Humanoid Robotics

Footstep Planning

Maren Bennewitz
Goal of this Chapter

- Given the robot’s pose in a map of the environment
- Compute a path to a target location
- First, footstep path in a 2D grid map representation of the environment
- Then, footstep path in a height map
Reminder: A* Search (1)

- Best-first search to find a cost-optimal path to the goal state on a graph

- Expands the states according to the evaluation function: \( f(n) = g(n) + h(n) \)

  - \( g(n) \): actual costs from start to state \( n \)
  - Heuristics \( h(n) \): estimated costs from \( n \) to the goal

- Expand a node = generate its successors, compute their \( f \)-values, and insert them into the priority queue (or update their \( f \)-value)
Reminder: A* Search (2)

- A* expands nodes, starting from the current robot state
- Successor nodes are sorted in a priority queue according to their $f$-value
- **In each iteration, A* expands the node with the minimum $f$-value until the minimum is the desired goal state**
- Each node keeps a pointer to its parent node to determine how it was found
- From that, the solution path can be reconstructed
A* Search (3)

- Optimal path = path with the minimum accumulated $g$-cost from the start to the goal state

- A* yields the **optimal path** if $h$ is **admissible**
  (proof: AI lect., Russell/Norvig)

- Let $h^*(n)$ be the actual cost of the optimal path from $n$ to the goal

- Heuristics $h$ is **admissible** if the following holds for all $n$: $h(n) \leq h^*(n)$
A* Algorithm

\[
\text{OPEN} = \text{priority queue containing START} \\
\text{CLOSED} = \text{empty set} \\
\text{while lowest rank in OPEN is not the GOAL:} \\
\text{current} = \text{remove lowest rank item from OPEN} \\
\text{add current to CLOSED} \\
\text{for neighbors of current:} \\
\text{cost} = g(\text{current}) + \text{movementcost}(\text{current}, \text{neighbor}) \\
\text{if neighbor in OPEN and cost less than g(neighbor):} \\
\quad \text{remove neighbor from OPEN, because new path is better} \\
\text{if neighbor in CLOSED and cost less than g(neighbor):} \\
\quad \text{remove neighbor from CLOSED} \\
\text{if neighbor not in OPEN and neighbor not in CLOSED:} \\
\quad \text{set } g(\text{neighbor}) \text{ to cost} \\
\quad \text{add neighbor to OPEN} \\
\quad \text{set priority queue rank to } g(\text{neighbor}) + h(\text{neighbor}) \\
\quad \text{set neighbor's parent to current} \\
\text{reconstruct reverse path from goal to start} \\
\text{by following parent pointers}
\]

does not happen with consistent admissible h

calculate g-value of successor

touched states

calculate f-value of successor

already expanded states

source: http://theory.stanford.edu/~amitp/GameProgramming/ImplementationNotes.html
Reminder: A* on a 2D Grid Map

- Given: 2D grid map (=states)
- 8-connected neighborhood (=actions)
- A* finds the cost-optimal path to the goal
- Consider move cost and occupancy value
- Starts with the cell containing the current pose of the robot:
Reminder: A* on a 2D Grid Map

- Generate successors (8-connected neighborhood) of node with lowest $f$-cost
- Add only states that are obstacle-free
- Continue until the lowest-cost node is close to the goal
- Actual costs $g = \text{costs of the path from the start to the current node}$
- Move cost: 1 per horizontal/vertical grid cell traversal, $\sqrt{2}$ per diagonal
- Estimated move costs to the goal $h$: straight-line distance (Euclidean)
Example: Found 2D Path and Expanded States
2D Path Planning for Humanoids

- Compute a collision-free 2D path first, then follow this path with a walking controller
- For 2D path planning use a safety margin around obstacles
- But how large should the margin be?

![Diagram showing large clearance (detour) vs. too small clearance (collision) paths between a start and goal point.](image)
2D Path Planning for Humanoids

- Compute a collision-free 2D path first, then follow this path with a walking controller
- For 2D path planning use a safety margin around obstacles
- But how large should the margin be?
- Footstep planning needed in narrow regions

![Diagram showing large clearance (detour) and too small clearance (collision)]
Path Planning for Humanoids

- Humanoids can avoid obstacles by stepping over or close to them
- However, planning whole-body motions has a high computational complexity
- Planning in the space of foot poses reduces the computational complexity
Footstep Planning with A*

- Search space: foot poses \((x, y, \theta)\)
- Global position and orientation of the stance foot
- **Given: discrete set of footsteps**
- Find the optimal footstep path with A*
Footstep Planning with A*

