda



- Konstante a i b moraju zadovoljavati sljedeću relaciju:

$$a - b\left(\frac{w-1}{2}\right) = 1$$

- *H* ima proizvoljnu rezoluciju *α* takvu da je *n* = 360/*α* cjelobrojnik (naprimjer, *α* = 5° i *n* = 72).
- Svakom sektoru *k* odgovara diskretni ugao *ρ*, kvantiziran kao višekratnik ugla *α*, koji iznosi *ρ* = *kα*, gdje je *k* = 0, 1, 2, ..., *n*-1.
- Povezanost aktivne ćelije sa sektorom *k* izražena je na sljedeći način:

$$k = \text{int}\left(\frac{\beta_{i,j}}{\alpha}\right)$$

VFH metoda

- Gustoća prepreke u polarnom histogramu h_k za svaki sektor k računa se kao:

$$h_k = \sum_{i,j} m_{i,j}$$

- Svaka aktivna ćelija je povezana sa određenim sektorom preko jednadžbi za računanje β i k.
- Na slici sa slajda 21. koja prikazuje preslikavanje iz C^* u H, sve aktivne ćelije povezane sa sektorom k su istaknute.
- Širina sektora na ovoj slici iznosi $\alpha = 10°$ (ne $\alpha = 5°$, kako je u stvarnom algoritmu) da bi prikaz bio jasniji.

VFH metoda

- Zbog diskretnog karaktera histogramske mreže, rezultat ovog preslikavanja izgleda nedotjeran ("čupav") i uzrokuje pogreške u izboru smjera napredovanja robota.
- Osim toga, izglađena funkcija primijenjena na histogram *H* definirana je sa:

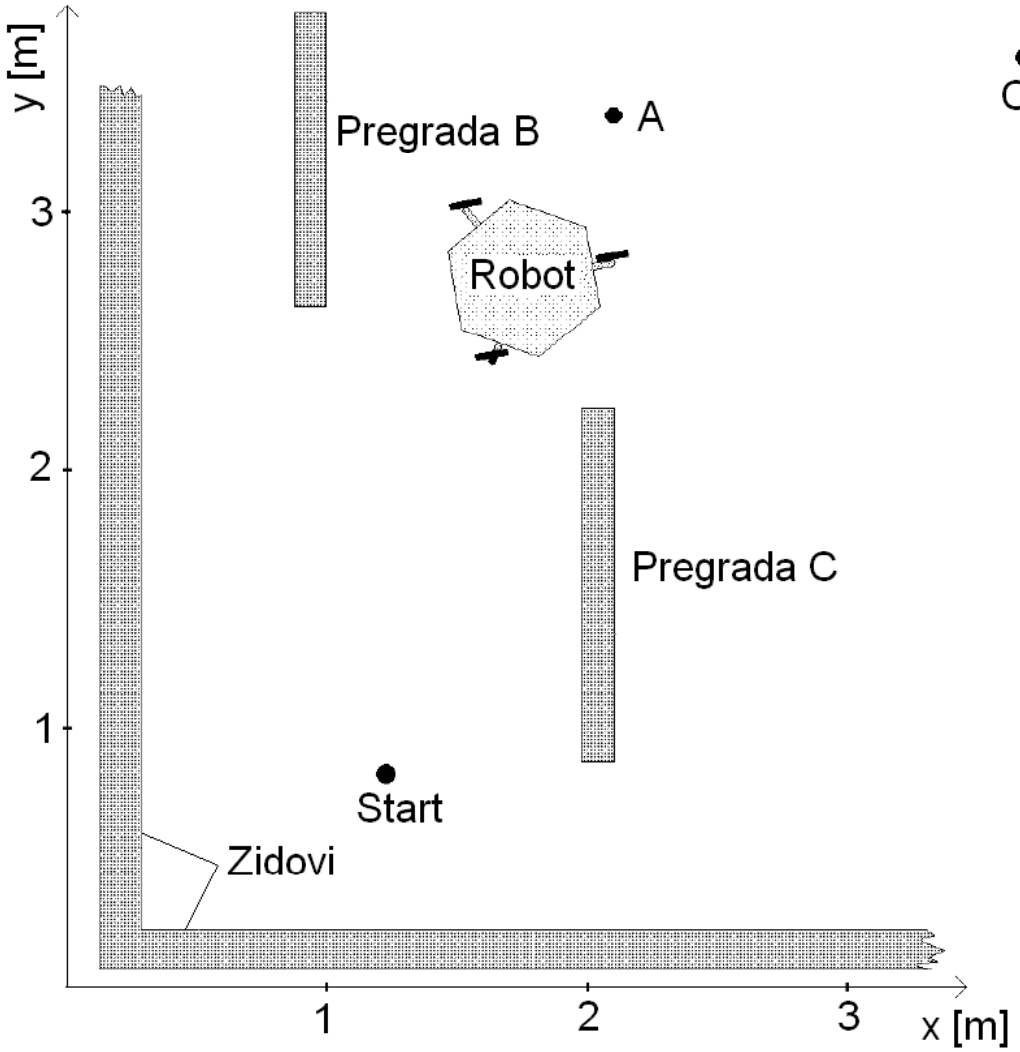
$$h'_k = \frac{h_{k-1} + 2h_{k-l+1} + \ldots + lh_k + 2h_{k+l-1} + h_{k+1}}{2l+1}$$

gdje h'_k predstavlja ***izglađenu polarnu gustoću prepreke***.

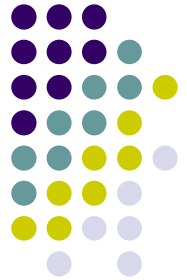
- U primjeru (Borenstein i Koren, 1991) *l*=5 daje zadovoljavajuće izglađene rezultate.

VFH metoda

- Na sljedećoj slici prikazana je prostorija sa preprekama (Borenstein i Koren, 1991).



