

# **Mars-luotainten reitinhakualgoritmit ja visuaalinen paikantaminen**

Jerry Mesimäki

Kandidaatintutkielma  
HELSINGIN YLIOPISTO  
Tietojenkäsittelytieteen laitos

Helsinki, 20. lokakuuta 2014

Tiedekunta — Fakultet — Faculty		Laitos — Institution — Department	
Matemaattis-luonnontieteellinen		Tietojenkäsittelytieteen laitos	
Tekijä — Författare — Author			
Jerry Mesimäki			
Työn nimi — Arbetets titel — Title			
Mars-luotainten reitinhakualgoritmit ja visuaalinen paikantaminen			
Oppiaine — Läroämne — Subject			
Tietojenkäsittelytiede			
Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages	
Kandidaatintutkielma	20. lokakuuta 2014	7	
Tiivistelmä — Referat — Abstract			
Tiivistelmä.			
Avainsanat — Nyckelord — Keywords			
avainsana 1, avainsana 2, avainsana 3			
Säilytyspaikka — Förvaringsställe — Where deposited			
Muita tietoja — Övriga uppgifter — Additional information			

# Sisältö

<b>1</b>	<b>Johdanto</b>	<b>1</b>
<b>2</b>	<b>MER eli Mars Exploration Rover</b>	<b>1</b>
2.1	Mönkijöille asetetut tavoitteet . . . . .	2
2.2	Laitteisto . . . . .	2
2.3	AutoNav . . . . .	2
2.4	GESTALT ennen globaalia reitinsuunnittelua . . . . .	3
2.5	Päivitetty GESTALT . . . . .	3
<b>3</b>	<b>Field D* ja lineaarinen interpolointi</b>	<b>3</b>
3.1	Aikaisemmat algoritmit . . . . .	3
3.2	Kustannusarvion parantaminen interpoloinnin avulla . . . . .	3
3.3	Field D* . . . . .	5
<b>4</b>	<b>Visuaalinen paikannus</b>	<b>6</b>
4.1	Algoritmi . . . . .	7
	<b>Lähteet</b>	<b>7</b>

# 1 Johdanto

Ihmiskunta on ollut kiinnostunut avaruudesta ja erityisesti lähiplaneetoistaan jo antiikin ajoista asti. Tähtitieteen historiaan mahtuu useita merkkittäviä läpimurtoja kuten aurinkokeskeisen maailmankuvan ymmärtäminen ja vuoden 1969 Apollo-ohjelman miehitetty lento Kuuhun. Intressit avaruuden tutkimiselle kasvavat jatkuvasti ja viime vuosina myös kaupalliset tahot ovat luoneet työllisyyttä esimerkiksi avaruusturismin avulla. Jotkin yritykset tutkailevat jo mahdollisuuksia kaivaa kallisarvoisia mineraaleja planeettaamme kiertävistä asteroideista. Kaikki tämä vaatii useiden eri tieteenalojen yhteistyötä, jotta teknologiat kehittyisivät vastaamaan avaruuden asettamia haasteita.

Viimeisten vuosikymmenten aikana keskeisiksi tutkimusvälineiksi aurinkokuntamme kartoittamisessa ovat muodostuneet miehittämättömät avaruusluotaimet, joiden avulla tiedemiehet ovat keränneet huomattavia määriä kiinnostavaa dataa Maan naapureista. Tänä päivänä luotaimia löytyy jo useimpien planeettojen kiertoradoilta, Venuksen ja Marsin pinnoilta sekä tuoreimpien arvioiden mukaan Aurinkokunnan ulkopuolelta.

