برنامه ریزی مسیر بهینه و کارا برای محیط های تا حدی نا شناخته

آنتونی استنز

موسسه رباتیک ؛ دانشگاه کارنگی ملون ؛ پیتزبورگ

ترجمه: اکتای حسن زاده

دانشکده مهندسی کامپیوتر ؛ دانشگاه صنعتی شریف

چکیده

وظیفه برنامه ریزی مسیر برای یک ربات متحرک توجه بسیاری از نوشته هـای تحقیقـاتی را بـه خود جلب کرده است. در این نوشته ها، بیشتر فرض براین است که ربات قبل از حرکت یک مدل کامل و دقیق از محیط اطراف خود دارد. کمتر به مساله محیط های تا حدی ناشـناخته توجـه شـده است. این موقعیت برای یک ربات کاوشگر یا رباتی که باید به سوی یک هدف بدون داشتن نقشـه ناحیه، حرکت کند پیش می آید. در خط مشی های کنونی، براساس اطلاعات موجود، یک مسیر را برنامه ریزی می کنند سپس باقربانی کردن بهینگی و کارآیی محاسباتی، ضمن اینکه ربات به کمـک حسگرها موانع را می یابد، مسیر را به طور موضعی تغییر می دهند یـا اینکـه کـل مسـیر را دوبـاره برنامه ریزی می کنند. این مقاله یک الگوریتم جدید به نام D* را معرفـی مـی کنـد، الگـوریتمی بـا قابلیت برنامه ریزی مسیر در محیط های ناشناخته، تا حدی ناشناخته و محیط های متغیر به صورت بهینه ، کامل و کارا.

۱ - معرفی

نوشته های تحقیقاتی به صورت گسترده مساله برنامه ریزی حرکت برای یک یا چند ربات با عبور از یک میدان از موانع به سوی یک هدف را مورد بررسی قرار داده اند. اکثر این نوشته ها فرض می‌کنند که محیط قبل از شروع حرکت ربات به طور کامل شناخته شده است (برای یک بررسی خوب به مرجع ۴ [Latombe] رجوع کنید). الگوریتم بهینه در این نوشته ها یک فضای حالت را جستجو می کند (مثلاً گراف میدان دید یا سلولهای مشبک) و با بهره گیری از تبدیل فاصله [۲] یا مکاشفه [۸]، کم هزینه ترین مسیر را از حالت شروع ربات تا حالت هدف بیابند. هزینه می‌تواند به صورت فاصله طی شده، انرژی مصرف شده، زمان قرار گیری را در معرض خطر،... تعریف شود.

متاسفانه، ربات ممکن است اطلاعات جزئی در مورد محیط داشته باشد یا اینکه از محیط قبل از شروع حرکت بی اطلاع شد، اما به یک حسگر که قابلیت سنجیدن محیط راهنگام حرکت دارد مجهز شده است، یک رویکرد به برنامه ریزی مسیر در این سناریو این است که یک مسیر "سراسری" با استفاده از اطلاعات موجود، ایجاد کنیم و سپس سعی کنیم به صورت "موضعی" از موانع موجود در مسیرکه با حسگرها کشف شده اند گذر کنیم [۱]. اگر مسیر به طور کامل مسدود بود، یک مسیر سراسری جدید برنامه ریزی می شود. لوملسکی[۱] فرض می کند که در ابتدا محیط عاری از موانع است و ربات را مستقیماً به سوی هدف حرکت می دهد. اگر یک مانع مسیر را سد

[۱] Lumelskey

کند، ربات در پیرامون آن حرکت می کند تا نزدیکترین نقطه به هدف پیدا شود، سپس ربات دوباره به حرکت مستقیم به سوی هدف ادامه می دهد، پیرزاده [۹] یک استراتژی اتخاذ می کند که به موجب آن ربات در محیط سرگردان است تا به هدف برسد. ربات مکرراً به سوی مکان مجاور با کمترین هزینه حرکت می کند و هزینه هر مکان را با ملاقات آن افزایش می دهد تا حرکت بعدی بر روی همان فضا، جریمه شود.

کُرف[۱] از اطلاعات نقشه در ابتدا استفاده می کند تا هزینه رسیدن به هدف را در هر حالت تخمین بزند و سپس آن را همین طور که ربات در محیط حرکت می کند به صورت کارا و با پس گرد کردن هزینه ها به روز می کند.

با وجود اینکه این رویکردها کامل هستند، اما آنها نیمه بهینه اند از این لحاظ که آنها با داشتن اطلاعات حسگرها، آن طور که به دست آمده، کم هزینه ترین مسیر را ایجاد نمی کنند و فرض می‌کنند تمام اطلاعاتی که از پیش داشته اند صحیح است. این امکان وجود دارد که یک رفتار بهینه ایجاد کنیم به این صورت که یک مسیر بهینه با اطلاعات فعلی نقشه محاسبه کنیم،ربات را در طول مسیر حرکت دهیم تا به هدف برسد یا اینکه یک اختلاف بین محیط و اطلاعات نقشه پیدا کند، اطلاعات نقشه را درست کند و یک مسیر بهینه جدید تا هدف را مجدداً برنامه ریزی کند. با وجود اینکه این رویکرد باز برنامه ریزی[۱]، بهینه است، اما می تواند به طور فاحشی ناکارا باشد، به طور خاص در محیط های وسیع که هدف بسیار دور است، اطلاعات نقشه‌ای کمی موجود است.

---

[۱] Korf

زلینسکی[۲] با استفاده از یک درخت چهارگانه[۳] [۱۳] برای نمایش فضای آزاد و موانع، کارایی را بهبود بخشید، بدین گونه تعداد حالات را برای جستجو در فضای برنامه‌ریزی کاهش داد. البته برای زمین طبیعی نقشه می تواند نقاط قابل عبور توسط ربات را در هر محل در یک محدوده پیوسته کدگذاری کند، پس ارائه درخت چهارگانه نامناسب یا نیمه بهینه است.

این مقاله یک الگوریتم جدید برای ایجاد مسیرهای بهینه برای یک ربات که با یک حسگر و یک نقشه از محیط کار می کند، ارائه می دهد. نقشه می تواند کامل باشد، خالی باشد یا اینکه اطلاعات جزئی درباره محیط داشته باشد. برای آن قسمت از محیط که ناشناخته است، نقشه می‌تواند حاوی اطلاعات تقریبی، مدل های تصادفی برای سکونت، یا حتی یک تخمین باشد. الگوریتم از لحاظ کارکرد معادل باز برنامه ریزی brufe-force است اما بسیار کاراتر از آن است.