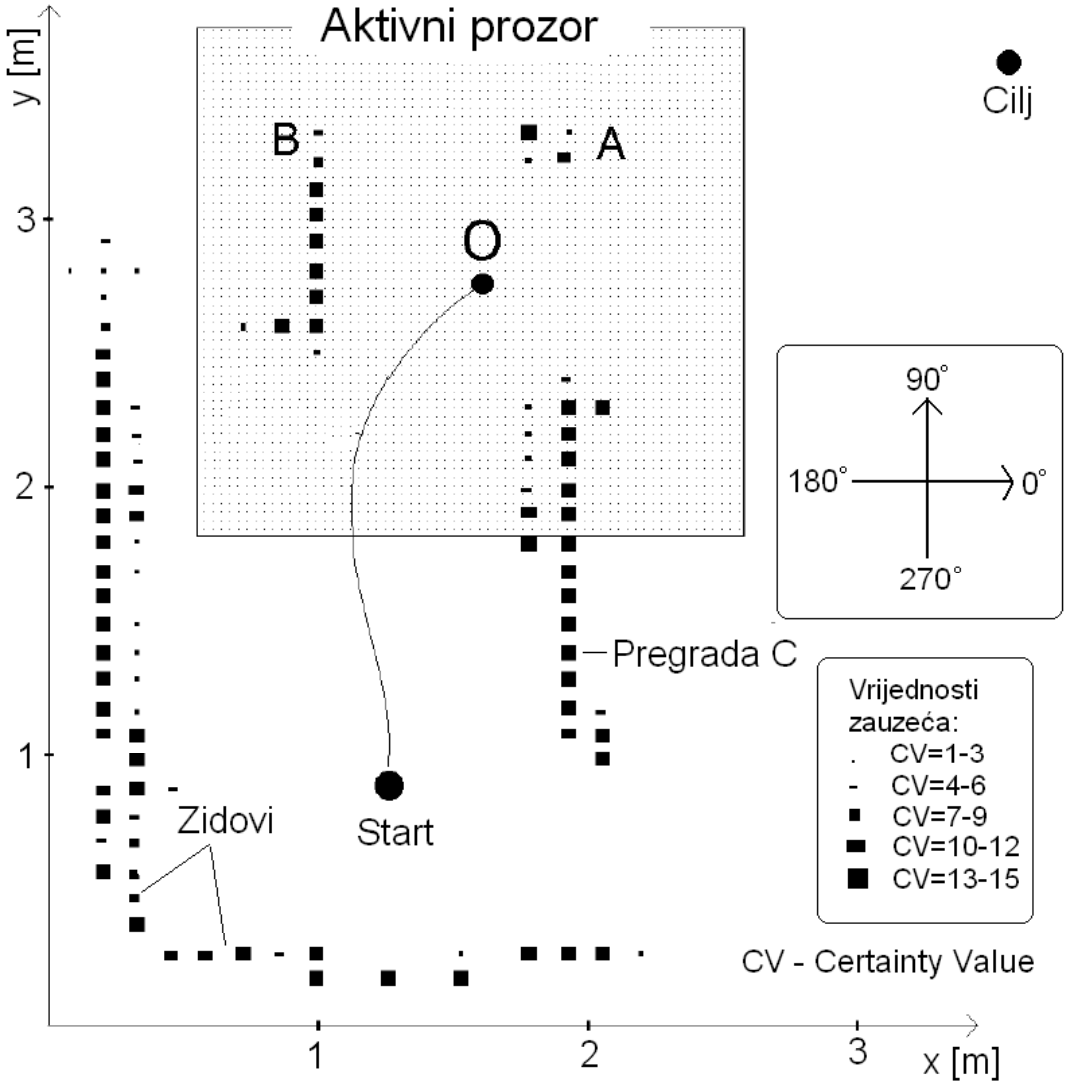
VFH metoda



- Razmak između prepreke (pregrade) *B* i *C* iznosi 1.2 metra.
- Prepreka *A* je kružnog oblika promjera 3/4".
- Histogramska mreža situacije sa slike na slajdu 27. prikazana je na slajdu 29., dok je polarni histogram iste predočen slikom na slajdu 30.
- Pravci (izraženi u stupnjevima) u polarnom histogramu odgovaraju smjerovima pravaca mjerenim u smjeru obrnutom od kretanja kazaljke na satu od pozitivne *x* osi histogramske mreže.
- Šiljci *A*, *B* i *C* u polarnom histogramu su rezultat segmentiranja prepreka *A*, *B* i *C* u histogramskoj mreži.

