گزارش

برنامه ریزی مسیر

(برنامه ریزی با اطلاعات کامل، فضای موقعیت، گراف دیداری)

محمد مهران شهیدی

محمد مهرداد شهیدی

سروش مهدی

استاد ناظر

جواد افشاری

۱۳ خرداد ۱۳۹۹

چکیده

برنامه ریزی مسیر یکی از مسائل مهم در بحث رباتیک و ربات های محرک است. مسئله این است که چگونه یک ربات میتواند از نقطه ایی شروع به یک نقطه دلخواه برسد و مجموعه حرکت را چگونه بدست بیاوریم. در این گزارش چگونگی نشان دادن موقعیت نقاط و موانع بحث می شود و متناسب با آنها الگوریتم هایی بحث خواهد شد. و برخی از این الگوریتم ها نیز در زبان پایتون پیاده و نتیجه با هم مقایسه شده اند. در این گزارش در مورد راه حل هایی برای یکسری مشکلات در برنامه ریزی به خصوص مشکلات محاسباتی که ممکن است رخ دهد بحث شده. و در آخر به برخی از کاربرد های دیگر برنامه ریزی مسیر پرداختیم. برای دسترسی به کد مربوط به پیاده سازی ها از لینک زیر استفاده کنید.

https://github.com/m3hransh/Robotics-path-planning-

فهرست مطالب

۱ مقدمه

برنامه ریزی مسیر یکی از مهمترین پایه های ربات های محرک خودمختار است که به ربات اجازه می دهد که کوتاهترین یا بهینه ترین مسیر بین دو نقطه پیدا کند. مسیر بهینه می تواند مسیری باشد که میزان تعداد چرخش، ترمز کردن یا هر چیز بخصوص که مد نظر است کمینه باشد. الگوریتم ها برای پیدا کردن کوتاهترین مسیر نه تنها در رباتیکز، بلکه در مسیریابی شبکه ها، بازی ویدیویی و ... اهمیت دارند.

در برنامه ریزی مسیر، یک نقشه از محیط و آگاهی ربات از موقعیت خود نسبت به نقشه نیاز است. در این گزارش این فرض ها رو داریم:

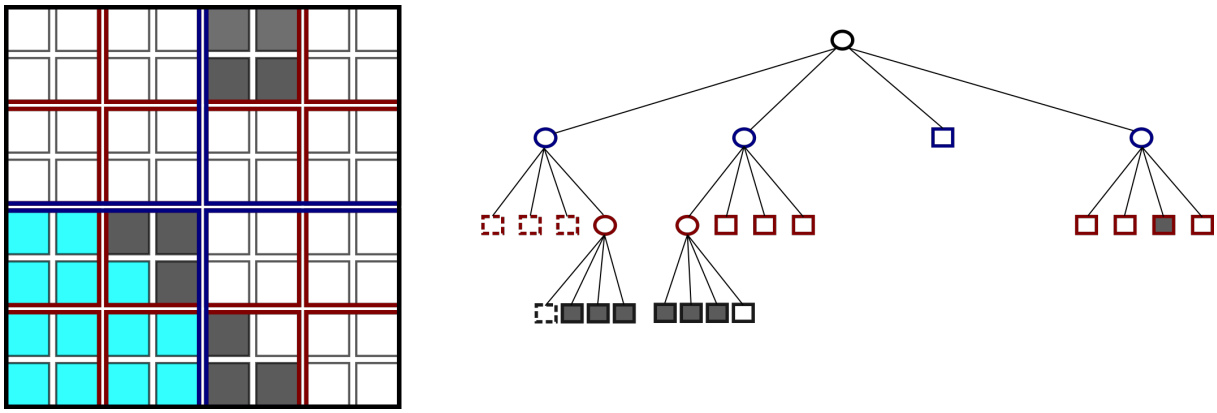
- ربات توانایی موقعیت یابی خود را دارد
- ربات مجهز به نقشه است
- ربات توانایی دوری از مانع های را دارد

اینکه چگونه نقشه را بدست می آوریم یا خود را موقعیت یابی می کنیم و یا چگونه با عدم قطعیت اطلاعات موقعیت عمل می کنیم در این گزارش پوشش داده نمی شوند.

۲ نمایش نقشه

برای برنامه ریزی مسیر، ما به این احتیاج داریم که به یک شکلی محیط را در کامپیوتر نشان دهیم. ما بین دو روش تکمیل کننده تقریب گسسته و پیوسته تفاوت قائل هستیم. در تقریب گسسته ما نقشه را به قطعه های هم اندازه و یا اندازه های متفاوت (مثل اتاق های یک ساختمان) تقسیم می کنیم. نقشه های دومی به نقشه های توپولوژیکال معروف هستند. نقشه های گسسته را میتوان به راحتی به صورت گراف نمایش داد. در این حالت هر قطعه نقشه به عنوان راس های گراف در نظر گرفته می شوند که به وسیله ی یال ها به هم اتصال پیدا می کنند اگر ربات بتواند از راسی به راس دیگر راه پیدا کند. برای مثال نقشه ی جاده ها یک نقشه ی توپولوژیکال است که نقاط اتصال راس ها و جاده ها یال های هستند که با طولشان برچسب گذاری می شوند. به صورت محاسباتی یک گراف می تواند به صورت لیست یا ماتریس همسایگی یا وقوع (adjacency or incidence list/matrix) ذخیره شود. در تقریب پیوسته داخلی (موانع) و خارجی مرز ها که معمولا به شکل چند ضلعی نمایش داده می شود، مشخص می شود که مسیر ها به صورت دنباله از اعداد حقیقی نمایش ک گذاری می شوند. با وجود برتری حافظه روش پیوسته، نقشه های پیوسته بیشتر در رباتیکز استفاده می شوند.