الگورتیم به صورت یک مساله پیدا کردن مسیر بهینه در یک گراف جهت دار تنظیم شده است، که یالها با مقادیرهزینه برچسب گذاری شده اند و می توانند در یک محدوده پیوسته باشند. حسگر ربات می تواند هزینه بالهای در مجاورت یک ربات را اندازه گیری کند، و مقادیر معلوم و تخمین زده شده نقشه راتشکیل می دهند. سپس، الگوریتم می تواند برای هر نوع ارائه از برنامه ریزی، از جمله گرافهای میدان دید [۵] و ساختارهای سلولی مشبک مورد استفاده قرار گیرد. این مقاله الگوریتم را شرح می دهد، عمل آن را به تصویری می کشد و اثباتهای غیردقیق از صحت، بهینگی

---

[۱] Brute force

[۲] Zelinsky

[۳] Quad tree

و کامل بودن آن ارائه می دهد و سپس با یک مقایسه تجربی الگوریتم با باز برنامه ریـزی بهینـه بـه پایان می‌رسد.

۲ - الگوریتم *D

نام الگوریتم، *D، به خاطر شباهت آن به *A [۸] انتخاب شده است، فقط ایـن الگـوریتم پویـا (دینامیک) است از این لحاظ که پارامترهای هزینه یالها در حین پردازه حل مساله می تواننـد تغییـر کنند با دانستن اینکه حرکت ربات به طور صحیح به الگوریتم وابسته است، *D مسیرهای بهینـه را ایجاد می کند. این بخش با تعاریف و معرفی نماد گذاریهای استفاده شـده در الگـوریتم آغـاز می شود، الگوریتم *D ارائه می شود. و با به تصویر کشیدن عمل آن به پایان می رسد.

۲ - ۱ - تعاریف

هدف یک برنامه ریز مسیر این است که ربات را از یک محل در جهان بـه سـوی محـل مقصـد حرکت دهد، به طوریکه از تمام موانع اجتناب کند و یک معیار هزینه ها مثبت (مثلاً طول مسیر) را کمینه کند. فضای مساله می تواند به صورت مجموعه ای از "حالات" تنظیم شود که مشخص کننده محل های قرار گیری ربات هستند که با "یال های جهت دار" متصلند، کـه هـر کـدام یـک هزینـه وابسته به خود دارند. ربات در یک حالت خاص از مجموعه حالات شروع می کند ودر طول یالهـا به سوی حالات دیگر حرکت می کند (که هزینه عبور را موجب مـی شـوند) تـا اینکـه بـه حالـت "هدف" می رسند، که با G نشان داده می شود. هر حالت X به جز G یک "پشـت- اشـاره‌گـر" بـه حالت بعدی Y دارد که با $b(X)=Y$ نمایش داده می شود. *D از پشت اشـاره گرهـا بـرای نمـایش

مسیرها به هدف استفاده می کند. هزینه عبور از یک یال از حالت Y به حالت X یک عدد مثبت است که با تابع "هزینه یال"، C(X,Y) داده می شود. اگر Y یالی به X ندارد، C(X,Y) تعریف نشده است. دو حالت Y,X در فضا "همسایه" اند اگر C(X,Y) و یا C(X,Y) تعریف شده باشد.

مانند A* , D* یک لیست OPEN از حالات را نگهداری می کند از لیست OPEN برای انتشار اطلاعات درباره تغییرات در تابع هزینه و نیز برای محاسبه هزینه های مسیر به حالات در یک فضا استفاده می شود. هر حالت X دارای یک برچسب t(X) است به طوریکه t(X)=NEW هرگاه X هرگز در لیست OPEN نبوده است، t(X)=OPEN اگر X در حال حاضردر لیست OPEN است و t(X) = CLOSED هرگاه X دیگر در لیست OPEN قرار نداشته باشد.برای هر حالت X، D* یک تقریب از جمع هزینه ها از X به G در تابع "هزینه مسیر" h(G,X) نگهداری می کند. با داشتن شرایط مناسب، این تقریب معادل هزینه بهینه (کمینه) از حالت X به G است که با تابع ضمنی o(G,X) داده می شود. برای هر حالت X در لیست OPEN (یعنی t(X)= OPEN)، تابع "کلید" k(G,X) برابر مینیمم h(G,X) قبل از تغییرات و تمام مقادیری است که h(G,X) تا زمانی که X برروی لیست OPEN قرار داده شود ،میگیرد. تابع کلید یک حالت X بر روی لیست OPEN را به یکی از دو نوع ذیل دسته بندی می کند.

یک حالت RAISE است اگر k(G,X)<h(G,X) و LOWER است اگر k(G,X)=h(G,X). D* از حالتهای RAISE در لیست OPEN برای انتشار اطلاعات درباره افزایشهای هزینه مسیر مثلاً به خاطر افزایش هزینه یک یال و از حالتهای LOWER برای انتشار اطلاعات درباره کاهشهای هزینه

ها (مثلاً به خاطر کاهش یافتن هزینه یک یال یا مسیر جدید به هدف) استفاده می کند این انتشار در زمان حذف مکرر حالات از لیست OPEN رخ می دهد. هر بار یک حالت از لیست حذف می شود عموماً توسعه می یابد تا تغییرات هزینه را به همسایه هایش منتقل کند این همسایه ها به نوبت در لیست OPEN قرار می گیرند تا روند ادامه یابد.

حالت ها در لیست OPEN براساس مقدار تابع کلیدشان مرتب شده اند. پارامتر X_{min} برابر $min(k(X))$ به ازای تمام X هایی که $t(X)=OPEN$ باشد تعریف می شود، پارامتر k_{min} یک آستانه مهم را در D* نشان می دهد: مسیرهایی که کمتر یا مساوی k_{min} هزینه دارند بهینه اند و مسیرهایی که بیش از k_{min} هزینه دارند ممکن است بهینه نباشند. پارامتر k_{old} برابر k_{min} قبل از آخرین حذف یک حالت از لیست OPEN تعریف می شود. اگر هیچ حالتی حذف نشده باشد k_{old} تعریف نشده است.