Kosmisessa mittakaavassa etäisyydet ovat aivan omaa luokkaansa ja tästä syystä luotainten ohjaaminen Maasta käsin on useimmissa tapauksissa hidasta. Asiaa hankaloittaa myös se ettei reaaliaikaisen kuvan saaminen ole mahdollista vaikka radioaallot kulkevat valonnopeudella halki avaruuden. Tästä johtuen tiedemiehet ovat pyrkineet automatisoimaan erilaisin algoritmein luotaintensa toimintaa. Tekoälyä taas käytetään, jotta laite kykenisi suorittamaan sille annetut tehtävät mahdollisimman itsenäisesti. Esimerkiksi Mars-mönkijät pyrkivät väistelemään maastosta löytyviä vaaroja kuvaamalla omaa ympäristöään ja analysoimalla sen perusteella itselleen suotuisimpia reittejä.

Uusimpien mönkijöiden kohdalla on jo mahdollista päivittää näiden ohjelmistoa vaikka ne sijaitsisivat toisella planeetalla. Siksi onkin oleellista jatkaa niissä käytettyjen ohjelmistojen kehittämistä vielä laukaisun jälkeen, jolloin laitteistojen potentiaalinen hyöty kasvaa ja tarve uusien laitteiden lähettämiseksi avaruuteen pienenee.

## 2 MER eli Mars Exploration Rover

Marsiin lähetettiin vuonna 2003 Nasan toimesta kaksi MER-laitetta (Mars Exploratorion Rover, myöhemmin mönkijä) – Spirit sekä Rover – etsimään merkkejä veden esiintymisestä planeetan pinnalta. Yksi ongelmista on mönkijöiden liikuttelu maasta käsin 26 minuuttia kestävän signaalin siirtymisestä aiheutuvan viiveen takia. Tyypillisesti laitteille lähetetään komentoja vain aamuisin kerran Marsin vuorokaudessa (myöhemmin kierto), jotka ne toteuttavat kierron aikana. Ennen iltaa niiden keräämä data taas lähetetään Maahan analysoitavaksi. Tästä johtuen mönkijöiden riittävä autonomia on

kriittistä, jotta ne kykenisivät liikkumaan mahdollisimman nopeasti tieteellisesti kiinnostavien kohteiden välillä.

Mönkijöitä voidaan liikuttaa kahdella eri tavalla: sokkoajona, jossa Maahan saapuneista kuvista päätellään mihin suuntaan on turvallista ajaa ja ajaminen suoritetaan vaaroista välittämättä haluttuun kohteeseen. Toinen vaihtoehto on luovuttaa vastuu ajamisesta mönkijöissä sijaitsevalle AutoNav-järjestelmälle, joka pyrkii liikkumaan haluttuun kohteeseen ottaen kuitenkin huomioon ympäristössä sijaitsevat vaarat ja väistämään niitä. Useimmiten näitä käytetään yhdessä siten, että sokkoajoa sovelletaan niin pitkälle kuin saaduista kuvista päätellen on turvallista ja tämän jälkeen AutoNav jatkaa ajamista.

Toisinaan automaatio ei toimi ja tämä estää mönkijöitä saavuttamasta kohteitaan. Heinäkuussa 2006 molemmat saivat järjestelmäpäivityksen mukana korjauksia em. ongelmaan, joita tämä essee pääasiassa käsittelee.

## **2.1 Mönkijöille asetetut tavoitteet**

## **2.2 Laitteisto**

## **2.3 AutoNav**

AutoNav perustuu GESTALT-algoritmiin (grid-based estimation of surface traversability applied to local terrain), jossa mönkijän stereokameroiden ottamista kuvista luodaan malli tämän välittömästä ympäristöstä. Yksi osa mallista on ruudukkopohjainen hyvyyskartta. Kartalla jokainen solu saa sen ajettavuutta kuvaava hyvyysarvon. Hankala maasto tuottaa matalan arvon ja helppo maasto korkean. Maastoa, jota mönkijä ei voi läpäistä levitetään kartalla mönkijän koon verran, jolloin laite voidaan sijoittaa pisteeksi kartalle laskentaa varten.

