A0M33BEP
Unmanned vehicles

Course 13: Flight trajectory planning

David Šišlák
sislakd@fel.cvut.cz
Motivation

» UAVs operations in dynamic large-scale environment

» **domain properties:**
  » 4D problem - 3D world considering time
  » excluded zones (positions and time) are given by physical obstacles and no-flight zones
  » excluded zones can be dynamically updated after each planning

» desired trajectory planner properties:
  » optimization-based search in 4D space
  » high performance
  » can be used for nonholonomic vehicle
State space search

» **algorithm properties**
  » time complexity – e.g. number of searched states
  » space complexity – e.g. number of states in memory
  » quality of solution – optimal, complete (find a solution if it exists)
  » effective branching factor – number of states after expansion

» **Uniformed** methods
  » breadth-first search (BFS)
  » depth-first search (DFS)
  » depth-limited search (DLS)
  » iterative deepening search (IDS)
  » bi-directional search

» **Informed** methods
  » use heuristics to select appropriate state to expand
  » good heuristics minimize searched state space to find optimal solution
  » best-first search – A* search, greedy algorithm
  » localized search – hill-climbing algorithm

\[ N = 1 + b^* + (b^*)^2 + \cdots + (b^*)^d \]
Breadth-first search

1. begin
2. open := [Start], closed := []
3. while (open <> []) do begin
4. X := FIRST(open)
5. closed := closed + [X], open := open - [X]
6. if X = GOAL then return(SUCCESS)
7. else begin
8. E := expand(X)
9. E := E - closed
10. open := open + E
11. end
12. end
13. return(failure)
14. end.
1. begin
2. open := [Start], closed := []
3. while (open <> []) do begin
4.     X := BEST(open)
5.     closed := closed + [X], open := open - [X]
6.     if X = GOAL then return(SUCCESS)
7.     else begin
8.         E := expand(X)
9.         E := E - closed
10.        open := open + E
11.     end
12. end
13. return(failure)
14. end.
Search with heuristics

» operations on lines 9 and 10:
  » if a node is already in open
    » do nothing iff f(e) of the existing one is better
    » replace iff f(e) of the existing one is worse
  » if a node is already in closed
    » remove from E iff f(e) of the existing one is better
    » remove from closed iff f(e) of the existing one is worse
evaluation function \( f(\ldots) \) used as a criterion in the algorithm
  
  \[
  c(m,n) - \text{cost of step from } m \text{ to } n
  \]
  
  \[
  g(m) - \text{cost of all steps from the start to the state } m
  \]
  
  \[
  h^*(n) - \text{real cost of all steps from the state } n \text{ to the goal}
  \]

examples of evaluation functions
  
  \[
  f(m,n) - \text{evaluation function used for the transition from the state } m \text{ to the state } n
  \]
  
  \[
  f(m,n) = c(m,n) \text{ ... hill-climbing search – can stuck in a local minima}
  \]
  
  \[
  f(m,n) = h^*(n) \text{ ... greedy algorithm – optimal, complete}
  \]
  
  \[
  f(m,n) = g(n)+h^*(n) \text{ where } g(n) = g(m)+c(m,n) \text{ ... A* algorithm}
  
  \quad \text{– optimal, complete}
  \]
A* algorithm

- $h^*(n)$ is unknown and cannot be used directly
- $h(n)$ is heuristics – estimation of $h^*(n)$ value
- evaluation function is then $f(m,n) = g(n)+h(n)$ usually denoted as $f(n)$ only

- admissible heuristics in A*
  - for every $n$: $0 \leq h(n) \leq h^*(n)$
  - guarantee optimality of A*

- monotonic heuristics
  - for every $n_1$ and $n_2$ where $n_2$ is child of $n_1$
    - $h(n_1)-h(n_2) \leq c(n_1,n_2)$ and $h(\text{goal}) = 0$
  - every monotonic heuristics is admissible

- dominance of heuristics
  - $h_2$ dominates over $h_1$ iff for every $n$ $h_1(n) \leq h_2(n)$
  - A* with $h_2$ will use less state space than with $h_1$
A* algorithm remarks

» has exponential memory complexity
  » usually out of memory earlier than defined time limit

» iterative deepening A* (IDA*) – optimal, complete; reduces memory req.
  » limits the search branches with estimated maximum f for a solution
    1. f_limit=f(n_0)
    2. do A* with nodes of which f <= f_limit,
       in f_1 remember the smallest f > f_limit
    3. if solution found
       then return(solution)
       else f_limit = f_1 and go to the step 2.