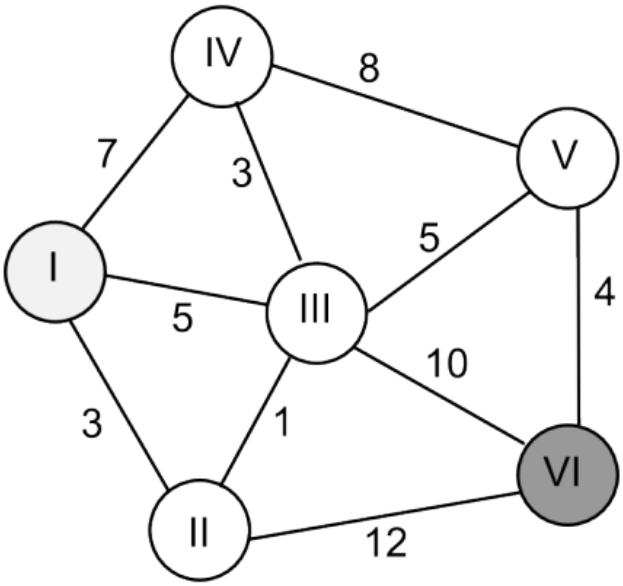
برای نشان دادن موانع، یکی از نقشه های کارآمد نقشه شبکه ایی تصرف (occupancy grid map) است. در نقشه شبکه ایی، محیط به مربع های با وضوح دلخواه تقسیم می شوند که روی آن موانع نشانه گذاری می شوند. در شبکه های تصرفی احتمالی در هر مربع درمورد احتمال حضور یک مانع بحث می شود. این موضوع زمانی که سنسور های ربات قطعیت ندارند اهمیت دارد. نقطه ضعف این روش مصرف شدن زیاد حافظه و همانطور زمان محاسباتی برای پیمایش ساختمان داده ها با راس های زیاد است. راه حل این مسئله استفاده از **k-d Tree** برای نمایش نقشه های شبکه ایی است. یک k-d Tree به صورت بازگشتی محیط را به k قطعه تقسیم میکند برای مثال اگر $k = ۴$ باشد یکه ناحیه به چهار قسمت تقسیم می شود و هرکدوم از این بخش ها نیز خود به چهار بخش تقسیم می شوند تا به وضوح مورد نظر برسیم. در شکل ۱ یک quad-tree نشان داده شده است. همانطور که مشاهده می کنید در این درخت دیگر لازم نیست برای هر مربع یک برگ صرف کنیم و تعداد برگ های ما کمتر از تعداد مربع های شبکه است. یک راه حل ایده آل وجود ندارد و برای هر کاربرد ممکن است راه حل های متفاوت نیاز باشد که می توانند ترکیبی از همه آن ها باشند.



شکل ۱: Quad Tree

۳ الگوریتم های برنامه ریزی مسیر

مسئله پیدا کردن کوتاهترین مسیر از یک راس به راس دیگر در یک گراف متصل در کاربرد های مختلف به خصوص در اینترنت برای جستجوی مسیر بهینه برای پاکت های داده مورد علاقه است. کلمه ی **کوتاهترین** در اینجا به معنی کمترین هزینه جمعی یال ها است که می تواند فاصله فیزیکی، تاخیر و هر اندازه دیگری که برای یک کاربرد خاص مد نظر است باشد. یک مثال از گراف با یال های دلخواه در شکل ۲ نشان داده شده است.

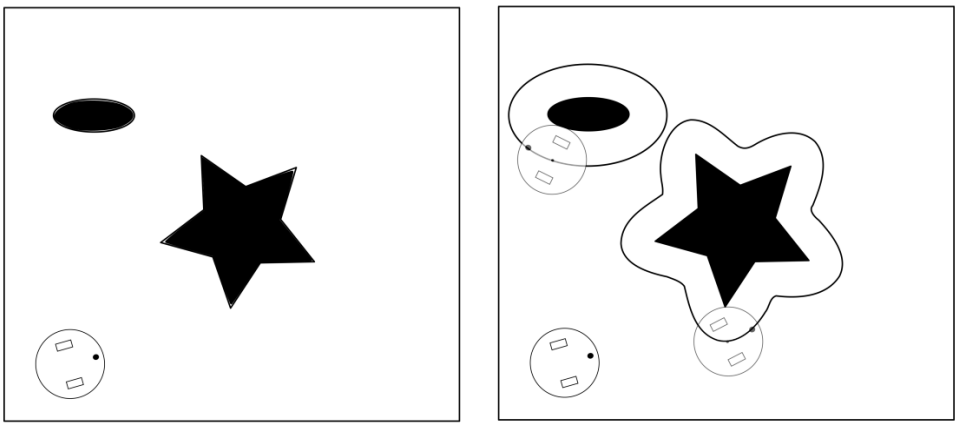


شکل ۲: مسئله برنامه ریزی مسیر از راس I به راس II .