Kartoituksen jälkeen järjestelmä luo joukon kaaria, joita pitkin kulke- malla voidaan saavuttaa haluttu kohde. Mikäli kaari ei ole suora niin sille lasketaan myös kääntymispisteet. Jokaista kaarta arvioidaan kolmella kriteerillä: vaarojen välttely, kääntymisajan minimointi ja kohteen saavuttaminen. Arvion perusteella sovellus järjestää kaarien välille äänestyksen siten, että hyvyyskartalla turvallisille kaarille annetaan enemmän ääniä kuin turvattomille, kääntymispisteiden lukumäärä korreloi negatiivisesti annettujen äänien kanssa ja mönkijää lähemmäksi kohti kohdetta vievät kaaret saavat enemmän ääniä. Äänet lasketaan tämän jälkeen yhteen ja mönkijä ajaa pienen ennalta määrätyn matkan voittanutta kaarta. Edellä selitettyä prosessia toistetaan kunnes kohde saavutetaan, ennalta määrätty tavoiteaika ylittyy tai tapahtuu virhe.

Ongelmaksi MER-laitteiden järjestelmän ensimmäisessä versiossa muodostui tilanne, jossa mönkijä kohtasi riittävän ison kiviröykkiön. Sen kiertämiseksi olisi jouduttu valitsemaan sellainen kaari, joka ei vie mönkijää riittävän suoraan kohti kohdetta. Suoremmat kaaret taas kulkivat kivien

yli. Tilanne aiheutti äänestysvaiheessa konfliktin, jossa vaaran välttäminen oli liian suuressa konfliktissa suorimman reitin haun kanssa. Järjestelmä ei antanut mönkijän liikkua mihinkään suuntaan.

## 2.4 GESTALT ennen globaalia reitinsuunnittelua

## 2.5 Päivitetty GESTALT

# 3 Field D\* ja lineaarinen interpolointi

## 3.1 Aikaisemmat algoritmit

Useimmissa mobiilirobotiikan navigaatiosovelluksissa ympäristöä mallinetaan ruudukkona, jossa jokainen ruutu saa arvokseen jonkin arvion sen kuljettavuudesta. Tällä tavoin robotti voi tarvittaessa väistää ympäristönsään sijaitsevia esteitä tai liian vaaralliseksi koettuja alueita. Ruudukon lisäksi yleisesti käytetyt algoritmit generoivat myös verkon, jonka solmut sijoitetaan keskelle jokaista ruutua. Verkon kaaret muodostetaan ruudussa sijaitsevan solmun ja tämän naapuriruutuihin asetettujen solmujen välille. Tässä mallissa reitinhaku verkossa voidaan toteuttaa esimerkiksi Anthony Stentzin vuonna 1995 esittelemän D\*-algoritmin avulla [Ste95]. Pääasialliseksi ongelmaksi jää kuitenkin em. kyky löytyä vain sellaiset reitit, joissa robotti liikkuu  $\pi/4$  käännöksillä ruutujen välillä.

## 3.2 Kustannusarvion parantaminen interpoloinnin avulla

Algoritmin perustana toimii metodi, jossa jokaisesta ruudukossa sijaitsevasta solmusta lasketaan halvin mahdollinen kustannusarvio haluttuun kohdepisteeseen [FS07]. Perinteisesti ruudukkopohjaisessa reitinsuunnittelussa on käytetty seuraavaa kaavaa:

$$g(s) = \min_{s' \in nbs(s)} (c(s, s') + g(s')),$$

missä  $nbs(s)$  on joukko kaikista solmun  $s$  naapureista,  $c(s, s')$  on kustannus kulkemiseen kaaren  $s$  ja  $s'$  välillä, sekä  $g(s')$  on kustannusarvio solmulle  $s'$ .

Kaavassa oletetaan, että solmusta  $s$  voidaan siirtyä tämän naapureihin ainoastaan suoraa linjaa pitkin, joka taas johtaa aikaisemmin mainittuun ongelmaan muodostaa parhaita mahdollisia reittejä robotin rajoittuneen suuntaamisen takia. Tämä voitaisiin korjata sijoittamalla  $s'$ :n tilalle  $s_b$ , jossa  $s_b$  on mikä tahansa piste solmuun liittyvän ruudun reunalla. Näitä pisteitä on kuitenkin ääretön määrä, joka tekee jokaisen pisteen laskennasta mahdotonta.