- Construct a search tree of successor states
- Fixed set of possible footsteps
- Check foot placements for collisions with obstacles during expansion

source: Kuffner et al.
Footstep Planning

- State $s = (x, y, \theta)$
- Footstep action $a = (\Delta x, \Delta y, \Delta \theta)$
- Fixed set of footsteps $F = \{a_1, \ldots, a_n\}$
- Successor state $s' = t(s, a)$
- Transition costs:
  
  $$c(s, s') = \| (\Delta x, \Delta y)^T \|$$
Footstep Planning

goal

start
Footstep Planning

start

goal
Footstep Planning
Footstep Planning

estimated costs from $s'$ to goal

transition costs $c(s, s')$

path costs from start to $s$

$g(s)$

$S'$

$h(s')$
Footstep Planning

Obstacles create local minima in the search space.

What is a good heuristic function?
Recap: Footstep Planning

- State $s = (x, y, \theta)$
- Footstep action $a = (\Delta x, \Delta y, \Delta \theta)$
- Fixed set of footsteps $F = \{a_1, \ldots, a_n\}$
- Successor state $s' = t(s, a)$
- Transition costs:
  \[
  c(s, s') = \| (\Delta x, \Delta y)^T \|
  \]
  used to calculate g-value
Heuristics for Footstep Planning

- **Highly influence the A* performance**
- Estimate the costs to the goal taking into account the largest possible forward step
- Typical heuristic functions are based on:
  - Euclidean distance (straight line)
  - Shortest 2D path with safety margin around obstacles

In general, shortest 2D path heuristic is inadmissible for humanoids!
Heuristics for Footstep Planning

- Shortest 2D path heuristics yields good results in practice

- Other possibility
  - “erase” oversteppable objects from grid map in a preprocessing step
  - Compute heuristics using remaining objects
  - Consider oversteppable objects in collision check during node expansion
Local Minima in the Search Space

- A* searches for the optimal path
- But sub-optimal results are often sufficient for navigation
Anytime Repairing A* (ARA*)

- Weighted A* (wA*): Heuristics “inflated” by a factor $w$
- ARA* runs a series of wA* searches, iteratively lowering $w$ as long as a given time limit is not met
- The resulting paths are guaranteed to cost no more than $w$ times the costs of the optimal path
- ARA* reuses information from previous searches

[Likhachev et al. ’04]
Anytime Repairing A* (ARA*)

- An initially large \(w\) causes the search to find an initial non-optimal solution quickly.
- Given enough time, ARA* finally searches with \(w = 1\), producing an optimal path.
- If the time limit is met before, the cost of the best solution found is guaranteed to be no worse than \(w\) times the optimal solution.
- ARA* finds a first sub-optimal solution faster than standard A* and approaches the optimal one as time allows.
ARA* with Euclidean Heuristic

\[ w = 1 \]
ARA* with Euclidean Heuristic

\[ w = 10 \]
ARA* with Euclidean Heuristic

\[ w = 5 \]
ARA* with Euclidean Heuristic

\[ w = 2 \]
ARA* with Euclidean Heuristic

\[ w = 1 \]
ARA* with Shortest 2D Path Heuristics

\[ w = 1 \]
Planning in Dense Clutter
Until a First Solution is Found

A* Euclidean heuristics

ARA* (w=5) Euclidean heuristics

ARA* (w=5) Shortest 2D path

very high computation costs

medium costs

very low costs

2D path heuristics: inadmissible and may lead to longer paths, but expands much fewer states
Footstep Planning on a Heightmap

- So far: planar obstacles
- Now: Consider the height of objects
- Planning of footsteps also includes stepping onto or over objects
- Use a modified heuristics on a height map
Action Set for the Nao Humanoid

- Standard planar steps
- Extended 3D stepping capabilities
Footstep Planning in Height Maps

- Consider the 8-neighborhood in the 2D height map
- For computing the costs of an action take also into account the height difference
Safe Stepping Actions

- Allow only states where all cells covered by the footprint have a small height difference

- Height difference of the step must be within the robot’s stepping capabilities
Efficient Whole-Body Collision Checking

- Pre-compute the volume covered by the robot when executing the footstep actions
- Use the minimum height values for fast collision checking within the height map
Efficient Whole-Body Collision Checking