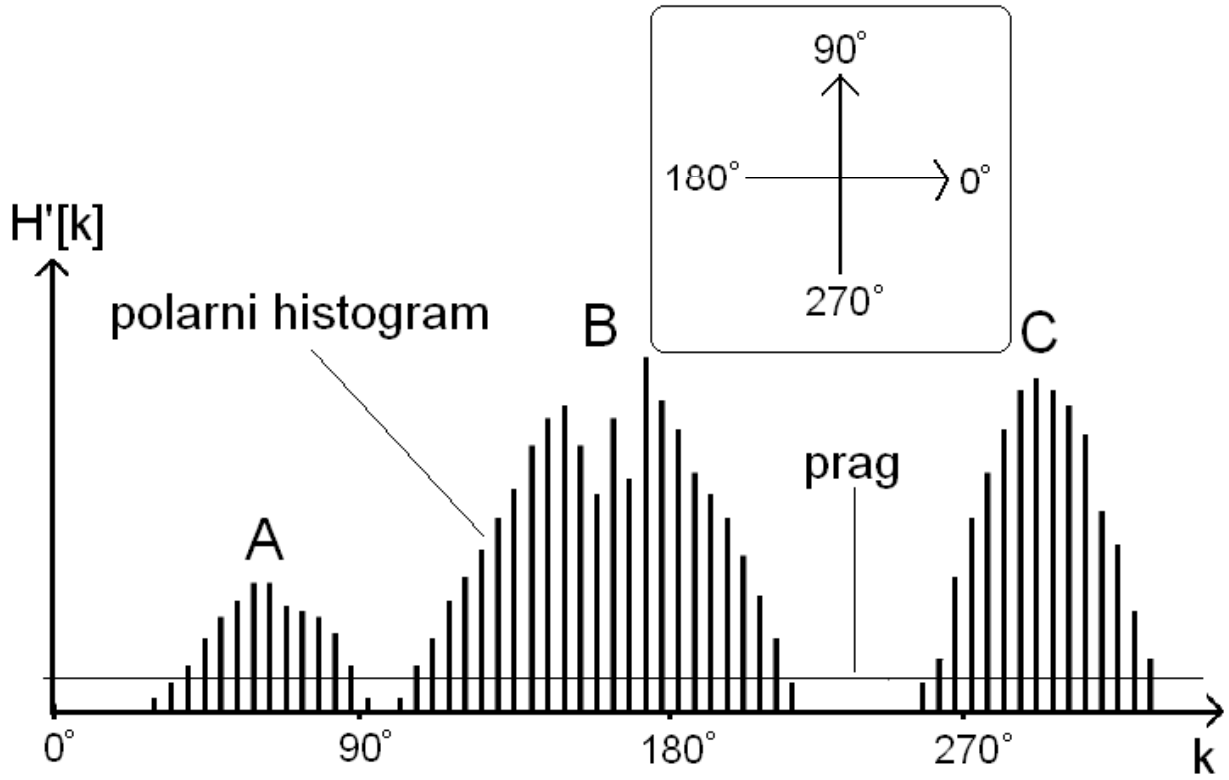
VFH metoda

- Histogramska mreža



VFH metoda

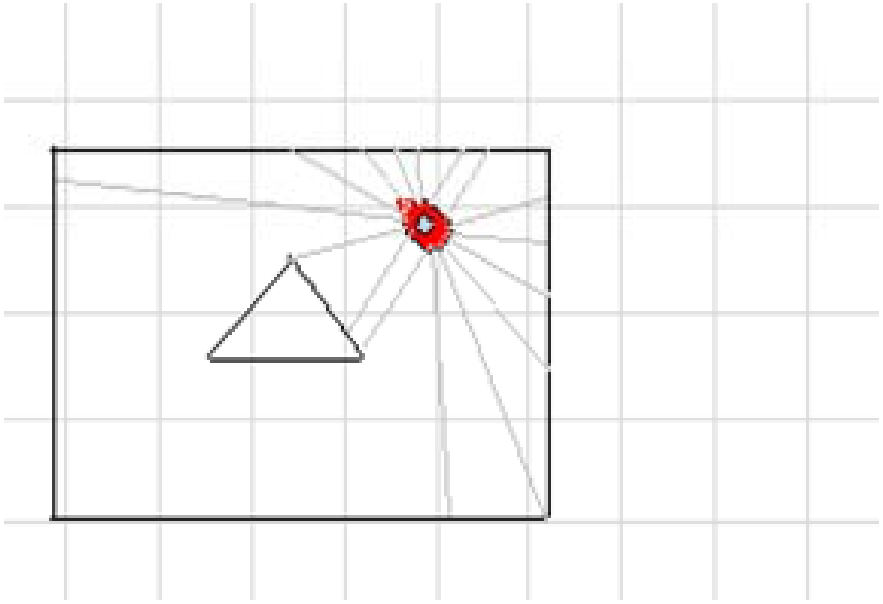
- Polarni histogram prostora za situaciju sa slajda 27.



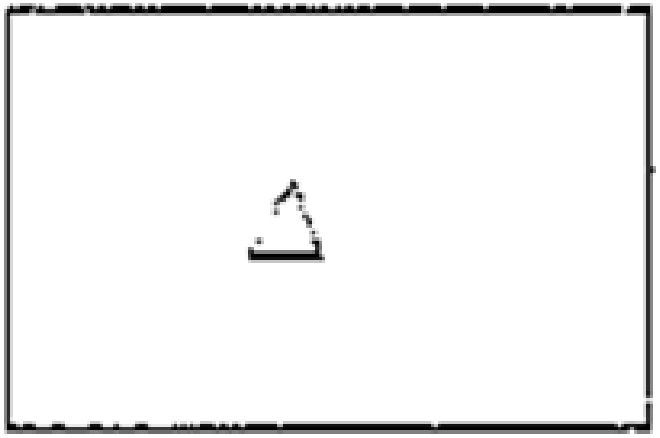
- Izvorni VFH algoritam je doživio izmjene i poboljšanja, tako da su se razvili VFH+ i VFH* (Ulrich i Borenstein, 1998, 2000).

VFH metoda

- Rezultati VFH sa Pioneer P3-DX mobilnim robotom



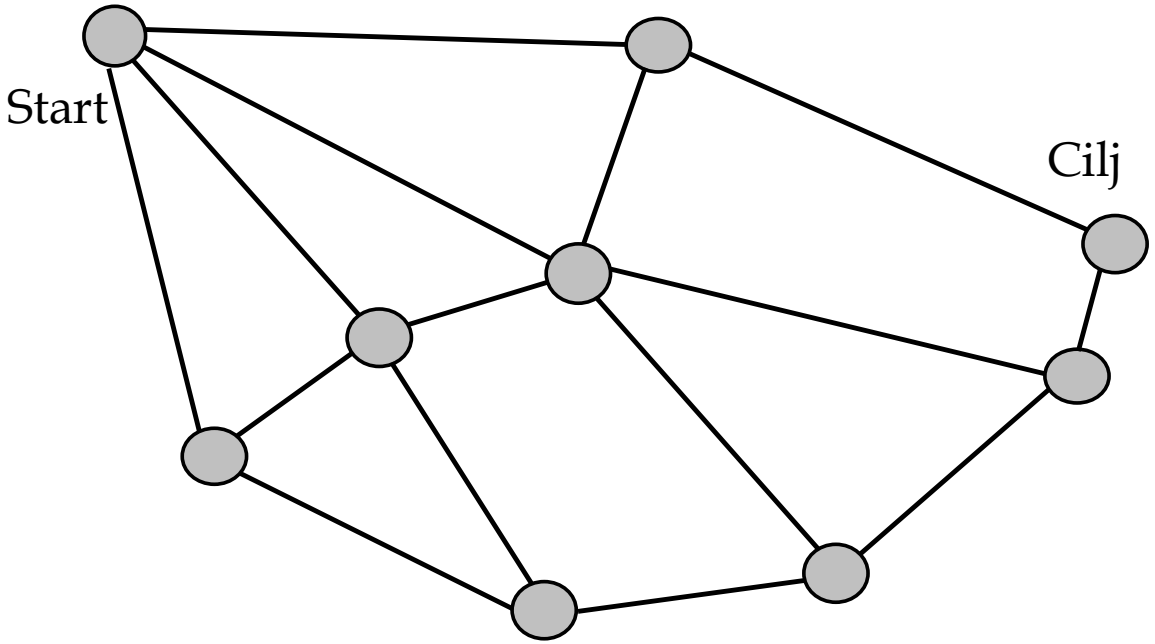
100 snimljenih tačaka



1000 snimljenih tačaka

13.2. Algoritmi pretraživanja grafova

- Prikaz prostora pomoću grafova (metričkog ili topološkog) se dobiva reduciranjem prostora u segmente.
- Pomoću povezanog grafa (slika ispod) može se putanja od početne (start) do ciljne tačke veoma jednostavno pronaći.



Algoritmi pretraživanja grafova

- Mnogi od algoritama pretraživanja grafova **zahtijevaju program za posjetu svakom čvoru u grafu** kako bi odredili najkraću putanju od početne do ciljne tačke.
- Posjećivanje svakog čvora može biti računarski naporno u slučaju razrijeđenih povezanih grafova, kao što je slučaj sa grafovima dobivenih pomoću Voronoijevih dijagrama.
- Također, u slučaju grafova sa mnoštvom čvorova i veza između njih pretraživanje postaje računarski prezahtjevno.
- Što se tiče postupaka pretraživanja grafova najviše su u upotrebi ***A**** **algoritam**, **dinamičko programiranje** i **Dijkstra algoritam**.