کوتاهترین میسر $I - II - III - V - VI$ به طول ۱۳ است

۱.۳ تجسم ربات

برای کار کردن با تجسم فیزیکی جسم که می تواند پروسه برنامه ریزی مسیر را پیچیده کند، ربات به جرم نقطه ایی کاهش یافته و موانع به اندازه بزرگترین فاصله نقطه ایی روی جسم که از مرکز جسم فاصله دارد به آن مقدار در جهت های مختلف اضافه می شود. این نمایش به **configuration space** معروف است. یک مثال در شکل ۳ نشان داده شده است. configuration space می تواند به عنوان پایه برای نقشه شبکه ایی یا پیوسته به کار رود.



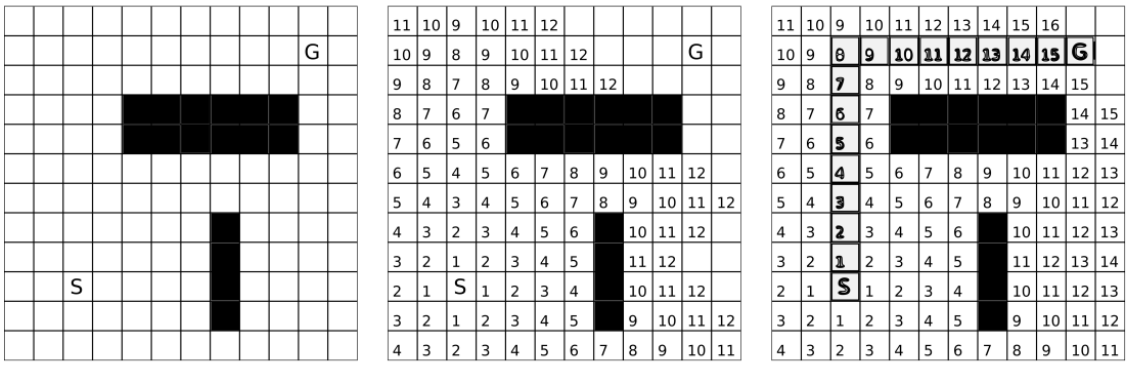
شکل ۳: یک نقشه با موانع با نمایش configuration space آن که با اضافه کردن فضای تصرف ربات به موانع بدست می آید.

۲.۳ الگوریتم دایجسترا

یکی از اولین و ساده ترین الگوریتم ها، الگوریتم دایجسترا است. الگوریتم به این شکل است که از راس ابتدایی شروع می کنیم، همه همسایه های مستقیم با هزینه رفتن به هر کدام مشخص می شود. سپس با راس با کوچکترین هزینه شروع کرده و تمام همسایه های آن با هزینه آن ها از راس شروع مشخص می شود و این عمل تا زمانی که راسی که انتخاب می شود راس پایان باشد ادامه میدهیم و پس از آن ربات میتواند یال هایی که از آن به پایان رسیدم رو دنبال کند. در شکل ۲ الگوریتم ابتدا راس های II ، III و IV با هزینه های به ترتیب ۳، ۵ و ۷ مشخص می شوند. سپس شروع به جستجوی تمام یال های راس II می کند که تا اینجا کمترین هزینه را داشته است. این کار باعث پیدا شدن این می شود که راس III با هزینه کمتری $۵ > ۱ + ۳$ قابل دسترسی است و این راس با هزینه ی ۴ قابل دسترسی است. دایجسترا به منظور اینکه ارزیابی کامل راس II نیاز دارد که یال های باقی مانده را قبل از رفتن به راس بعدی بررسی کند. بنابراین راس VI با $۱۵ = ۱۲ + ۳$ مشخص می شود.

حال راس با کمترین هزینه راس III است. حال می توانیم راس VI را با ۱۴ که کوچکتر از مقدار قبلی آن ۱۵ است مشخص کنیم و راس V با $۹ = ۵ + ۴$ برچسب می گذاریم و همینطور راس IV با $۷ = ۳ + ۴$ باقی می ماند. هرچند که ما دو مسیر به هدف پیدا کردیم که یکی از آنها بهتر است اما تا زمانی که راس هایی با یال های غیر بررسی شده و هزینه های کمتر از ۱۴ وجود دارد، ادامه می دهیم. در واقع، با ادامه جستجو از راس V به کوتاهترین مسیر $I - II - III - V - VI$ به طول ۱۳ است با هیچ راس دیگری برای جستجو باقی می مانیم. به این دلیل که دایجسترا تا زمانی که راس دیگری با هزینه کمتر از هزینه ایی که تا هر نقطه زمان به هدف پیدا شده تمام نمی شود، می توانیم مطمئن باشیم کوتاهترین مسیر در صورت وجود، پیدا می شود و الگوریتم کامل (complete) است. از آنجایی که دایجسترا ابتدا همیشه راس هایی با کمترین هزینه را بررسی می کند، محیط به صورت موجی است که از راس شروع سرچشمه می گیرد و به هدف می رسد. این روش در شرایط خاصی که هدف از راس ابتدایی دور است به شدت غیر بهینه است. با اضافه کردن چند راس به سمت چپ