» memory bounded A* (MA*) – limits the size of open, removes worst state if no space
  » can stuck in local optimum
State space search for trajectory planning

» planning in a grid
  » 4-neighbors
  » 8-neighbors
  » any-angle

» no widely accepted common benchmarks for trajectory planners -> reduced comparison problem

» reduced problem – any-angle path planning in a grid
  » grid cells are either blocked or unblocked
  » start and goal locations are in grid vertices
  » path is a sequence of linked line elements which ends are in grid positions, any line element cannot intersect any blocked cell
A* algorithm in reduced domain

» search over visibility graph

» NODES is the set of all corner vertices + start and goal vertices
A* algorithm in reduced domain

\begin{algorithm}
\begin{algorithmic}
    \STATE \textbf{Search}(s_{\text{start}}, s_{\text{goal}})
    \STATE \hspace{1em} \textbf{g}(s_{\text{start}}) \leftarrow 0;
    \STATE \hspace{1em} \textbf{h}(s_{\text{start}}) \leftarrow \textit{c}(s_{\text{start}}, s_{\text{goal}});
    \STATE \hspace{1em} \textbf{parent}(s_{\text{start}}) \leftarrow \textit{false};
    \STATE \hspace{1em} \textbf{OPEN} \leftarrow \{s_{\text{start}}\};
    \STATE \hspace{1em} \textbf{CLOSED} \leftarrow \emptyset;
    \WHILE {\textbf{OPEN} \neq \emptyset}
        \STATE \hspace{1em} \textbf{s}_c \leftarrow \text{RemoveTheBest}(\text{OPEN});
        \STATE \hspace{1em} \textbf{if} \ s_c = s_{\text{goal}} \ \textbf{then return} \ s_c;
        \STATE \hspace{1em} \textbf{Insert}(s_c, \text{CLOSED});
        \FOR {\textbf{s}_d \in \text{Candidates}(s_c)}
            \STATE \hspace{2em} \textbf{if} \text{Contains}(s_d, \text{CLOSED}) \ \textbf{then continue};
            \STATE \hspace{2em} \textbf{if} \text{Intersect}(s_c, s_d) \ \textbf{then continue};
            \STATE \hspace{2em} \textbf{g}(s_d) \leftarrow \textbf{g}(s_c) + \textit{c}(s_c, s_d);
            \STATE \hspace{2em} \textbf{h}(s_d) \leftarrow \textit{c}(s_d, s_{\text{goal}});
            \STATE \hspace{2em} \textbf{parent}(s_d) \leftarrow s_c;
            \STATE \hspace{2em} \text{ProcessNode}(s_d);
        \ENDFOR
        \STATE \hspace{1em} \textbf{return} \ \textit{false};
    \ENDWHILE
\ENDALGORITHM
\end{algorithm}
\end{algorithm}
Theta* algorithm

- works like A* algorithm except
  - generates only 4 children
  - reduces path after each expansion

- is not optimal

- but is faster than many other non-optimal versions
  - field D* (FD*) – which is using linear interpolation along grid-edges
RRT – Rapid random trees search

{54} \textbf{RRT}(s_{start}, s_{goal})
{55} \quad \tau.\text{init}(s_{start});
{56} \quad \textbf{for} \; k=1 \; \textbf{to} \; K \; \textbf{do}
{57} \quad \quad s_{rand} \leftarrow \text{RandomVertex}();
{58} \quad \quad s_{near} \leftarrow \tau.\text{nearest}(s_{rand});
{59} \quad \quad s_{new} \leftarrow \tau.\text{stopping\_configuration}(s_{near}, s_{rand});
{60} \quad \quad \textbf{if} \; s_{new} \neq s_{near} \; \textbf{then}
{61} \quad \quad \quad \tau.\text{add\_vertex}(s_{new});
{62} \quad \quad \quad \tau.\text{add\_edge}(s_{near}, s_{new});
{63} \quad \quad \textbf{end}
{64} \quad \quad s_{near} \leftarrow \tau.\text{nearest}(s_{goal});
{65} \quad \quad \textbf{if} \; \text{not} \; \text{Intersect}(s_{near}, s_{goal}) \; \textbf{then}
{66} \quad \quad \quad \textbf{return} \; \tau, s_{near};
{67} \quad \quad \textbf{end}
{68} \quad \textbf{return} \; false;
{69} \textbf{end}
KD-tree structure

» complexity
  » insert new point $O(\log n)$
  » remove point $O(\log n)$
  » query nearest $O(\log n)$
RRT – Rapid random trees search
RRT – Rapid random trees search
Accelerated A* (AA*)

- extension of A* algorithm
- no-preprocessing of an environment
- key concepts:
  - adaptive state generation using 4-successors at maximum -> reduction of branching factor to constant
  - search for any-angle path using progressive path truncation applied to every partial path represented by a state
AA* - adaptive state generation
AA* - progressive path truncation
\{39\} **Candidates**\((s_c)\)
\{40\} \hspace{1em} sq \leftarrow \text{DetectMaxSquare} (s_c);
\{41\} \hspace{1em} \text{return} \hspace{1em} \text{UsableSideCenters} (sq);
\{42\} \hspace{1em} \text{end}