13.2.1. A* algoritmi pretraživanja grafova

- **A* predstavlja klasični postupak računanja optimalnih putanja holonomskih robota izveden iz A algoritma.**
- Da bi se moglo objasniti kako funkcionira A* algoritam prvo će se ukratko prezentirati način rada A algoritma.
- Oba algoritma pretpostavljaju **metričku mapu**, gdje je lokacija svakog pojedinačnog čvora **poznata** u apsolutnim koordinatama i spojne linije grafa prikazuju da li je moguće ostvariti kretanje između čvorova.
- *A* algoritam pretraživanja prostora generira optimalnu putanju od početnog do ciljnog čvora i nakon toga omogućuje kretanje kroz graf do ciljnog čvora.

A* algoritmi pretraživanja grafova

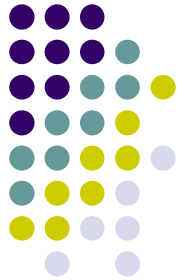
- Optimalnu putanju on generira inkrementalno, osvježava je u svakom koraku, i razmatra koji bi čvorovi mogli biti pridodani putanji i odabire između njih one najbolje.
- "Srce" ovog postupka predstavlja sljedeća formula za mjerenje vjerodostojnosti čvora:

$$f(n) = g(n) + h(n)$$

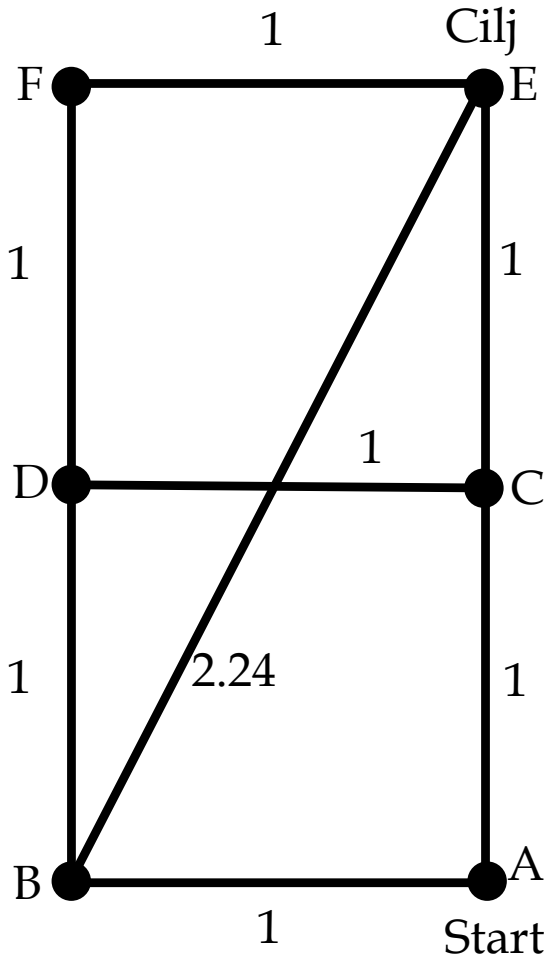
gdje je:

- $f(n)$ ocjena (mjeri) kako se dobro kreće čvor n,
- $g(n)$ funkcija kakvoće putanje od početnog do n-tog čvora,
- $h(n)$ funkcija kakvoće putanje od n-tog do ciljnog čvora.

A* algoritmi pretraživanja grafova



- Kako funkcionira prethodni izraz objasnit će se u sljedećem primjeru grafa prikazanog na slici.



A algoritam započinje u čvoru *A* i kreira stablastu strukturu kojom određuje koje je čvorove moguće dodati u graf.

$$f(B) = g(B) + h(B) = 1 + 2.24 = 3.24$$

g(*B*) ocjenjuje kvalitetu prijelaza iz *A* u *B*, a *h*(*B*) iz *B*-a u ciljni čvor *E*.

$$f(C) = g(C) + h(C) = 1 + 1 = 2$$

A algoritam pronalazi optimalnu putanju, ali pri tome obavlja usporedbu svih putanja međusobno.

A* algoritmi pretraživanja grafova

- A* algoritam predstavlja interesantan pristup **reduciranja broja putanja koje se mogu generirati i međusobno uspoređivati**.
- On uspoređuje moguće putanje sa **najboljom putanjom**, čak i ako ona ne postoji u realnom prostoru, odnosno nije je moguće postići.
- Algoritam estimira *h* i ne provjerava da li postoji stvarni segment putanje koji može dovesti do cilja sa te udaljenosti.
- Estimacija se nakon toga može koristiti za određivanje od kojih se čvorova najviše očekuje u procesu pretrage i kojim putanjama se ne može doći do ciljnog čvora i na taj način reducirati proces pretraživanja.

A* algoritmi pretraživanja grafova

- Evaluacijska funkcija A* algoritma postaje:

$$f^*(n) = g^*(n) + h^*(n)$$

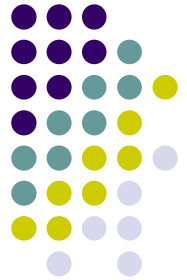
gdje oznaka '*' govori da se radi o estimiranim vrijednostima koje se mogu uključiti u A algoritam pretrage.

- U planiranju putanje, $g^*(n)$ je ista kao i $g(n)$, s tim da ona postaje poznata kroz **inkrementalnu gradnju putanje**.
- Međutim, $h^*(n)$ je posve različita od $h(n)$.
- **Postavlja se onda pitanje kako estimirati putanju od čvora *n* do ciljnog čvora?**

A* algoritmi pretraživanja grafova

- **Također se postavlja pitanje kako biti siguran da će estimacija biti dovoljno tačna da osigura optimalnu putanju?**
- Ovo se može osigurati tako da $h^*(n)$ nikad ne bude manje od $h(n)$.
- Restrikcija $h^*(n) \leq h(n)$ naziva se ***uvjet prihvatljivosti***.
- Budući da se estimira $h^*(n)$, ova funkcija se naziva još i heuristička funkcija, jer koristi pravila kojima se odlučuje koji čvor je najbolji za daljnje razmatranje.
- Srećom, postoji prirodna heuristička funkcija za estimaciju kvalitete putanje od čvora *n* do ciljnog čvora. Radi se o **Euklidskoj distanci**.