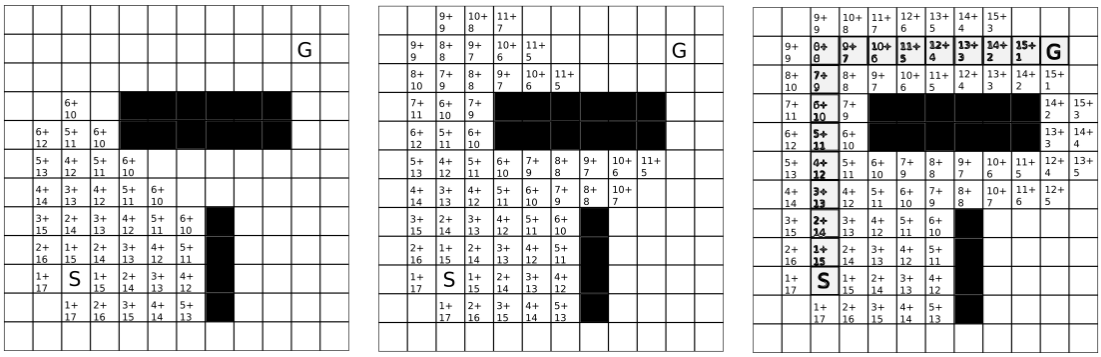
شکل ۲ قابل لمس است. دایجسترا این راس ها را تا زمانی که هزینه آنها از کمترین هزینه که به هدف پیدا شده است، بالاتر برود جستجو می کند. این زمانی که الگوریتم روی شبکه ها اجرا می شوند قابل مشاهده است همانطور که در شکل ۴ نشان داده شده است.



شکل ۴: پیدا کردن کوتاهترین مسیر از **S** به **G** با این شرط که ربات فقط می تواند بالا ـ پایین و چپ ـ راست برود و هزینه یک برای برای هر حرکت.

۳.۳ الگوریتم A*

به جای جستجو در تمام جهت ها، دانش تقریبی از موقعیت هدف در پرهیز از جستجوی راس ها که کاملا از نظر عامل انسانی غیر منطقی است میتواند کمک کند. چنین دانشی که همچنین مشاهده گری دارد از طریق تابع heuristic کدگذاری می توان شد. برای مثال ما می توانیم به راس هایی که فاصله تقریبی کمتری از هدف دارند اولویت بیشتری بدهیم. در این روش نه تنها فاصله واقعی تا راس ها بلکه فاصله تقریبی تا هدف هم در نظر میگیریم. برای مثال این تقریب می تواند فاصله اقلیدسی یا منهتنی بین راس مورد نظر و هدف باشد. این الگوریتم با نام A* معروف است. با توجه به شرایط، این الگوریتم از الگوریتم دایجسترا می تواند خیلی سریع تر عمل کند، و در بدترین شرایط بازده آن یکسان باشد. این در شکل ۵ نشان داده شده. فاصله منهتنی به عنوان تابع heuristic استفاده شده است.



شکل ۵: پیدا کردن کوتاهترین مسیر از **S** به **G** با این شرط که ربات فقط می تواند بالا ـ پایین و چپ ـ راست برود و هزینه یک برای برای هر حرکت.

A* در صورت اینکه فضای جستجو بزرگ باشد، وضوح تصویر برای انجام عمل نیاز باشد و یا ابعاد مسئله جستجو بالا باشد، هزینه محاسباتی بالایی خواهد داشت. جواب این مسائل در الگوریتم برنامه ریزی مسیر بر پایه نمونه برداری (Sampling-based) برآورده

شده است. که در بخش بعدی بررسی خواهیم کرد.

۴.۳ پیاده سازی الگوریتم A* در پایتون

پیاده سازی الگوریتم A* چندان پیچیده نیست. مسئله ایی که اهمیت دارد این است که در این الگوریتم ما یک صف اولویت خواهیم داشت که بر اساس تابع $f(n) = g(n) + h(n)$ (که در آن $g(n)$ کمترین هزینه پیدا شده از شروع تا گره n و $h(n)$ تابع heuristic است) در هر مرحله گره ها را انتخاب و بررسی می کنیم. تا گره ایی که از صف خارج می شود گره ی هدف باشد. در این مثال فاصله منهتنی بین راس مورد نظر تا گره هدف به عنوان تابع heuristic انتخاب شده است. در کد زیر ورودی تابع شامل نقشه که ماتریسی از صفر و یک ها است که موانع را در صورت وجود با یک نشان می دهد و نقطه شروع و هدف است. و خروجی تابع مسیر بهینه، شامل دنباله ایی از موقعیت ها از گره شروع تا پایان خواهد بود.