Muokkaamalla verkkoa voimme silti muodostaa approksimaation jokaiselle pisteelle  $s_b$  käyttäen lineaarista interpolointia. Sen sijaan, että solmu sijoitettaisiin ruudun keskelle, asetetaan solmu jokaisen ruudun kulmaan ja täten kaaret kulkevat ruudukon reunoja pitkin. Nyt yhden kaaren kustannus

voidaan valita siten, että se on pienempi niistä kahdesta ruudusta joiden välillä se kulkee.

Tämä muokkaus johtaa siihen, että paras mahdollinen reitti kulkee jotkin kaksi naapurisolmua  $\overrightarrow{s_1 s_2}$  yhdistävän kaaren läpi. Solmulle  $s$  voidaan nyt laskea kustannusarvio, kun reitti kulkee sen ruudun läpi em. kaarelle. Laske- mista varten tarvitaan arviot solmujen  $s_1$  ja  $s_2$  sekä keskimmäisen ruudun  $c$  ja alemman ruudun  $b$  kustannuksista.

Kustannusarvion tuottamiseen käytetään vielä oletusta, että kustannusarvio mille tahansa pisteelle  $s_y$ , joka sijaitsee kaarella  $\overrightarrow{s_1 s_2}$ , on funktioiden  $g(s_1)$  ja  $g(s_2)$  lineaarikombinaatio:

$$g(s_y) = yg(s_2) + (1 - y)g(s_1),$$

missä  $y$  on etäisyys  $s_1$ :stä  $s_y$ :hyn. Tulee kuitenkin huomata, että  $s_y$  ei välttämättä ole em. funktioiden lineaarikombinaatio, mutta tämän oletuksen tuottama approksimaatio toimii käytännössä riittävän hyvin kun halutaan muodostaa suljettu muoto solmun  $s$  kustannusarvion palauttavalle funktiolle.

Approksimaation perusteella solmun  $s$  kustannus kun tiedetään  $s_1$ ,  $s_2$ , ruutukustannukset  $c$  ja  $b$  voidaan laskea seuraavasti:

$$\min_{x,y} [bx + c\sqrt{(1-x)^2 + y^2} + g(s_2)y + g(s_1)(1-y)],$$

missä  $x \in [0, 1]$  on solmusta  $s$  alareunaa pitkin kuljettu matka kunnes käännytään ruudun yli kohti oikeaa reunaa pisteeseen, joka on  $y \in [0, 1]$  etäisyyden päässä solmusta  $s_1$ .

Tehdään vielä oletus, että  $(x^*, y^*)$  ovat  $x$ :n ja  $y$ :n arvot, joilla ylläoleva kaava saadaan ratkaistua. Lineaarisen interpoloinnin johdosta toinen arvoista on joko yksi tai nolla. Mikäli kustannus liikkua ruudun  $c$  yli on pienempi kuin ruudun reunoja pitkin kulkeminen niin halvin reitti halkaisee ruudun  $c$  ja täten joko  $x^* = 0$  tai  $y^* = 1$ . Jos taas polku ei halkaise ruutua  $c$  niin  $y^* = 0$ . Täten polku on kulkee solmusta  $s$  suoraan alareunaa pitkin kohtia solmua  $s_1$ , siirtyy jonkin matkan  $x$  alareunalla ja leikkaa tämän jälkeen ruudun halki suoraan solmuun  $s_2$ , tai halkaisee ruudun  $c$  kulkemalla suoraan solmusta  $s$  johonkin oikean reunan pisteeseen  $s_y$ . Halvin polku riippuu  $c$ :n ja  $b$ :n koosta, sekä  $s_1$ :n ja  $s_2$ :n kustannuserosta  $f = g(s_1) - g(s_2)$ . Mikäli  $f < 0$  niin paras mahdollinen polku on 1. tapaus, jos taas  $f = b$  niin polun kustannus kulkien jonkin matkan alareunaa on yhtäpitävä sen kanssa, että alareunaa ei kuljeta ollenkaan. Jälkimmäisestä polusta voidaan ratkaista kustannuksen minimoiva  $y$  seuraavasti.