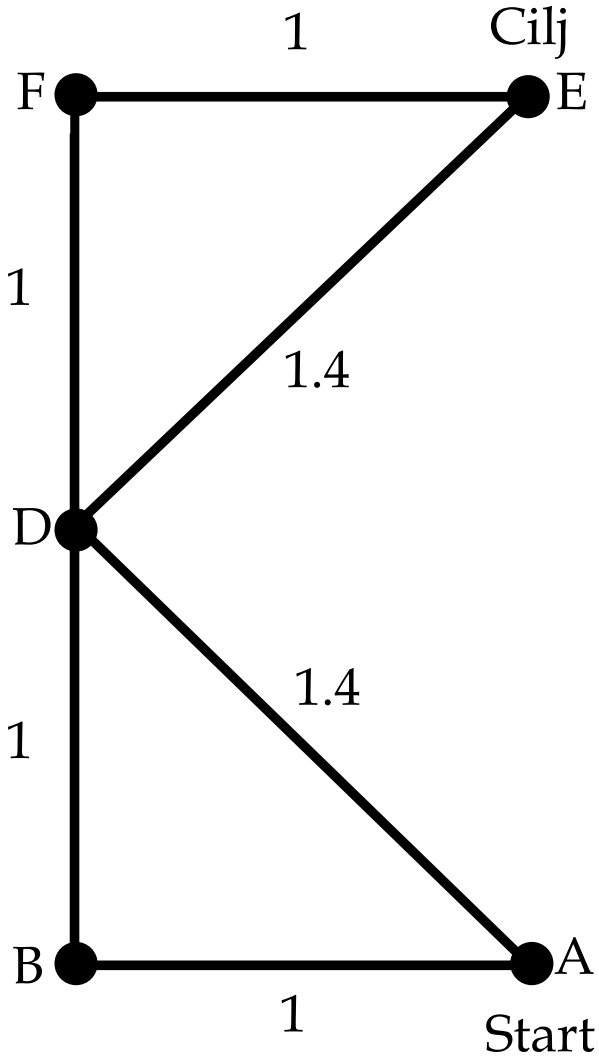
A* algoritmi pretraživanja grafova



- Lokacija svakog čvora je poznata i moguće je izračunati duljinu spojne linije između dva čvora.
- Najkraća putanja između dva čvora je prava linija.
- Budući da realna putanja nikad ne može biti kraća od pravolinijske putanje, slijedi da je uvjet prihvatljivosti zadovoljen.
- Da bi se vidjelo kako A* algoritam **eliminira posjećene čvorove**, razmatra se primjer sa sljedeće slike.
- Prvi korak u ovom algoritmu je razmatrati izbore iz početnog čvora.

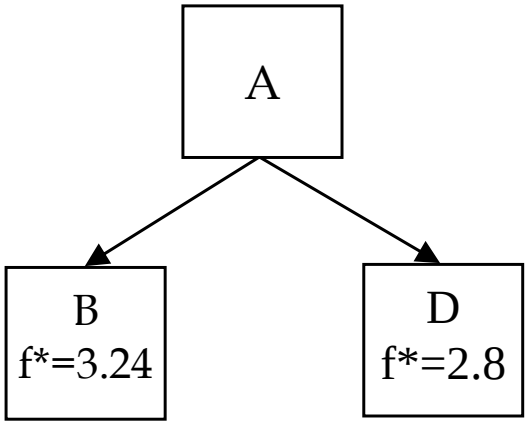
A* algoritmi pretraživanja grafova

- Prikaz grafa za pretragu

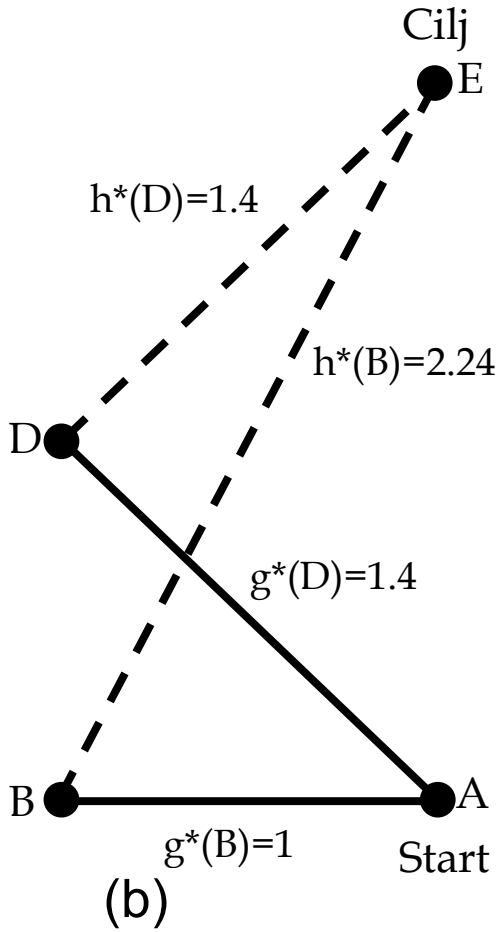


A* algoritmi pretraživanja grafova

- Izbori su čvorovi *B* i *D*, kojima se može kroz stablo grafa direktno pristupiti iz početne tačke *A* (slika a) ili preko podskupova originalnog grafa (slika b)



(a)



(b)

A* algoritmi pretraživanja grafova

- Slika b. prikazuje kako algoritam "vidi" ovu tačku u toku postupka izvršavanja.
- Izbori evaluiraju u:

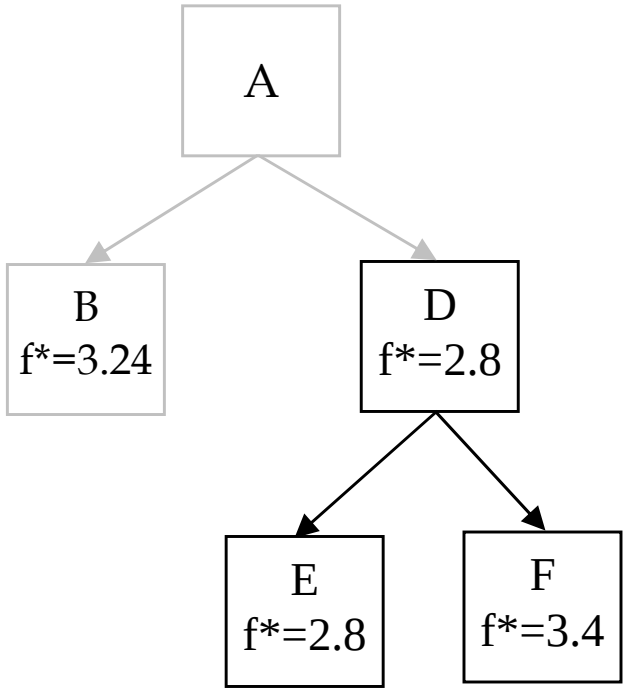
$$f^*(B) = g^*(B) + h^*(B) = 1 + 2.24 = 3.24$$