- Store minimal height values of the robot for the footstep actions
- Use these inverse height maps (IHM) for fast collision checking
- An action $a$ can be executed collision-free in state $s$ if

$$\forall (u, v) \in \text{IHM}^a : \text{IHM}^a_{(u,v)} + z_s > h_{(u,v)}$$

- $z_s$: height of stance foot at $s$
- $h_{(u,v)}$: heightmap value
- $\text{IHM}^a_{(u,v)}$: volume covered by the robot
Heightmap learned online from the robot's depth camera data
Footstep Planning with A*

- Uses a set of footstep actions to reduce the computational demand
- Standard approach: fixed set of actions
- How to choose the set of actions?
Footstep Planning with A*

Small set → fast planning
limited search space

Large set → large coverage
long planning time
Adaptive Node Expansion

New approach:

- Add only a small set of nodes at each expansion step
- Systematically search for viable successors
- Apply fast validity checks using height information

- Leads to a high success rate, short paths, and fast planning times
Reachability Map (RM) for Generating Footsteps

- Discretization of feasible footsteps
- RM can be precomputed using inverse kinematics
- Depending on the displacement direction, the RM provides
  - Maximum displacement from stance foot
  - Maximum displacement along the upward and downward directions
Inverse Kinematics (IK)

- IK computes the joint angles so that the end-effector reaches a desired pose
- Example: Consider a simple 2D kinematic chain with two 1-DOF rotational joints

\[ e = (e_x, e_y) \]
Reachability Map
Adaptive Action Set for Node Expansion

- Local search around maximum forward step
- Check each possible successor:
  - Footstep feasible according to the reachability map?
  - Footstep on a planar region?
  - Later (node expansion): No collision of the robot’s swept volume with obstacles?
- Result: set of viable successor states that adapt to the local environment
Node Expansion
Node Expansion
Node Expansion
Node Expansion

- Only a small set of viable nodes is added to the priority queue at each expansion step
- The set covers the valid areas in the reachable range
Experiments

- Planning area of 2.4m x 2.4m, randomly generated obstacles
- Resolution of 1.5cm for the height map
- Local goal located at the opposite side on the map
- Comparison to A* with fixed sets of 10 and 20 footsteps
Example Map

A* (small)          A* (large)          Adaptive

start          start          start

goal          goal          goal
Example Map

Planning times

A* (small)
A* (large)
Adaptive

Total path costs

A* (small)
A* (large)
Adaptive
Further Examples

A* (small)        A* (large)        adaptive

map 1

start

goal

map 2

start

goal

start

goal

start

goal
Experimental Results

<table>
<thead>
<tr>
<th>Map 1</th>
<th>A* (small)</th>
<th>A* (large)</th>
<th>adaptive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Planning Time</td>
<td>85.7 ms</td>
<td>60.3 ms</td>
<td>14.3 ms</td>
</tr>
<tr>
<td>Path Cost</td>
<td>7.6</td>
<td>4.0</td>
<td>3.7</td>
</tr>
<tr>
<td>Expanded Nodes</td>
<td>28260</td>
<td>13835</td>
<td>1516</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Map 2</th>
<th>A* (small)</th>
<th>A* (large)</th>
<th>adaptive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Planning Time</td>
<td>67.2 ms</td>
<td>75.0 ms</td>
<td>48.7 ms</td>
</tr>
<tr>
<td>Path Cost</td>
<td>6.9</td>
<td>5.0</td>
<td>4.3</td>
</tr>
<tr>
<td>Expanded Nodes</td>
<td>22374</td>
<td>16805</td>
<td>5852</td>
</tr>
</tbody>
</table>

Computations performed on a single Intel Core i7 3770 CPU
Real-Time Footstep Planning
Speeding Up Footstep Planning

- Planning the whole path with footsteps may not always be necessary
- Idea: combine global, fast 2D path planning with local footstep planning
- Confine footstep planning to a bounded local map around the robot
Speeding Up Footstep Planning

- Combine global, fast 2D path planning with local footstep planning
Summary

- Search-based footstep planning with A*
- Anytime search-based footstep planning with sub-optimality bounds: ARA*
- Heuristics influences planning performance
- Footstep planning for 3D obstacle traversal
- Adaptive node expansion to speed up the search
- Use a global 2D path in combination with a local map to bound the complexity of footstep planning
Literature

- Anytime Search-Based Footstep Planning with Suboptimality Bounds,

- Real-Time Footstep Planning in 3D Environments,
  P. Karkowski, S. Osswald, and M. Bennewitz, HUMANOIDS, 2016

- Fast Footstep Planning with Aborting A*,
  M. Missura and M. Bennewitz, ICRA, 2021
Acknowledgment

- Previous versions of parts of the slides have been created by Armin Hornung, Philipp Karkowski, and Stefan Osswald