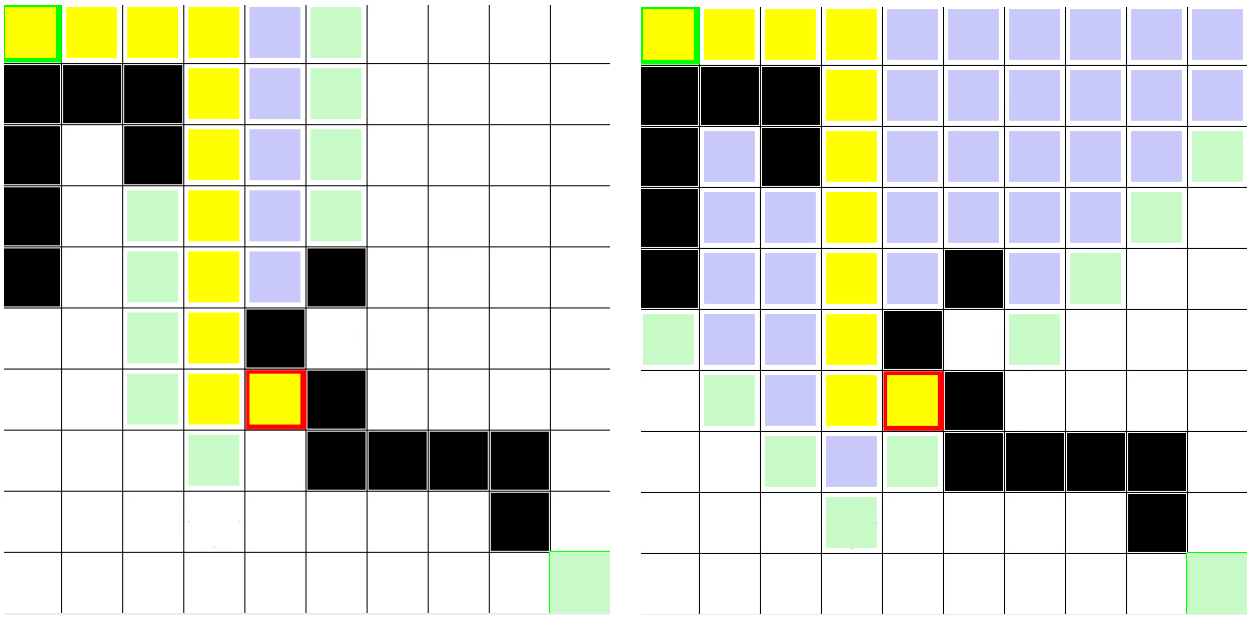
```
def astar(map, start, end):

    # parent, postion, g(n)
    start_node = (None,start,0)
    #f(n)= g(n) + h(n)
    #h: manhatan distance
    #A* priority function
    f = lambda n: n[2]+ abs(n[1][0]-end[0]) + abs(n[1][1]-end[1])

    #postion that we've seen at each step
    closed =set()
    h =[]
    path = []
    hp.heappush(h, (f(start_node),id(start_node), start_node))
    while h:
        node = hp.heappop(h)[2]
        #add position of the node to the closed dict
        closed.add(node[1])
        #check if the end
        if node[1]==end:
            path = optimal_path(m, node)
            break

        #get neighbors
        neighbors = get_neighbors(m,node, closed)
        #add neighbor to the priority queue
        for n in neighbors:
            hp.heappush(h, (f(n),id(n),n))
    for n in path:
        image[n[0]*winW+5:(n[0]+1)*winW-5, n[1]*winH+5:(n[1]+1)*winH-5] =gold

    return path
```

Listing 1: **A***

این تیکه کد را با یک تغییر ساده می توان به الگوریتم دایجسترا تبدیل کرد. فقط کافی است $f(n) = g(n)$ بگذاریم و در قطعه کد بالا با تغییر کد خط ۸ به

```
f = lambda n: n[2]
```

به این نتیجه خواهیم رسید.



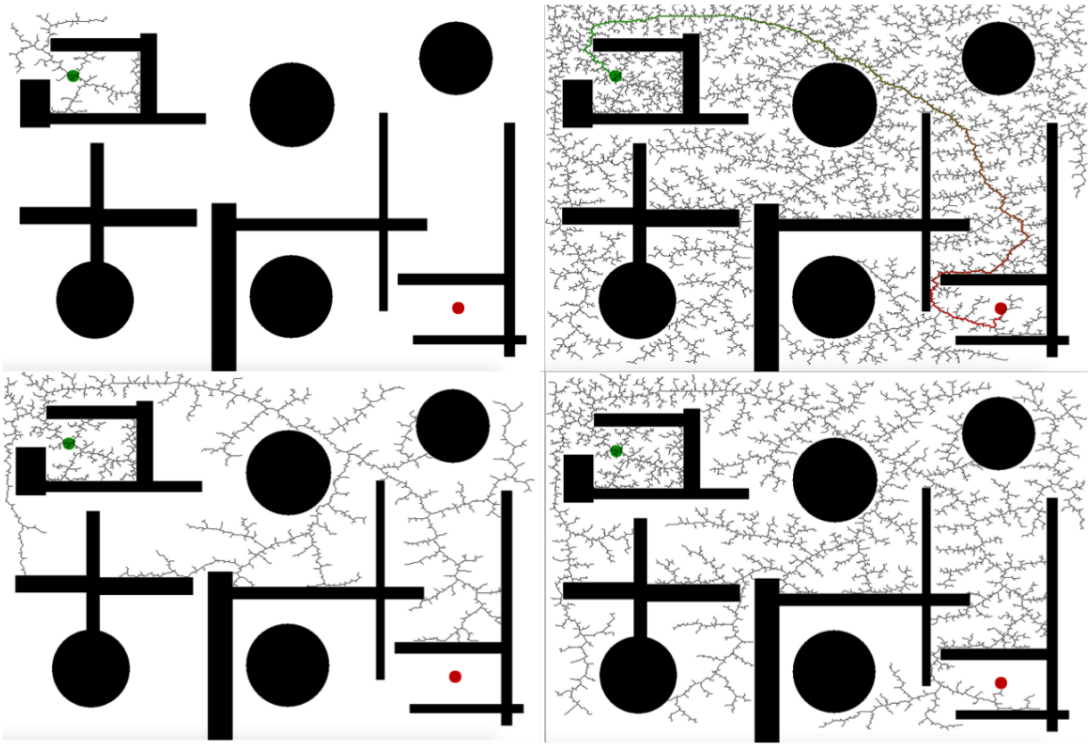
(ب) (آ)