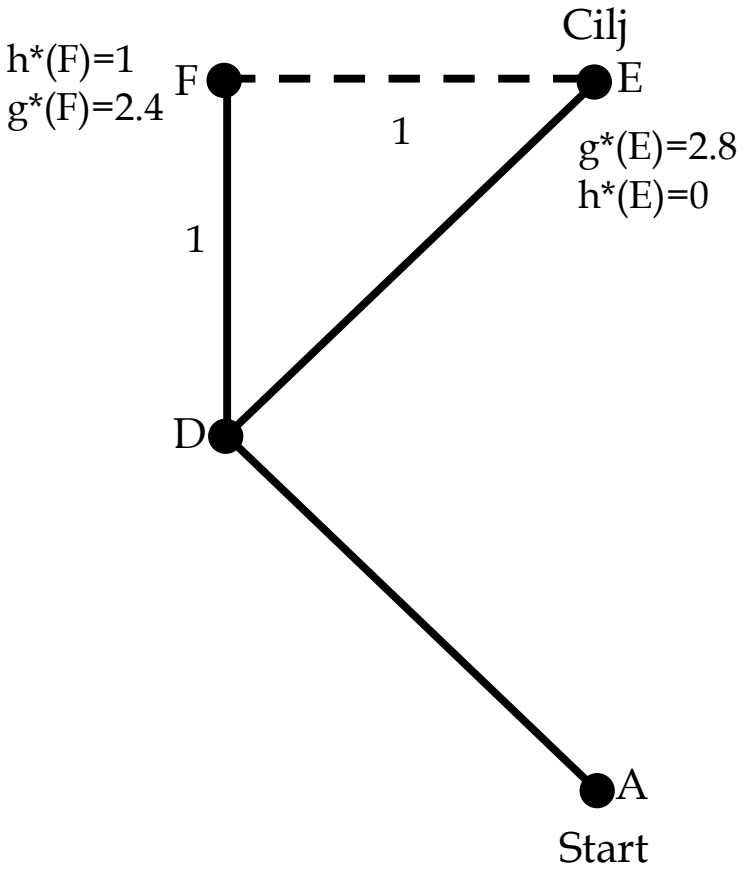
$$f^*(D) = g^*(D) + h^*(D) = 1.4 + 1.4 = 2.8$$

- Putanja *A*–*D*-?–*E* je kraća od putanje *A*–*B*-?–*E*. Slijedi da je *D* najprihvatljivija tačka putanje gledano iz početne tačke *A*.
- Važno je napomenuti da A* algoritam **ne može eliminirati stazu** kroz čvor *B* jer algoritam ne može "vidjeti" putanju koja stvarno ide iz čvora *D* u čvor *E* i da je ona zaista najkraća moguća putanja grafa do cilnog čvora *E*.

A* algoritmi pretraživanja grafova

- U drugom koraku, A* algoritam rekurzivno računa (ponavljajući evaluaciju) iz čvora *D*, jer je *D* najprihvatljiviji sljedeći čvor na putanji prema cilju, što se vidi na sljedećoj slici.





A* algoritmi pretraživanja grafova

- Dvije opcije putanje iz *D* su *E* i *F* čvorovi, koje se evaluiraju kako slijedi:

$$f^*(E) = g^*(E) + h^*(E) = 2.8 + 0 = 2.8$$
$$f^*(F) = g^*(F) + h^*(F) = 2.4 + 1 = 3.4$$

- Iz algoritma se vidi da je čvor *E* najbolji izbor da ostane u stablu pretrage, uključujući spojnu liniju kroz *B*. *E* je bolji izbor od *F* i *B*.
- Kada A* algoritam nastoji proširiti putanju iz čvora *E* uočava da je taj čvor ciljni, tako da se algoritam kompletira.
- **Optimalna putanja je *A-D-E*** i ne mora se eksplicitno razmatrati putanja *A-B-F-E*.

A* algoritmi pretraživanja grafova

- Prema tome, bilo koja putanja mora prolaziti kroz čvor *D*, tako da **spojne linije koje spajaju čvor *B* sa drugim čvorovima se eliminiraju iz grafa pretraživanja**.
- Naravno, u gornjem primjeru algoritam nikad ne primjećuje ovo jer se čvor *B* nikad ne proširuje.
- Korištenje A* algoritma je posebno korisno kod grafova kreiranih iz pravilnih mreža, gdje je rezultat visoko povezani graf.
- Također je atraktivna njegova primjena u bilo kojem *C*-prostoru koji se može transformirati u graf.

A* algoritmi pretraživanja grafova

- U ovom slučaju je samo pitanje koliko mnogo računanja A* planer može obaviti kako bi pronašao putanju od početnog do ciljnog čvora.
- **Ograničenje A* algoritma je da ga je veoma teško koristiti za planiranje putanje gdje postoje i drugi faktori, osim udaljenosti između čvorova, koji se razmatraju u procesu generiranja putanje.**
- Naprimjer, pravolinijska spojna linije može se protezati preko stjenovitih ili pješčanih terena koji mogu biti rizični za robota.
- Isto tako, robot nastoji izbjegavati kretanje uz brdo zbog uštede energija, a istovremeno se želi kretati niz brdo kad je god to moguće.

A* algoritmi pretraživanja grafova

- Ovisno o tome kakav je teren kojim prolazi robot, odnosno njegova funkcija cijene kretanja, potrebno je mijenjati heurističku funkciju $h^*(n)$.
- Pri tome se mora voditi računa da ova nova heuristička funkcija mora uvijek zadovoljavati **uvjet prihvatljivosti** $h^*(n) \leq h(n)$.
- Ako novi $h^*(n)$ predstavlja najlošiji slučaj trošenja energije ili najlošije sigurnosti, on će biti prihvatljiv, ali neće biti od posebne koristi u reduciranju broja putanja.
- Također, smanjenje utroška energije kada se robot kreće niz brdo ima za posljedicu postojanje čvorova u grafu sa negativnim težinama kojim A* algoritam ne može rukovati.

A* algoritmi pretraživanja grafova

- U ovakvim situacijama se može koristiti Bellman-Ford algoritam pretraživanja grafa.
- Jedno od poboljšanja A* algoritma pretraživanja grafova je D* algoritam [Stentz, 1995] koji je sličan A* algoritmu, ali je dinamičan, to jest funkcija cijene koštanja spojnih linija se može mijenjati u toku postupka pronalaženja optimalne putanje od početnog do ciljnog čvora.
- Daljnje poboljšanje je D* fokusirani algoritam koji koristi heurističke funkcije za određivanje cijene prijelaza između dva stanja (cijena koštanja spojnih linija).