یک ترتیب از حالات که با $\{X_1,X_n\}$ نمایش داده می شود به صورت یک "دنباله" تعریف می‌شود که در آن $b(X_{i+1})=X_i$ برای هر i که $1<i<N$ و $x_i \neq x_j$ برای (i,j) که $1 \le i \le j \le N$. بدین گونه یک دنباله یک مسیر از پیش اشاره گرها از X_N تا X_1 را مشخص می کند. یک دنباله $\{X_1,X_N\}$ "یکنوا" است هرگاه $(t(X_i)=CLOSED,h(G,X_i)<h(G,X_i))$ برای هر i که $1 \le i < N$. D* یک دنباله یکنوای $\{G,X\}$ را می سازد و نگهداری می کند، که هزینه مسیرهای کاهش یافته محلی یا کران پایینی را، برای هر حالت X که در لیست OPEN وجود دارد یا وجود داشته، نمایش می دهد. با داشتن یک دنباله از حالتهای $\{X_1,X_N\}$، حالت X_i یک "جد" حالت X_j است هرگاه

$N \geq j \geq i \geq 1$ و یک "نوه" حالت X_j است اگر $N \geq t > j \geq 1$.

برای همه توابع دو حالته که شامل حالت هدف هستند، نماد خلاصه F(X)=F(G,X) استفاده می‌شود. به طور مشابه برای دنباله ها نماد {X}={G,X} استفاده می شود. نماد f(°) برای نمایش یک تابع مستقل از دامنه اش استفاده می شود.

۲ – ۲ – *توصیف الگوریتم*

الگوریتم *D به طور عمده از دو تابع تشکیل شده است:PROCESS-STATE و MODIFY-COST تابع PROCESS-STATE برای محاسبه هزینه مسیرها به هدف استفاده می‌شود، و تابع MODIFY-COST برای عوض کردن تابع هزینه C(°) و وارد کردن حالات مورد تاثیر در لیست OPEN استفاده می شود.در ابتدا t(°) برای همه حالات برابر NEW قرار داده می‌شده h(G) صفر می شود و G در لیست OPEN قرار داده می شود. تابع اول PROCESS-STATE به طور مکرر و اخوانی می شود تا اینکه حالت ربات، X، از لیست OPEN حذف شود (یعنی t(X)=CLOSED) یا اینکه مقدار ۱- بازگردانده شود که دراین نقطه، به ترتیب یا دنباله {X} ساخته شده است و یا چنین دنباله ای موجود نیست. سپس ربات پیش می رود و پیش اشاره گرها را دنبال می کند تا اینکه یا به هدف می رسد یا یک خط در تابع هزینه یال C(°) کشف می کند (مثلاً به دلیل پیدا کردن یک مانع). تابع دوم ، MODIFY-COST، بی درنگ فراخوانی می شود تا تابع هزینه C(°) را صحیح کند و حالات متاثر را در لیست OPEN قرار دهد. فرض کنید Y حالتی باشد که در آن ربات خطایی در C(°) را می یابد. با فراخوانی PROCESS-

STATE تا برگرداندن $k_{min} \geq h(y)$، تعویض هزینه ها به حالت Y انتشار مـی یابـد، بـه طوریکـه h(Y)=0(Y). در این نقطه، یک دنباله احتمالاً جدید {Y} ساخته شده است، و ربـات دنبـال کـردن پیش اشاره گرها در دنباله به سوی هدف را ادامه می دهد.

الگوریتم های PROCESS-STAT و MODIFY-COST در زیرنمایش داده شده اند. روال‌های گنجانده شده عبارتند از: MIN-STATE که حالتی در لیست OPEN را بر مـی گردانـد کـه مقـدار k(°) می نیمم دارد. (اگر لیست خالی است، NULL برمی گرداند) GET-KMIN که k_{min} رابـرای لیست OPEN بر می گرداند (۱- اگر لیست خالی است)؛ DELETE (X) که حالت X را از لیست OPEN حـذف مـی کنـد و t(X) را برابـر CLOSE قـرار مـی دهـد و INSERT (Y,h_{new}) کـه $k(X)=h_{new}$ را محاسبه می کند، اگـر $k(X)=min(k(X),h_{new})$, t(X)=NEW را اگـر t(X)=OPEN باشد $k(X)=min(k(X),h_{new})$ را اگر t(X)=CLOSED، قرارمی دهد $h(X)=h_{new}$ و t(X)=OIPEN و حالت X را بر روی لیست OPEN مرتب شده براساس k(°) قرار می دهد.

Function: PROCESS-STATE ()

```
L1   X = MIN-STATE( )
L2   if X = NULL then return -1
L3   k_old = GET-KMIN( ); DELETE(X)
L4   if k_old < h(X) then
L5     for each neighbor Y of X:
L6       if h(Y) ≤ k_old and h(X) > h(Y) + c(Y, X) then
L7         b(X) = Y; h(X) = h(Y) + c(Y, X)
L8   if k_old = h(X) then
L9     for each neighbor Y of X:
L10      if t(Y) = NEW or
L11        (b(Y) = X and h(Y) ≠ h(X) + c(X, Y)) or
L12        (b(Y) ≠ X and h(Y) > h(X) + c(X, Y)) then
L13        b(Y) = X; INSERT(Y, h(X) + c(X, Y))
L14  else
L15    for each neighbor Y of X:
L16      if t(Y) = NEW or
L17        (b(Y) = X and h(Y) ≠ h(X) + c(X, Y)) then
L18        b(Y) = X; INSERT(Y, h(X) + c(X, Y))
L19      else
L20        if b(Y) ≠ X and h(Y) > h(X) + c(X, Y) then
L21          INSERT(X, h(X))
L22        else
L23          if b(Y) ≠ X and h(X) > h(Y) + c(Y, X) and
L24            t(Y) = CLOSED and h(Y) > k_old then
L25            INSERT(Y, h(Y))
L26  return GET-KMIN( )
```

در تابع PROCESS-STATE در خطوط L1 تا L3، حالت X با کوچک تـرین مقـدار (°)k از لیست OPEN حذف می شود. اگر حالت X از نـوع LOWER باشـد (یعنـی k(X)=h(X)) هزینـه مسیر آن بهینه است زیرا h(X) برابر k_{min} قدیمی است. در خطوط L8 تا L13 هر همسـایه Y از X آزمایش می شود که آیا هزینه مسیر آن می تواند کم شود یا نه. علاوه بر این، حالت هـای همسـایه که NEW هستند یک هزینه مسیر اولیه می گیرند و تغییرات هزینه به هر همسایه Y که یـک پـیش اشاره گر به X دارد انتشار می یابد، بدون در نظرگرفتن اینکه هزینه جدیـد بزرگتـر یـا کـوچکتر از

قبلی است. از آنجا که این حالتها از نوه های X هستند، هر تغییری در هزینه مسیر X بر هزینه مسیر آنها نیز تاثیر می گذارد. پیش اشاره گر Y تغییر جهت داده می شود (در صورت لزوم) تا دنباله یکنوای {Y} ساخته شود. تمامی همسایه هایی که یک هزینه مسیرتازه دریافت می کنند در لیست OPEN قرارداده می شوند، تا آنها تغییر هزینه ها را به همسایه هایشان انتقال دهند.