Olkoon  $k = f = b$ . Kustannus solmusta  $s$  kaaren  $\overrightarrow{s_1 s_2}$  lävitse on

$$c\sqrt{1 + y^2} + k(1 - y) + g(s_2).$$

Jossa kustannuksen derivaatasta suhteessa  $y$ :hyn ja asettamalla se nolllaksi

saadaan

$$y^* = \sqrt{\frac{k^2}{(c^2 - k^2)}}$$

Lopputulos on sama huolimatta siitä kuljetaanko alareunaa pitkin, joten merkitseväksi tekijäksi jää se kumpaa reunaa kulkeminen tulee halvemmaksi. Mikäli  $f < b$  niin käytetään oikeaa reunaa ja lasketaan polun kustannus arvolla  $k = f$ . Jos taas  $b < f$  niin käytetään alareunaa jolloin  $k = b$  ja  $y^* = 1 - x^*$ . Täten algoritmi halvimman polun laskemiseen solmusta  $s$  mihin tahansa pisteeseen kaarelle, joka sijaitsee vierekkäisten naapurien  $s_a$  ja  $s_b$  välissä laskemiseen on seuraava:

```

ComputeCost( $s, s_a, s_b$ )
  if ( $s_a$  on solmun  $s$  diagonaalinaapuri)
     $s_1 = s_b$ ;
     $s_2 = s_a$ ;
  else
     $s_1 = s_a$ ;
     $s_2 = s_b$ ;

   $c$  on kustannus ruudulle, jonka kulmat ovat  $s, s_1, s_2$ ;
   $b$  on kustannus ruudulle, jonka kulmat ovat  $s, s_1$ , mutta ei  $s_2$ ;

  if ( $\min(c, b) = \infty$ )
     $v_s = \infty$ ;
  else if ( $g(s_1) \leq g(s_2)$ )
     $v_s = \min(c, b) + g(s_1)$ ;
  else
     $f = g(s_1) - g(s_2)$ ;
    if ( $f \leq b$ )
      if ( $c \leq f$ )
         $v_s = c\sqrt{2} + g(s_2)$ ;
      else
         $y = \min(\frac{f}{\sqrt{c^2 - f^2}}, 1)$ ;
         $v_s = c\sqrt{1 + y^2} + f(1 - y) + g(s_2)$ ;
    else
      if ( $c \leq b$ )
         $v_s = c\sqrt{2} + g(s_2)$ ;
      else
         $x = 1 - \min(\frac{b}{\sqrt{c^2 - b^2}}, 1)$ ;
         $v_s = c\sqrt{1 + (1 - x)^2} + bx + g(s_2)$ ;

  return  $v_s$ ;

```

### 3.3 Field D\*

Seuraava algoritmi on optimoimaton esimerkki Field D\*:in toteutuksesta [FS07], joka pohjautuu aikaisempaan D\* Lite-algoritmiin [KL02]. Sen tarkoituksena on koota ylempänä esitellyn interpoloinnin avulla lasketut kustannukset yhteen ja muodostaa edullisin reitti kahden pisteen välillä. Algoritmi on toteutettu siten, että se ottaa huomioon muutokset ympäristössä, joita voi syntyä esimerkiksi sään muuttumisen seurauksena.

```

key( $s$ )
  return [ $\min(g(s), rhs(s)) + h(s_{start}, s)$ ;  $\min(g(s), rhs(s))$ ];

UpdateState( $s$ )
  if solmussa  $s$  ei ole vielä kayty
     $g(s) = \infty$ ;
  if ( $s \neq s_{goal}$ )
     $rhs(s) = \min_{(s', s'') \in connbrs(s)} ComputeCost(s, s', s'')$ ;
  if ( $s \in OPEN$ )