13.2.2. Dijkstra algoritam

- Dijkstra algoritam kao ulaz koristi težinski usmjereni graf $G = (V, E)$ sa nenegativnim bridovima, težinsku funkciju $w : E \to R^+$, početnu tačku s i ciljnu tačku u, a kao izlaz daje podatke o najkraćoj putanji $d[u] = \delta(s, u)$ od početne do ciljne tačke.
- **Princip na kome funkcionira Dijkstra algoritam je sljedeći:**
  - Definiran je skup vrhova S, čiji su elementi vrhovi za koje su već određene najkraće putanje od početnog vrha s. Ukoliko nije određena najkraća putanja niti za jedan vrh, tada je skup S prazan skup.

Dijkstra algoritam

- Algoritam iterativno bira vrhove *u*, koji su elementi skupa *V*, a nisu elementi skupa *S*, u oznaci $u \in (V/S)$, procjenjuje im najkraću putanju i dodaje ih skupu *S*. Ovo dodavanje se vrši tako što je svaki sljedeći vrh najbliži prethodnom vrhu.
- Zatim se vrši operacija ***relaksacije*** (otpuštanja) svih ivica koje izlaze iz vrha *u* i procjena novih najkraćih putanja do svih dostupnih vrhova. Relaksacija znači da se za susjedne vrhove zadnje dodatog vrha računaju nove udaljenosti. Ako su one po vrijednosti manje od postojećih najkraćih, nova se vrijednost upisuje u niz preko stare. Ukoliko su veće, stara vrijednost se zadržava. Informacije o najkraćim udaljenostima i putanjama se čuvaju u prioritetnim redovima.

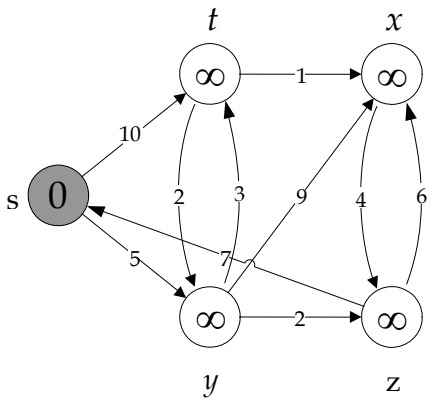
Dijkstra algoritam

- Algoritam se završava nakon što u skupu *S* budu svi čvorovi iz skupa *V*, a iz prioritetnih redova se kao rezultat daju informacije o najkraćoj udaljenosti i najkraćoj putanji.
- U općem slučaju vrijeme izvršavanja Dijkstra algoritma je $O(n^2)$, gdje je *n* broj vrhova u skupu *V*, za rješavanja zadatka određivanja najkraćih putanja od jednog vrha *s* do svih ostalih vrhova iz skupa *V*.
- Ova je vrijednost određena na osnovu analize rada algoritma.
- Naime, svaka se nova minimalna udaljenost ubacuje u niz, što se izvrši za $O(1)$ vremenskih trenutaka.

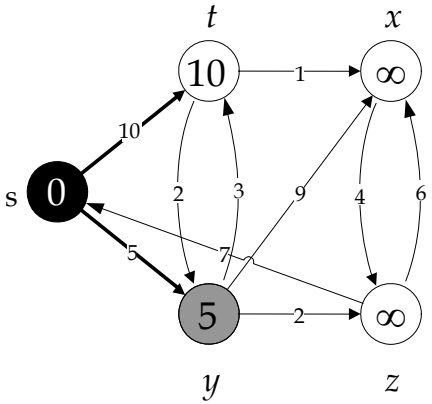
Dijkstra algoritam

- Računanje nove minimalne vrijednosti u nizu se izvršava u $O(n)$ vremenskih trenutaka, jer se pretražuje cijeli niz, što rezultira konačnim vremenom izvršavanja od $O(n^2)$.
- Na sljedećoj slici prikazana je procedura po kojoj se implementacijom Dijkstra algoritma određuje najkraća putanja od početnog vrha *s* do svih ostalih vrhova u grafu.
- Pod a) je prikazana situacija kada je skup *S* prazan.
- To je i označeno tako što svi vrhovi imaju oznaku ∞, osim očiglednog rastojanja vrha od samog sebe.

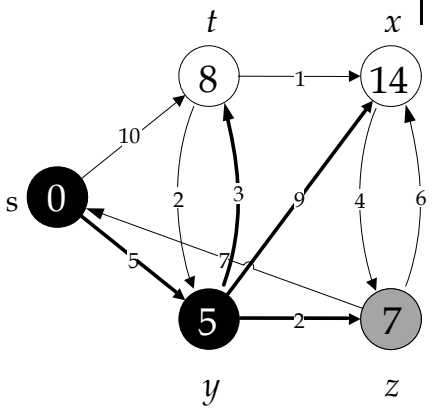
Dijkstra algoritam



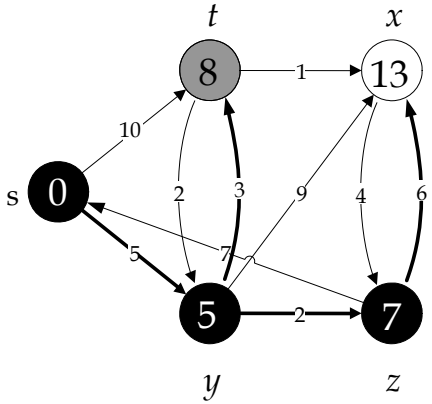
(a)



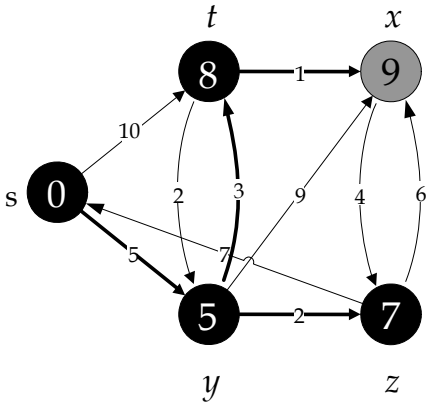
(b)



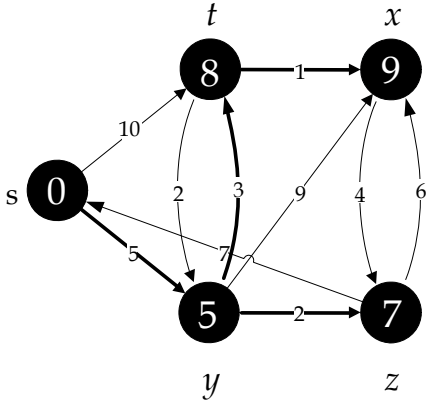
(c)



(d)



(e)



(f)

Dijkstra algoritam

- U sljedećem koraku (b), skupu se dodaje vrh *s*, zatamnjeni vrh na slici, i to je prvi korak. Prema gore navedenoj proceduri vrši se relaksacija ivica i procjenjuju se udaljenosti početnog vrha do susjednih vrhova ($w(s, t) = 10$, $w(s, y) = 5$).
- Sličan proces se nastavlja sve dok u skupu *S* ne budu svi vrhovi iz skupa *V*. Interesantna situacija je promjena vrijednosti najkraće putanje do vrha *x*, u koracima c), d) i e). Naime, prvo je najkraća udaljenost iznosila 14, pa 13 i konačno 9 tokom pomenutih koraka, respektivno. Ova promjena je posljedica toga da su se dodavanjem vrhova u skup *S* otkrile nove, kraće putanje, do konačne putanje *s*-*y*-*t*-*x*, za vrh *x*.