اگر X یک حالت از نوع RAISE است، هزینه مسیر آن شاید بهینه نباشد. قبل از اینکه X تغییرات هزینه را به همسایه‌هایش انتشاردهد. همسایه‌های بهینه اش در خطوط L4 تا L7 سنجیده می‌شوند تا ببنیم که h(X) می تواند کم شود یا نه. در خطوط L15 تا L18، تغییرات هزینه به حالتهای NEW و نوه های مستقیم، انتشار می یابد، به همان روشی که در مورد حالتهای LOWER عمل می شود. اگر X قادر باشد که هزینه مسیر یک حالت را که نوه مستقیم نیست را کاهش دهد (خطوط L20 و L21)، X برای توسعه آینده باز بر روی لیست OPEN قرار داده می‌شود. در بخش بعد نشان داده می شود که این عمل برای جلوگیری از به وجود آمدن یک حلقه بسته در پیش اشاره گرها ضروری است. اگر هزینه مسیر X می تواند توسط یک همسایه نیمه کارا کم شود (خطوط L23 تا L25)، همسایه بر روی لیست OPEN مجددا قرار داده می شود. بدین گونه، به روز رسانی "به تعویق" می افتد تازمانیکه همسایه یک هزینه مسیر بهینه داشته باشد.

در تابع MODIFY-COST، تابع هزینه یال با مقدار تغییر یافته، به روز می شود.

از آنجا که هزینه مسیر برای حالت Y عوض خواهد شد، X در لیست OPEN قرار داده می‌شود. وقتی X توسط PROCESS-STATE توسعه می یابد، آن یک h(Y)=h(X)+c(X,Y)

جدید محاسبه می‌کند و Y را در لیست OPEN قرار می دهد. توسعه حالت هـای دیگـر، هزینـه را به نوه‌های Y انتشار می دهد.

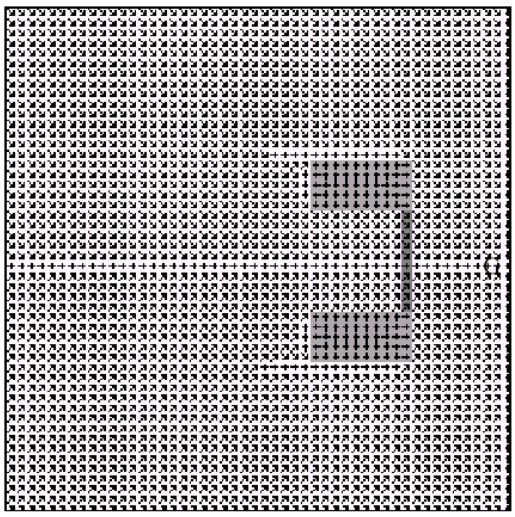
Function: MODIFY-COST (X, Y, cval)
L1 $c(X, Y) = cval$
L2 if $t(X) = CLOSED$ then $INSERT(X, h(X))$
L3 return $GET-KMIN( )$

۲ - ۳ - نمایش عملکرد الگوریتم

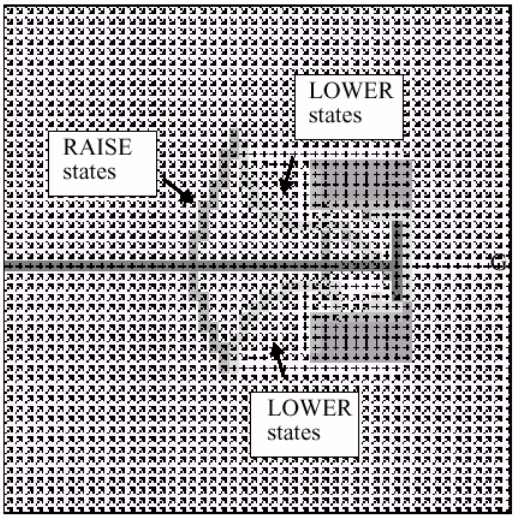
نقش حالتهای RAISE و LOWER در عملکرد الگوریتم اساسی اسـت.حالتهـای نـوع RAISE (یعنـی $k(X)<h(X)$) افـزایش هـای هزینـه را انتشـار میدهنـد و حالتهـای نـوع LOWER (یعنـی $k(X)=h(X)$) کاهش های هزینه راانتشار می دهند. زمانیکه هزینه عبور از یک یال افزایش می‌یابـد، یک همسایه تاثیر پذیرفته در لیست OPEN قرار داده می‌شـود، و افـزایش هزینـه توسـط حالتهـای RAISE در تمام دنباله حالات شامل یال انتشار می یابد. همین طور که حالتهای RAISE با حالات همسایه با هزینه کمتر تماس پیدا می کنـد، ایـن حالتهـای LOWER در لیسـت OPEN قـرار داده می‌شوند، و متعاقباً هزینه حالتهای قبلاً افزایش یافته را هر جا که ممکن باشد کاهش می دهـد. اگـر هزینه عبور از یک یال کاهش یابد، این کاهش ازطریق حالات LOWER به تمام دنباله های شـامل آن یال منتشر می شود، همچنین به حالت های همسایه که هزینه آنها نیز می تواند کاهش یابد.