شکل ۶: خانه شروع $(۰, ۰)$ خانه هدف $(۷, ۵)$ مسیر زرد رنگ مسیر بهینه را نشان می دهد، خانه های آبی خانه های مورد بررسی شده ی الگوریتم را نشان می دهد، خانه های سبز گره هایی که داخل صف که بررسی نشده اند را نشان میدهد، (آ)اجرای الگوریتم دایجسترا روی gird map (ب)اجرای الگوریتم A* روی gird map

۴ برنامه ریزی مسیر مبتنی بر نمونه برداری

در بخش های قبلی الگوریتم هایی معرفی شدند که کامل هستند یعنی اگر راه حلی وجود داشته باشد حتما ان را پیدا میکنند اما در عمل این الگوریتم ها وابسته به وضوح نقشه هستند زیرا باید فضای نقشه را که یک فضای پیوسته است برای ان ها به صورت گسسته نمونه گیری کنیم که ممکن است در این عملیات برخی از راه حل ها از بین بروند.

الگوریتم های مبتنی بر نمونه برداری به جای استفاده از روش های ناکامل و یا محاسبه تمام مسیر های ممکن به طور رندوم نقاطی را به یک درخت اضافه می کنند تا زمانی که یک مسیر پیدا شود و یا محدودیت زمانی به پایان برسد.



شکل ۷: جستجوی تصادفی روی فضای جستجوی دو بعدی به این شکل که نقاط تصادفی نمونه برداری و به گراف متصل می شوند تا زمانی که یک مسیر ممکن بین شروع و هدف پیدا شود.

در این الگوریتم ها احتمال پیدا کردن مسیر به سمت یک می رود وقتی که زمان به سمت بی نهایت میل کند. دو تا از الگوریتم های برجسته در این زمینه عبارتند از:

- Rapidly exploring random trees (RRT)
- Probabilistic roadmaps (PRM)

الگوریتم اول از نقطه شروع ربات شروع میکند و یک درخت یکتا را به طور تصادفی رشد میدهد تا یکی از شاخه های آن به هدف برخورد کند اما الگوریتم دوم نقاطی را به عنوان نمونه از فضای مورد نظر انتخاب میکند اگر برخورد نداشته باشند ان ها را به هم متصل میکند و سپس با استفاده از الگوریتم های کلاسیک پیدا کردن کوتاه ترین مسیر در گراف کوتاه ترین مسیر بین نقطه شروع و پایان را پیدا میکند مزیت الگوریتم دوم این است که فقط یکبار درخت را می سازد اما در الگوریتم اول هربار درخت باید از اول ساخته شود.

۱.۴ الگوریتم پایه

فرض کنید X یک وضعیت فضایی چند بعدی باشد. این می تواند بصورت عبارت های انتقالی یا چرخشی یا یک زیرمجموعه یا یک فضای مفصل با یک بعد در هر زاویه ممکن بیان شده باشد. فرض کنید $G \subset X$ یک توپ چند بعدی در فضای موقعیت به عنوان هدف و t زمان قابل مجاز در نظر گرفته شود.برنامه ریزی درخت به شکل زیر است:

الگوریتم ۱ Tree-planner

$Tree = Init(\mathrm{X}, start)$
while $ElapsedTime() < t$ **and** $NoGoalFound(G)$ **do**
  $newpoint = StateToExpandFrom(Tree)$
  $newsegment = CreatePathToTree(newpoint)$
  if $ChooseToAdd(newsegment)$ **then**
    $Tree = Insert(Tree, newsegment)$
  end if
end while
return Tree

این الگوریتم می تواند تا زمانی که زمان مجاز تمام نشده تکرار شود. همچنین می توانیم فاصله تا مقصد را در هر گره درخت ذخیره کنیم که به کمک آن میتوانیم به راحتی کوتاه ترین مسیر را بیابیم. چهار نقطه کلیدی در این الگوریتم وجود دارد:

۱. پیدا کردن نقطه بعدی برای اضافه کردن به درخت

۲. چگونه متصل کردن نقاط جدید به درخت

۳. تست کردن اینکه آیا این مسیر مناسب و بدون برخورد هست یا خیر

۴. پیدا کردن نقطه بعدی برای اضافه کردن

یک روش برجسته این است که یک نقطه تصادفی انتخاب کنیم و آن را به نزدیک ترین نقطه در درخت و یا به هدف متصل کنیم. این روش نیازمند این است که تمامی نقاط در درخت را سرچ کنیم و نزدیک ترین را به نقطه مورد نظر را بیابیم. رویکرد های دیگر نقاطی را در اولویت قرار می دهند که درجه خروجی کمتری دارند. اگر محدودیت هایی روی ربات اعمال شود مثلا ربات به دلیل حمل یک فنجان نتواند مچش را بچرخاند می توانیم این بعد را از فضای جستجو کنار بگذاریم.

۲.۴ وصل کردن نقاط به درخت

به صورت کلاسیک یک نقطه جدید به نزدیکترین نقطه در درخت یا هدف متصل می شود که با محاسبه فاصله این نقاط به راحتی امکان پذیر است اما لزوما کوتاه ترین مسیر را به ما نمی دهد. اما در الگوریتم RRT روشی وجود دارد که هزینه مسیر را به حداقل میرساند. در این روش تمامی نقاط درخت که در یک d-ball با شعاع ثابت از نقطه جدید میباشند در نظر گرفته می شوند و بهترین نقطه انتخاب میشود.