Dijkstra algoritam

- Jedna od mogućih reprezentacija grafa sa prethodne slike je pomoću **matrice susjedstva**.

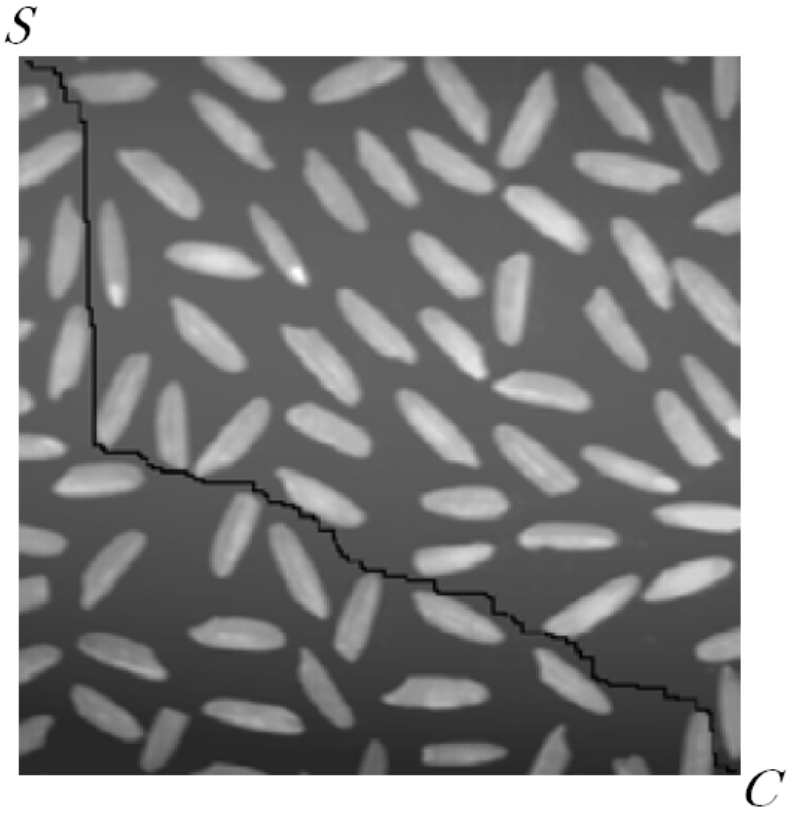
-	s	t	x	Y	z
s	0	10	∞	5	∞
t	∞	0	1	2	∞
x	∞	∞	0	∞	4
y	∞	3	9	0	2
z	8	∞	6	∞	0

13.2.3. Usporedba Dijkstra i A* algoritama

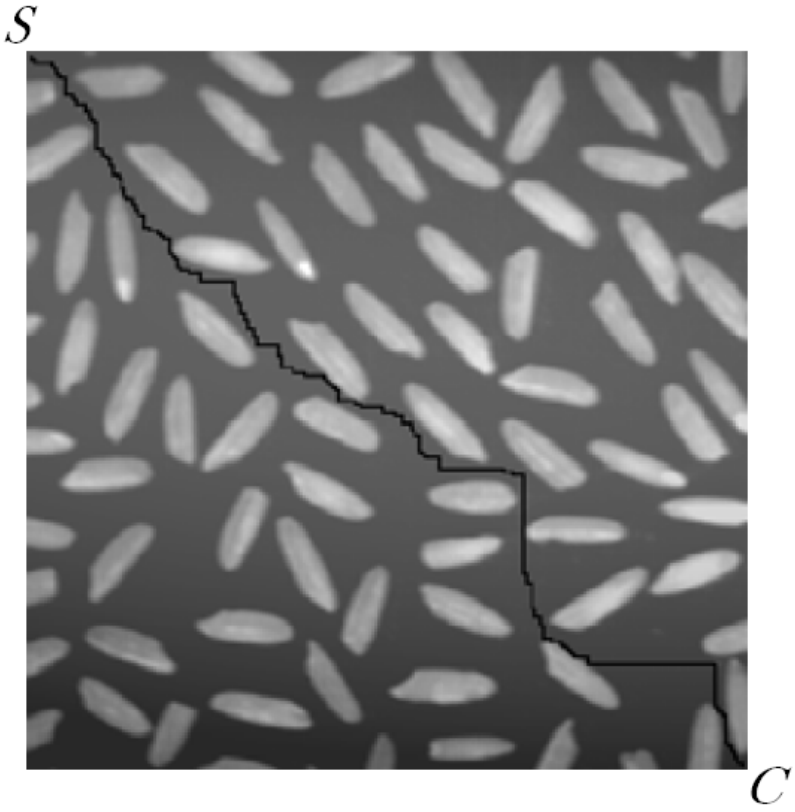
- U nastavku se prikazuje primjena Dijkstra i A* u pronalaženju najkraće putanje od početne do ciljne lokacije mobilnog robota.
- Prostor pretrage, mapa prostora, predstavljen je poljem (256×256) ćelija, pri ćemu inicijalno svaka ćelija ima istu težinsku vrijednost.
- Slika je preuzeta iz Matlaba (Image Processing Toolbox) i nakon toga je konvertirana u metričku mapu.
- Ćelije bijele boje predstavljaju zauzete ćelije (prepreke), a sive ćelije slobodni prostor.
- Početne i ciljne pozicije robota su tačke sa koordinatama (2,2) i (255,255).

Usporedba Dijkstra i A* algoritama

- Rezultati sa Dijkstra (a) i A* algoritmom (b).



(a)



(b)

Usporedba Dijkstra i A* algoritama

- Vremena pronalaženja najkraće putanje iznose 2.625 sekundi kod Dijkstra algoritma i 2.298 sekundi kod A* algoritma.
- Prilikom pretrage korištenjem Dijkstra algoritma posjećeno je 43004 ćelije, a primjenom A* algoritma posjećeno je ukupno 25134 ćelije.
- Prema tome A* algoritam brže pronalazi optimalnu putanju od startne do ciljne pozicije mobilnog robota.