شکل های ۱ تا ۳، عملکرد الگوریتم را برای یـک مسـاله "بـالقوه خـوب" برنامـه ریـزی مسـیر نمایش می دهد. فضای برتامه ریزی از "سلّولهای" مشبّک ۵۰ × ۵۰ تشکیل شده اسـت. هـر سـلول یک حالت را نمایش می دهد و توسط یالهای دو طرفه به ۸ همسایه اش متصل شده اسـت. هزینـه

یالها برای سلولهای "خالی" کوچک است و ناگزیر برای سلولهای "مانع" بـزرگ اسـت[1]. ربـات بـه اندازه یک نقطه است و به یک حسگر تماسی مجهز شده است. شکل ۱ نتیجه هـای یـک محاسـبه مسیر از هدف به تمام حالا در فضای برنامه ریزی را نشان می دهـد. دو مـانع خاکسـتری در نقشـه ذخیره شده‌اند، اما مانع سیاه ذخیره نشده است. فلش ها، تابع پیش اشاره گـر را ترسـیم کـرده انـد. پس یک مسیر بهینه به سوی هدف می تواند با دنبال کردن فلش ها از هر حالت به هدف به دسـت آید. توجه کنید که فلش ها کنارموانع معلوم خاکستری منحرف می شوند اما از میان مانع ناشـناخته سیاه می گذرند.



شکل ۱ – پیش اشاره گرها بر اساس انتشار اوّلیه

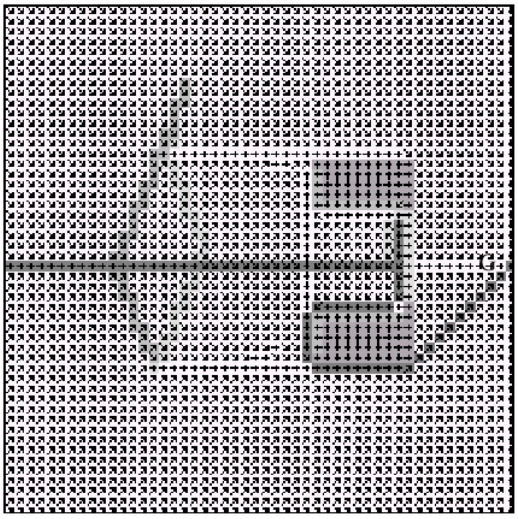


شکل ۲ – حالتهای LOWER چاه را جاروب می کنند

ربات در مرکز دیوار چپ شروع به حرکت می کند و پیش اشاره گرها را به سوی هـدف دنبـال می کند. هنگامی که به یک مانع ناشناخته می رسد، بین نقشه و محیط، اختلاف می یابد، نقشه رابـه

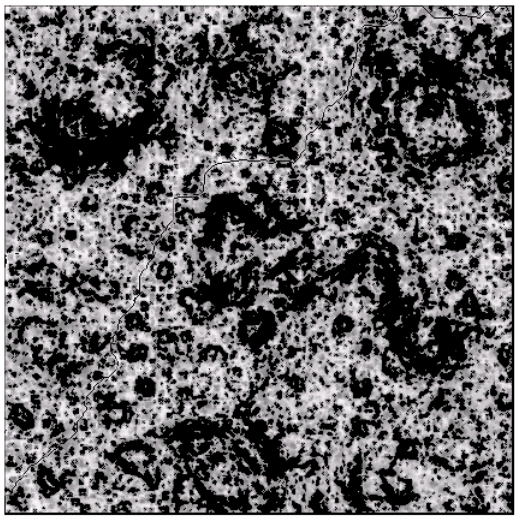
[1] مقدار هزینه یال "مانع" باید از بزرگترین زنجیره ممکن از سلولهای "خالی" بزرگتر انتخاب شود تا اینکه یک آستانه ساده بر هزینه مسیر مورد استفاده قرار گیرد تا تعیین شود که مسیر بهینه به هدف باید از میان یک مانع گذر کن یا خیر.

روز می کند، سلول را خاکستری رنگ می کند و سلول مانع را وارد لیست OPEN می کند. پیش اشاره گرها تغییر جهت داده می شوند تا ربات را از مانع عبور دهند. شکل ۲ انتشار اطلاعات را بعد از اینکه ربات کشف کرد که چاه بسته شده، نمایش می دهد مسیر ربات سیاه و حالتهای لیست OPEN خاکستری نشان داده شده. حالتهای RAISE از راه خارج می شوند تا افزایش های هزینه مسیر انتقال یابد. این حالات، حالتهای LOWER در اطراف "لبه" چاه را که کنار موانع بالایی و پایینی را جاروب می کنند و پیش اشاره گرها را به خارج از چاه تغییر جهت می‌دهند، فعال می‌کنند.

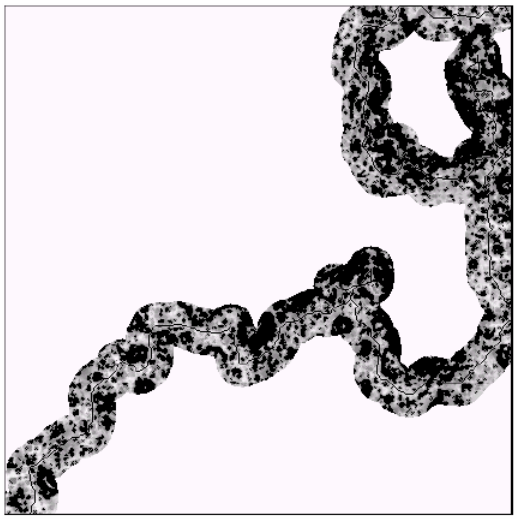


شکل ۳ – وضعیت نهایی پیش اشاره گرها

این روند هنگامی به پایان می رسد که حالات LOWER به سلول ربات می رسند، نقطه‌ای که در آن در کنار مانع پایین تر به سوی هدف حرکت می کند (شکل ۳). توجه کنید که بعد از پیمایش، پیش اشاره گرها فقط به صورت جزئی به روز می شوند. اشاره گرها در میان چاه به خارج اشاره می کنند اما آنهاییکه که در نیمه چپی فضای برنامه ریزی هستند، هنوز به سوی چاه اشاره می کنند. همه حالات یک مسیر به هدف دارند، اما مسیرهای بهینه برای حالات محدودی محاسبه می شوند. این اثر کارایی *D را نمایش می‌دهد. به روز رسانی پیش اشاره گرها که برای تضمین طی مسیر بهینه توسط ربات لازم است، فقط محدود به مجاورت مانع است.