\{43\} **ProcessNode**\((s_d)\)
\{44\} \hspace{1em} \text{foreach} \hspace{1em} s_n \in \text{EllipseMbs} (CLOSED, s_{start}, s_d)
\hspace{1em} \text{do}
\{45\} \hspace{1em} \hspace{1em} \text{if} \hspace{1em} g(s_n) + c(s_n, s_d) < g(s_d) \hspace{1em} \text{then}
\{46\} \hspace{1em} \hspace{2em} \text{if} \hspace{1em} \text{not} \hspace{1em} \text{Intersect} (s_n, s_d) \hspace{1em} \text{then}
\{47\} \hspace{1em} \hspace{3em} g(s_d) \leftarrow g(s_n) + c(s_n, s_d);
\{48\} \hspace{1em} \hspace{3em} \text{parent}(s_d) \leftarrow s_n;
\{49\} \hspace{1em} \hspace{1em} \text{end}
\{50\} \hspace{1em} \text{end}
\{51\} \hspace{1em} \text{end}
\{52\} \hspace{1em} \text{InsertOrReplaceIfBetter} (s_d, OPEN);
\{53\} \hspace{1em} \text{end}
Evaluation – randomized grids

» grids size 100x100 with randomly blocked cells, start and goal positions
» four different densities of obstacles: 5%, 10%, 20% and 30%
» results averaged from 500 tasks for the same configurations provided to all algorithms
» each generated task was validated by A* and then by others

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Shortest Paths A*</th>
<th>Θ*</th>
<th>AA*</th>
<th>RRT PS</th>
<th>dynamic bi-RRT PS</th>
</tr>
</thead>
<tbody>
<tr>
<td>5% blocked cells</td>
<td>54.210 (10.084)</td>
<td>54.345 (0.023)</td>
<td>54.210 (0.079)</td>
<td>73.834 (0.003)</td>
<td>73.411 (0.0004)</td>
</tr>
<tr>
<td>10% blocked cells</td>
<td>53.190 (11.896)</td>
<td>53.428 (0.026)</td>
<td>53.190 (0.082)</td>
<td>79.558 (0.013)</td>
<td>75.896 (0.0015)</td>
</tr>
<tr>
<td>20% blocked cells</td>
<td>53.301 (18.476)</td>
<td>53.623 (0.036)</td>
<td>53.301 (0.101)</td>
<td>85.207 (0.032)</td>
<td>78.030 (0.0037)</td>
</tr>
<tr>
<td>30% blocked cells</td>
<td>53.206 (31.493)</td>
<td>53.611 (0.049)</td>
<td>53.206 (0.129)</td>
<td>85.344 (0.077)</td>
<td>81.566 (0.0089)</td>
</tr>
</tbody>
</table>

Table: Path lengths and run-times (in parenthesis), each averaged from 500 runs for random grids of size 100 x 100.
Planning for nonholonomic vehicle

» path specifies motion trajectory for airplane reference point
» airplane dynamics can be converted to flight envelope elements:
  » straight
  » horizontal turn
  » vertical turn
  » spiral
» elements parameters are constrained by
  » minimum horizontal turn radius
  » minimum vertical turn radius
  » maximum airplane climb/descend rate
Planning for nonholonomic vehicle
Planning for nonholonomic vehicle

segment$_1$

segment$_2$

segment elements:
- blue: straight element
- green: horizontal turn
- red: vertical turn
- yellow: spiral element
Dubin car problem

\[ R_\alpha L_\beta R_\gamma \]

\[ R_\alpha S_d L_\gamma \]
Dubin car problem
AA* for nonholonomic planning

» adaptive sampling – removes the trade-off between speed and search precision
» minimal sampling step – search precision
» progressive path smoothing
» similarity check
» heuristics based on shortest connection
AA* for nonholonomic planning

speed: 8x
current time: 5990
fps: 19

W - toggle waypoint
G - toggle GC visibility
F - toggle zones radius
A - toggle actions
P - toggle NC predictions
D - toggle no-flight zones
K - toggle night plane
C - toggle communication
L - toggle entity info
J - toggle ground
N - toggle non-cooperative NFZ
M - toggle 2D/3D
Large-scale planning
Comparison to planning with current plan

16/05/16

A0M33BEP – Course 12

W - toggle waypoints  B - toggle accessibility  R - toggle zones radius  A - toggle actions  F - toggle NC predictions  M - toggle 2D/3D  P - toggle flight plans  C - toggle communication  I - toggle entity info  G - toggle ground  N - toggle non cooperative NFZ
Comparison to planning with current plan

speed: normal
current time: 458
Comparison to planning with current plan

16/05/16

A0M33BEP – Course 12