Mars-luotainten reitinhakualgoritmit

Jerry Mesimäki

Kandidaatintutkielma
HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Helsinki, 19. lokakuuta 2014

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Laitos — Institution — Department
Matemaattis-luonnontieteellinen	Tietojenkäsittelytieteen laitos

Tekijä — Författare — Author
Jerry Mesimäki

Työn nimi — Arbetets titel — Title

Mars-luotainten reitinhakualgoritmit

Oppiaine — Läroämne — Subject

Tietojenkäsittelytiede

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Kandidaatintutkielma	19. lokakuuta 2014	4

Tiivistelmä — Referat — Abstract

Tiivistelmä.

Avainsanat — Nyckelord — Keywords
avainsana 1, avainsana 2, avainsana 3

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — Övriga uppgifter — Additional information

Sisältö

1 Ensimmäinen luku

2 Field D* ja lineaarinen interpolointi

2.1 Aikaisemmat algoritmit

Useimmissa mobiilirobotiikan navigaatiosovelluksissa ympäristöä mallinnetaan ruudukkona, jossa jokainen ruutu saa arvokseen jonkin arvion sen kuljettavuudesta. Tällä tavoin robotti voi tarvittaessa väistää ympäristössään 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 löytyää vain sellaiset reitit, joissa robotti liikkuu $\pi/4$ käännöksillä ruutujen välillä.

2.2 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:

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

missä $nbrs(s)$ 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. Laskemista varten tarvitaan arviot solmujen $s1$ ja $s2$ 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 $g(s1)$ ja $g(s2)$ 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 käytännässö riittävän hyvin kun halutaan muodostaa suljettu muoto solmun s kustannusarvion palauttavalle funktiolle.

Approksimaation perusteella solmun s kustannus kun tiedetään $s1$, $s2$, 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 $s1$, siirtyy jonkin matkan x alareunalla ja leikkaa tämän jälkeen ruudun halki suoraan solmuun $s2$, 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

$$c\sqrt{1+y^2} + k(1-y) + g(s2).$$

Jossa kustannuksen derivaatasta suhteessa y:hyn ja asettamalla se nollaksi 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) = ∞)
        v_s = ∞;
    else if (g(s_1) ≤ g(s_2))
        v_s = min(c, b) + g(s_1);
    else
        f = g(s_1) − g(s_2);
        if (f ≤ b)
            if (c ≤ f)
                v_s = c√2 + g(s_2);
            else
                y = min(f / √(c^2 − f^2), 1);
                v_s = c√(1 + y^2) + f(1 − y) + g(s_2);
        else
            if(c ≤ b)
                v_s = c√2 + g(s_2);
            else
                x = 1 − min(b / √(c^2 − b^2), 1);
                v_s = c√(1 + (1 − x)^2) + bx + g(s_2);
    return v_s;
```

2.3 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 viela kayty
        g(s) = ∞;
    if (s ≠ s_goal)
        rhs(s) = min_{(s', s'') ∈ connbrs(s)} ComputeCost(s, s', s'');
    if (s ∈ OPEN)
        OPEN.remove(s);
    if (g(s) ≠ rhs(s))
        OPEN.insert(key(s) : s)

ComputeShortestsPath()
    while (min_{s ∈ OPEN}(key(s)) < key(s_start) || rhs(s_start) ≠ g(s_start))
        OPEN.remove(smallest(s));
        if (g(s) > rhs(s))
            g(s) = rhs(s);
            for all s' ∈ nrbs(s) UpdateState(s');
        else
            g(s) = ∞;
```

```
        for all s' ∈ nbrs(s)  UpdateState(s');

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

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.

[Ste95] Anthony Stentz. The focussed dˆ* algorithm for real-time replanning. In *IJCAI*, volume 95, pages 1652–1659, 1995.