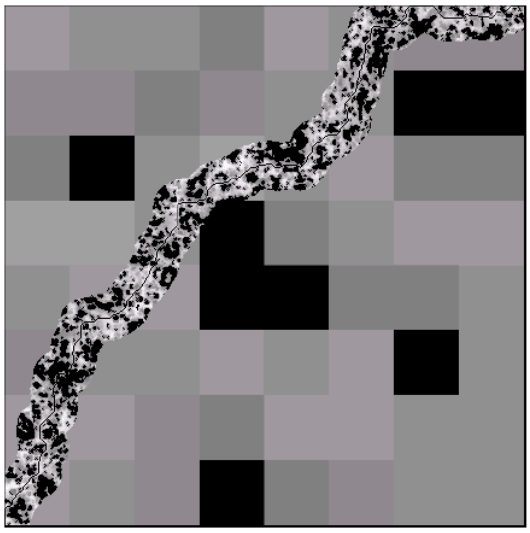
شکل ٤ – برنامه ریزی مسیر با نقشه کامل



شکل٥ – برنامه ریزی مسیر با نقشه خوش بینانه

شکل ۴ برنامه ریزی مسیر را در زمین ایجاد شده فرکتال نمایش می دهد. محیط ۴۵۰×۴۵۰ سلولی است پیمایش محیط های خاکستری ۵ برابر سخت تر از محیط های سفید است، و محیط های سیاه غیرقابل پیمایش هستند. منحنی سیاه مسیر ربات را از گوشه پایین سمت چپ به سمت گوشه بالای سمت راست با داشتن یک نقشه کامل محیط از قبل، نشان می دهد. این مسیر، "بهینه با اطلاعات کامل" خوانده می شود. شکل ۵ برنامه ریزی مسیر را در همان زمین با یک نقشه خوش بینانه (کاملاً سفید) نشان می دهد. ربات به یک حوزه دید دایره ای با شعاع ۲۰۰ سلولی مجهز شده است. نقشه در حین حرکت ربات با اطلاعات حسگر به روز می شود و اختلاف ها وارد لیست open برای پردازش در D* می شود به دلیل فقدان اطلاعات نقشه از قبل، ربات پایین انسداد بزرگ در مرکز حرکت می کند و قبل از پس گرد حول آخرین مانع به سوی هدف، در یک بن بست سرگردان می شود مسیر حاصل تقریباً دو برابرهزینه "بهینه با اطلاعات کامل"، هزینه دارد، هرچند این مسیر با دادن اطلاعاتی که ربات داشت، در هنگام گرفتن آنها، بهینه است.

شکل ۶ همین مساله را با داشتن اطلاعات تقریبی ازنقشه، که با میانگین گرفتن از هزینه یالها در هر محدوده مربعی حاصل شده، نمایش می دهد. این اطلاعات نقشه تا حدی که ربات را به سوی صحیح از انسداد مرکزی هدایت کند، دقیق است، و مسیر حاصل تنها ۶٪ در هزینه از مسیر "بهینه با اطلاعات کامل" بزرگتر است.



شکل ٦ – برنامه ریزی مسیر با نقشه ای با تفکیک پذیری پایین

۳ – صحت ،بهینگی و کامل بودن

بعد از اینکه تمام حالتهای X به t(X)=NEW مقدار دهی اولیه شدند و G به لیست OPEN وارد شده تابع PROCESS-STATE برای ساختن دنباله حالات، مکرراً صدا زده می شود. تابع MODIFY-COST برای ایجاد تغییرات لازم در c(°) و اعمال این تغییرات به لیست OPEN صدا زده می شود. *D ویژگی های زیر را ارائه می دهد.

ویژگی ۱: اگر t(X)=NEW باشد آنگاه دنباله {X} ساخته شده و یکنوا است.

ویژگی ۲: وقتی که مقدار k_{min} برگردانده شده توسط PROCESS-STATE بیشتر یا مساوی h(X) باشد، آنگاه h(X)=o(X).

ویژگی ۳: اگر یک مسیر از X به G موجود باشد و فضای جستجو شامل تعداد محدودی از حالات باشد، {X} بعد از تعدادی محدودی از فراخوانی PROCESS-STATE ساخته می شود. اگر مسیری موجود نباشد PROCESS-STATE، ۱- را با t(X)=NEW بر می گرداند.

ویژگی یک ویژگی برای صحت است: زمانی که یک حالت رویت می شود، یک دنباله متناهی از پیش اشاره گرها به سوی هدف ساخته شده است. ویژگی ۲ یک ویژگی بهینگی است. ویژگی ۳ یک ویژگی کامل بودن است: اگر یک مسیر از X به G موجود باشد، ساخته خواهد شد. اگرهیچ مسیری وجود ندارد، در زمانی محدود گزارش می شود. هر ۳ ویژگی، بدون توجه به الگوی دسترسی برای توابع PROCESS-STATE , MODIFY-COST صحت دارند.

برای اختصار، اثبات ۳ ویژگی فوق غیر دقیق ارائه می شوند. برای اثبات های دقیق و جزئیات به استنتز [۱۴] مراجعه کنید. ابتدا ویژگی ۱ را در نظر بگیرید. هرگاه PROCESS-STATE یک حالت جدید را رویت می کند، b(°) را برابر اشاره گری به یک دنباله حالت موجود قرار می دهد و h(°) را مقدار دهی می کند تا یکنوایی حفظ شود. متعاقباً دنباله های یکنوا با تغییر توابع t(°)، h(°)، k(°) و b(°) عوض می شوند. وقتی حالت X در لیست OPEN قرار داده می شود (یعنی t(X)=OPEN)، k(X) برابر h(X) قرار داده می شود تا یکنوایی برای حالات با پیش اشاره گر، به X حفظ شود. همچنین وقتی حالت X از لیست حذف می شود، مقادیر h(°) همسایه هایش در

صورت نیاز برای حفظ یکنوایی، افزایش می یابد. پیش اشاره گر حالت X، b(X)، فقط در صورتی دوباره برابر Y قرارداده می شود که h(Y)<h(X) و اگر دنباله {Y} هیچ حالتی از نوع RAISE نداشته باشد. از آنجا که {Y} حاوی هیچ حالتی از نوع RAISE نیست، مقدار h(°) هر حالت در دنباله باید کمتر از h(Y) باشد. در نتیجه، X نمی تواند جد Y باشد، و یک حلقه بسته از پیش اشاره گرها نمی‌تواند ایجاد شود. پس هرگاه یک حالت X رویت شود، دنباله {X} ساخته می شود. تغییرات بعدی برای اطمینان از این است که {X} هنوز وجود دارد.