۱.۲.۴ ارزیابی برخورد

این بخش حدود نود درصد تایم اجرا را در بر میگیرد. یک روش خوب برای بهتر کردن زمان محاسبه روش lazy collision evaluation می باشد. در این روش به جای چک کردن تمام نقاط برای برخورد الگوریتم ابتدا یک مسیر پیدا میکند و سپس بخش های مختلف این مسیر را برای برخورد بررسی میکند و اگر بخش هایی مشکل داشت ان ها را حذف کرده و بخش های بدون مشکل را ذخیره می کند و الگوریتم ادامه می یابد.

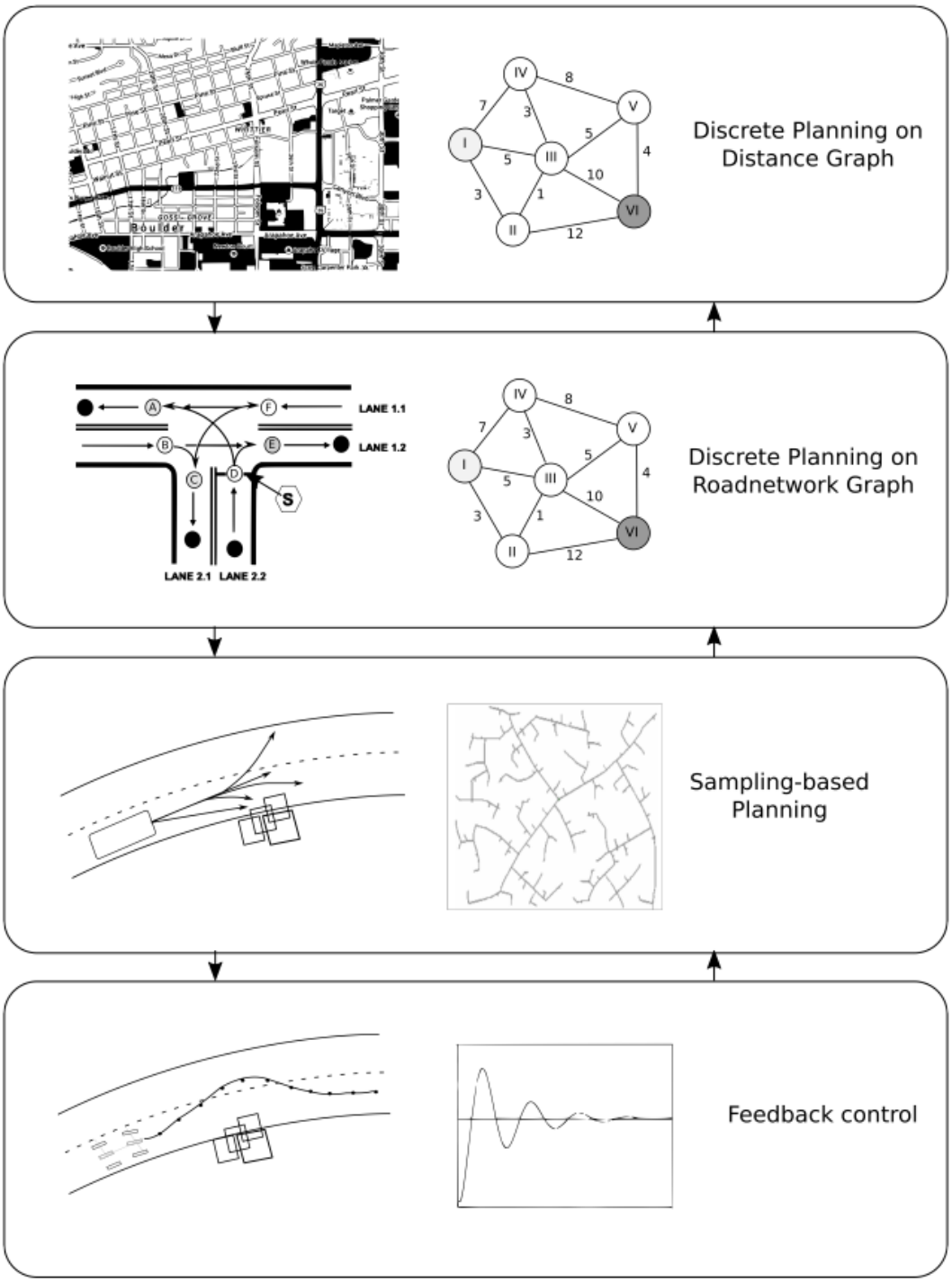
۳.۴ هموارسازی مسیر

از آنجایی که مسیر به صورت رندوم نمونه برداری میشود احتمال زیادی وجود دارد که بهینه نباشد برای همین می توانیم نتیجه را با انجام یک الگوریتم هموارسازی بهبود ببخشیم. یک روش استفاده از خطوط و منحنی ها برای پلتفرم های خاص میباشد. یک روش دیگر استفاده از نمونه واقعی و گرفتن فیدبک از آن می باشد.

۵ برنامه ریزی برای مقایس های طولی مختلف

در عمل ، ممکن است که یه نمایش نقشه و الگوریتم برنامه ریزی کافی نباشد. برای مثال برای برنامه ریزی مسیر یک خودرو ممکن است یه جستوجوی ناهنجار روی شبکه خیابان انجام بشه مانند عملیات در سیستم مسیریاب ماشین ، ولی برنامه ریزی راجب اینکه کدوم راه باید انتخاب بشه دخیل نیست.