```



```

    OPEN.remove(s);
    if (g(s) ≠ rhs(s))
        OPEN.insert(key(s) : s)

ComputeShortestsPath ()
    while (mins ∈ OPEN (key(s)) < key(sstart) || rhs(sstart) ≠ g(sstart))
        OPEN.remove(smallest(s));
        if (g(s) > rhs(s))
            g(s) = rhs(s);
            for all s' ∈ nrb(s) UpdateState(s');
        else
            g(s) = ∞;
            for all s' ∈ nrb(s) UpdateState(s');

Main ()
    g(sstart) = rhs(sstart) = ∞;
    g(sgoal) = ∞;
    OPEN.insert(key(sgoal) : sgoal)
    while true
        ComputeShortestPath ();
        Odota muutoksia ruutujen kustannusarvioissa
        for x in muuttuneet_ruudut
            for s in x
                UpdateState(s);

```

Pseudokoodissa *connbrs(s)* on joukko solmun  $s$  perättäisiä naapuripareja:  $(s_1, s_2), (s_2, s_3), (s_3, s_4), (s_4, s_5), (s_5, s_6), (s_6, s_7), (s_7, s_8), (s_8, s_1)$ .  $g(s)$  on solmun  $s$  kustannusarvio,  $rhs(s)$  kertoo arvion solmun  $s$  kustannuksesta yhden askeleen päästä, *OPEN* toimii prioriteettijonona nousevassa järjestyksessä solmuille, joille  $g(s) \neq rhs(s)$ .  $h(s_{start}, s)$  on heuristinen arvio kustannuksesta  $s_{start}$  solmusta solmuun  $s$ .

## 4 Visuaalinen paikannus

Yksi mönkijöiden haasteellisimmista tehtävistä on jatkuvasti ylläpitää tietoa tämän omasta paikasta ja asennosta [MCM07]. Suunnitteluvaiheessa tavoitteeksi asetettiin maksimissaan kymmenen porsentin virhemarginaali sadan metrin matkalla. Tehtävää varten niihin on sijoitettu inertian mittaussyksikkö, joka yhdessä rengasodometrian kanssa pyrkii päättelemään kuinka pitkän matkan mönkijä on liikkunut ja missä asennossa se kullakin hetkellä on. Tämä menetelmä toimii hyvin tasaisella sekä hyvälaatuisella maastolla saavuttaen sille asetetun tavoitteen.

Ongelmaksi muodostuvat kuitenkin tilanteet, joissa maaston pito pienenee huomattavasti kuten esimerkiksi hiekka-aavikolla. Myös kraaterien kaltaiset kaltevat alueet tuottivat hankaluuksia inertian mittaussyksikölle ylläpitää tietoa mönkijän paikasta. Näissä tilanteissa avuksi otetaan visuaalinen odometria, jossa kahdesta eri hetkenä otetusta 256x256 stereokuvaparista tunnistetaan ympäristön piirteitä ja pyritään havaittujen erojen avulla arvioimaan mönkijälle tarkemmat arvot tämän kuudelle vapausasteelle. Tämä ominaisuus kuitenkin hidastaa mönkijän liikkumista yhtä suuruusluokkaa pienemmäksi, joten se kytketään päälle ainoastaan tarvittaessa.

## 4.1 Algoritmi

### Lähteet

- [FS07] Dave Ferguson and Anthony Stentz. Field d\*: An interpolation-based path planner and replanner. In *Robotics Research*, pages 239–253. Springer, 2007.
- [KL02] Sven Koenig and Maxim Likhachev. D\* lite. In *AAAI/IAAI*, pages 476–483, 2002.
- [MCM07] Mark Maimone, Yang Cheng, and Larry Matthies. Two years of visual odometry on the mars exploration rovers. *Journal of Field Robotics*, 24(3):169–186, 2007.
- [Ste95] Anthony Stentz. The focussed d<sup>\*</sup> algorithm for real-time replanning. In *IJCAI*, volume 95, pages 1652–1659, 1995.