حال ویژگی ۲ را در نظر بگیرید. هر بار که یک حالت در لیست OPEN قرار داده می شود یا از آن حذف می شود، *D مقادیر h(°) را تغییر می دهد تا k(X)<h(Y)+c(Y,X) برای هر زوج حالت (X,Y) که OPEN X است و Y، CLOSED است. سپس، X وقتی برای توسعه انتخاب می‌شود (یعنی k_{min}=k(X))، همسایه های CLOSED از X نمی توانند h(X) را به زیر k_{min} کاهش دهند، و نیز همسایه های OPEN هم نمی توانند، زیرا مقدار h(°) آنها باید بیش از k_{min} باشد. حالت‌هایی که در حین توسعه X در لیست OPEN قرار داده شده اند باید مقادیر k(°) بزرگتر از k(X) داشته باشند، در نتیجه k_{min} با هر صدا زدن PROCESS-STATE افزایش می یابد یا تغییر نمی کند. اگر حالتهای با مقدار h(°) کوچکتر یا مساوی k_{old}، بهینه باشند، آنگاه حالتها با مقدار h(°) میان k_{old} و k_{min} (و یا برابر آنها) بهینه اند، زیرا هیچ حالتی در لیست OPEN نمی تواند هزینه مسیرها را کاهش دهد. بدین گونه، حالتهای با مقادیر h(°) کمتر یا مساوی k_{min} بهینه اند. با استقراء PROCESS-STATE دنباله های بهینه به تمام حالتهای قابل دسترسی می سازد.

اگر هزینه یال c(X,Y) تغییرداده شود، تابع MODIFY-COST ، X را در لیست OPEN قرار می‌دهد ، بعد از اینکه k_{min} کمتر یا مساوی h(X) است. از آنجا که هیچ حالت Y با $h(y) \le h(x)$ نمی‌تواند با تغییر هزینه یال تاثیر پذیرد، ویژگی کماکان صدق می کند.

ویژگی ۳ را در نظر بگیرید. هر بار که یک حالت توسط PROCESS-STATE توسعه می یابد، آن همسایه های NEW اش را در لیست OPEN قرار می دهد. در نتیجه، اگر دنباله {X} موجود باشد، ساخته می‌شود مگر اینکه یک حالت در دنباله، Y، هرگز برای توسعه انتخاب نشود. اما زمانیکه یک حالت در لیست OPEN قرار داده شد، مقدار k(°) آن نمی تواند افزایش یابد. در نتیجه به دلیل یکنوایی k_{min}، حالت Y سرانجام برای توسعه انتخاب می شود.

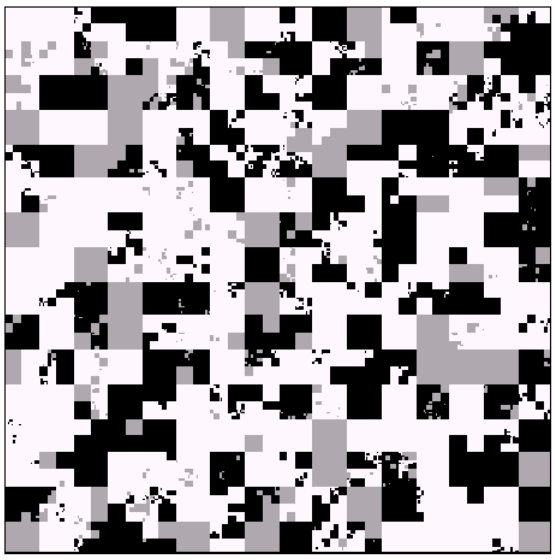
۴ - نتایج تجربی

D* با باز برنامه ریز بهینه[۱] مقایسه شده است تا بهینگی آن تحقیق شود و نیز بهبود سرعت آن معین شود. باز برنامه ریز بهینه در ابتدا یک مسیر یکتا به سوی هدف از محل شروع برنامه ریزی می کنند. ربات اقدام به دنبال کردن مسیر می کند تا زمانیکه حسگر به یک خطا در نقشه پی ببرد. ربات نقشه را به روز می کند، یک مسیر جدید از محل فعلی به هدف برنامه ریزی می کند و این کار را تکرار می کند تا اینکه به هدف برسد. یک تابع مکاشفه ای خوش بینانه $\hat{g}(x)$ برای متمرکز کردن جستجو استفاده می شود، به طوریکه $\hat{g}(x)$ برابرهزینه "خط مستقیم" مسیر از X به محل ربات است با این فرض که تمام سلولها "خالی" هستند. باز برنامه ریز مکرراً حالت های در لیست

[۱] Optimal replanner

OPEN با کمترین مقدار $h(X) + \hat{g}(x)$ توسعه می دهد. از آنجا که $\hat{g}(x)$ یک کران پایینی برای هزینه واقعی از X به ربات برای هر X است، باز برنامه ریز بهینه است [۸].

دو الگوریتم با مساله های برنامه ریزی با اندازه های متعددی مقایسه شده اند. هر محیط یک دایره بوده است که شامل یک نقطه آغازین در دیوار سمت چپ و یک نقطه هدف در دیوار سمت راست بوده. هر محیط شامل مخلوطی از موانع نقشه (یعنی در دسترس ربات قبل از حرکت) و موانع ناشناخته قابل سنجیدن توسط حسگر ربات بوده است. حسگر استفاده شده یک گیرنده همه جهته با حوزه دید شعاعی ۱۰ سلولی بوده است. شکل ۷ یک مدل محیط با ۱۰۰۰۰۰ حالت را نمایش می دهد. موانع نقشه خاکستری نشان داده شده اند و موانع ناشناخته سیاه هستند.



شکل ۷ – محیط نمونه برای مقایسه الگوریتم ها

زمان اجرای هر الگوریتم شدیداً وابسته به پیچیدگی محیط، شامل تعداد، اندازه و نحوه قرارگیری موانع است. نتایج نشان می دهد که با افزایش اندازه محیط، سرعت D* نسبت به بازار

برنامه ریز بهینه سریعاً رشد می کند. شهود این نتیجه این است کـه *D در زمـان پیـدا کـردن مـانع ناشناخته، به طور موضعی باز برنامه ریزی می کند، اما باز برنامـه ریـز بهینـه یـک مسـیر سراسـری جدید ایجاد می‌کند. با افزایش اندازه محیط، مسـیرهای موضـعی، ثابـت مـی ماننـد، امـا پیچیـدگی مسیرهای سراسری افزایش می‌یابد

جدول ۱ – مقایسه *D و بازبرنامه ریز بهینه

Algorithm	1,000	10,000	100,000	1,000,000
Replanner	427 msec	14.45 sec	10.86 min	50.82 min
D*	261 msec	1.69 sec	10.93 sec	16.83 sec
Speed-Up	1.67	10.14	56.30	229.30