برنامه ریزی راه ها و اینکه چطوری دوربرگردان ها و تقاطع ها مسیر یابی بشوند، می بایست لایه از برنامه ریزی گسسته داشته باشد. اینکه چطوری ربات را حرکت بدیم در یک مسیر و از موانع اجتناب کنیم با الگوریتم های sampling-based مناسب تر است. مسیر ها برنامه ریزی شده احتیاج دارند که در نهایت به سرعت چرخ و زاویه چرخش ها با استفاده از کنترل های بازخورد تبدیل شوند. این مراتب در شکل ۸ نمایش داده شده.



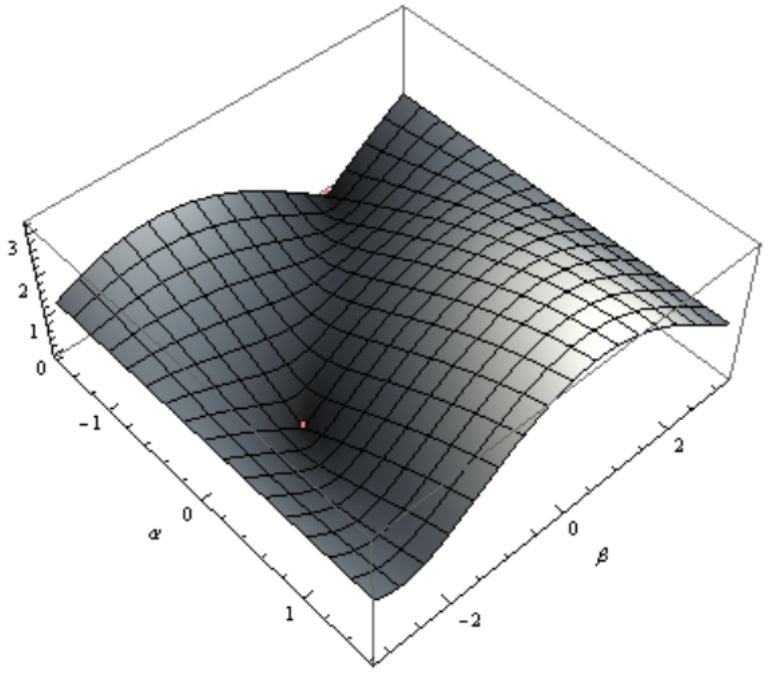
شکل ۸: برنامه ریزی مسیر در مقیاس های طولی مختلف

در اینجا این آرایه های جهت پایین ورودی های است که لایه بالایی برنامه ریزی برای لایه ی پایینی را فراهم میکند، نشون می دهد. و نشانگر های جهت بالا، استثنا هایی که با لایه مرحله پایین تر قابل بیان نیست را نشان میدهند. برای مثال یک کنترل کننده ی بازخورد نمیتواند موانع رو پوشش دهد و به یک برنامه ریزی sampling-based نیاز داریم. یک وضعیت مشابه برای هدایت ربات ها می توان

درست کرد که به چندین نوع از نمایش و کنترل کننده برای برنامه و انجام مسیریابی به صورت بهینه نیاز دارد. توجه کنید که در این نمایش سطح استدلالی که قانون راهنمایی و رانندگی و قضاوت درست را کدگذاری کند استفاده نمی شود. ولی ممکن است یک سری از این، با استفاده از توابع هزینه مانند بیشترین مسافت از موانع یا اطمینان از رانندگی روان و ثابت یا رفتار های پیچیده تر مثل سازگار کردن رانندگی در صورت حضور سرنشین یا ویژگی های سطح زمین که به لایه های بیشتر عمودی نیاز دارد(که به همه لایه برنامه ریزی دسترسی دارد) پیاده سازی شود.

۶ کاربرد های دیگر برنامه ریزی مسیر

زمانی که محیط به گراف گسسته تبدیل شد، ما می توانیم الگوریتم های دیگری که راجب تئوری گراف هست استفاده کنیم تا مسیر حرکتی دلخواه رو برنامه ریزی کنیم ، برای مثال، پوشش کف با انجام depth-first-search یا breadth-first-search روی یک گراف جایی که هر راس اندازه ی ابزار پوشش ربات رو دارد. پوشش (coverage) فقط برای تمیز کردن سطح استفاده نمیشه و در جستوجوی های کامل فضای پیکربندی مثل شکل ۹ استفاده میشه، نمایش داده شده.پیدا کردن مینیمم در این نمودار با استفاده از جستوجوی کامل مسئله ی سینماتیک معکوس انجام شده. و همچنین الگوریتم مشابه میتواند همه ی لینک های روی یک وبسایت را تا عمق دلخواه بدست آورد.



شکل ۹: برنامه ریزی مسیر در مقیاس های طولی مختلف

انجام (DFS) و (BFS) می تواند مسیر های پوششی کارآمد را تولید کند، ولی به اندازه ی کافی بهینه نیست چون یه راس ممکن است دوبار دیده شود. یک مسیر که همه ی راس ها را به هم وصل کرده و از هر راس یک بار عبور کرده را با مسیر همیلتونی می شناسند. یک مسیر همیلتونی که که به راس شروع برمیگردد را با دور همیلتونی میشناسند. این مسئله همچنین با نام فروشنده ی دوره گر نیز شناخته میشود و در این مسئله مسیر باید طوری حساب شود که از هر شهر فقط یکبار عبور شود و یک مسئله **NP-complete** است.

مراجع

[۱] Nikolaus Correll *Introduction to Autonomous Robots*. Magellan Scientific, March 6, 2020.