۵ – نتایج

۵ – ۱ – خلاصه

این مقاله *D را ارائه می دهد. یک الگوریتم برنامـه ریـزی مسـیر احتمـالاً بهینـه و کـارا بـرای ربات‌های مجهز به حسگر. الگوریتم می تواند با انواع اطلاعات نقشه قبلی سر و کار داشته باشد، از نقشه دقیق و کامل گرفته تا فقدان اطلاعـات نقشـه. *D یـک الگـوریتم بسـیار عمـومی اسـت کـه می‌تواند در مسائل هوش مصنوعی غیر از برنامه ریزی حرکت ربات نیـز بـه کـار رود. در عمـومی ترین حالت، *D می تواند در هر مساله بهینه سازی هزینـه مسـیر کـه پارامترهـای هزینـه در طـول جستجوی راه حل تغییر می کند، مورد استفاده قرار گیرد. *D زمانی که تغییرات کشف شده نزدیک محل شروع فعلی در فضای حالات باشد بیشترین کارایی را دارد، که این حالتی اسـت کـه در یـک

ربات مجهز به حسگر اتفاق می افتد.

برای شرح کامل کاربردهای مرتبط به *D شامل برنامه ریزی با شکل ربات، در نظر گرفتن حوزه دید، خطای محاسبه مطلق[1]، محیط های متغیر، نقشه های سکونت، حوزه های بالقوه، زمین طبیعی، هدفهای متعدد، و رباتهای متعدد به استنتز [۱۴] مراجعه کنید.

۵ - ۲ - کارآینده

برای زمینهای ناشناخته یا تا حدی ناشناخته، تحقیق های جدید در مورد مساله های اکتشاف و ساخت نقشه علاوه بر پیدا کردن مسیر بحث کرده اند. [۶][۹][۱۰][۱۱][۱۵]. با استفاده از استراتژی بالا بردن هزینه ها برای حالتهای رویت شده ، *D می تواند برای پشتیبانی مسائل اکتشاف، توسعه یابد.

درختهای چهارگانه[2] استفاده محدودی در محیط های با مقدار هزینه در یک محدوده پیوسته دارند، مگراینکه محیط شامل حوزه های بزرگ با هزینه پیمایش ثابت باشد. کارآینده شامل نمایش درخت چهارگانه برای چنین محیط هایی علاوه بر محیط هایی با هزینه دودویی (مانند "مانع" و "خالی")، خواهد بود، تا نیاز به حافظه کاهش یابد.

---

[1] dead-reckoning

[2] Quad-trees

تلاش برای بهبود بخشیدن *D برای یک سیستم اجتناب از مانع غیـر جـاده ای بـر روی ربـات متحرک صحرایی در حال انجام است [۱۲] تا به امروز، سیستم مرکب این قابلیت را نشـان داده کـه هدف را پس از چند صدمتر حرکت در محیط ناهموار بدون نقشه اولیه پیدا کند.

سپاس گزاری

ایـن تحقیـق توسـط ARPA تحـت قراردادهـای "درک هـدایت صـحرایی" (شـماره قـرارداد DACA76-89-C0014، تحت نظـارت US Army TEC) و "سیسـتم وسـیله نقلیـه زمینـی بـدون انسان" (شماره قرارداد DAAE07-90-C-R059) حمایت شده است.

نویسنده از آلونزو کلی[۱] و پل کلر[۲] برای نرم افزارهای گرافیکی و ایده های ایجاد زمـین فرکتـال تشکر می کند.

[۱] Alonzo Kelly

[۲] Paul Keller

منابع :

[1] Goto, Y., Stentz, A., “Mobile Robot Navigation: The CMU System,” IEEE Expert, Vol. 2, No. 4, Winter, 1987.

[2] Jarvis, R. A., “Collision-Free Trajectory Planning Using the Distance Transforms,” Mechanical Engineering Trans. of the Institution of Engineers, Australia, Vol. ME10, No. 3, September, 1985.

[3] Korf, R. E., “Real-Time Heuristic Search: First Results,” Proc. Sixth National Conference on Artificial Intelligence, July, 1987.

[4] Latombe, J.-C., “Robot Motion Planning”, Kluwer Academic Publishers, 1991.

[5] Lozano-Perez, T., “Spatial Planning: A Configuration Space Approach”, IEEE Transactions on Computers, Vol. C-32, No. 2, February, 1983.

[6] Lumelsky, V. J., Mukhopadhyay, S., Sun, K., “Dynamic Path Planning in Sensor-Based Terrain Acquisition”, IEEE Transactions on Robotics and Automation, Vol. 6, No. 4, August, 1990.

[7] Lumelsky, V. J., Stepanov, A. A., “Dynamic Path Planning for a Mobile Automaton with Limited Information on the Environment”, IEEE Transactions on Automatic Control, Vol. AC-31, No. 11, November, 1986.

[8] Nilsson, N. J., “Principles of Artificial Intelligence”, Tioga Publishing Company, 1980.

[9] Pirzadeh, A., Snyder, W., “A Unified Solution to Coverage and Search in Explored and Unexplored Terrains Using Indirect Control”, Proc. of the IEEE International Conference on Robotics and Automation, May, 1990.

[10] Rao, N. S. V., “An Algorithmic Framework for Navigation in Unknown Terrains”, IEEE Computer, June, 1989.

[11] Rao, N.S.V., Stoltzfus, N., Iyengar, S. S., “A ‘Retraction’ Method for Learned Navigation in Unknown Terrains for a Circular Robot,” IEEE Transactions on Robotics and Automation, Vol. 7, No. 5, October, 1991.

[12] Rosenblatt, J. K., Langer, D., Hebert, M., “An Integrated System for Autonomous Off-Road Navigation,” Proc. of the IEEE International Conference on Robotics and Automation, May, 1994.

[13] Samet, H., “An Overview of Quadtrees, Octrees and Related Hierarchical Data Structures,” in NATO ASI Series, Vol. F40, Theoretical Foundations of Computer Graphics, Berlin: Springer-Verlag, 1988.

[14] Stentz, A., “Optimal and Efficient Path Planning for Unknown and Dynamic Environments,” Carnegie Mellon Robotics Institute Technical Report CMU-RI-TR-93-20, August, 1993.

[15] Zelinsky, A., “A Mobile Robot Exploration Algorithm”, IEEE Transactions on Robotics and Automation, Vol. 8, No. 6, December, 1992.