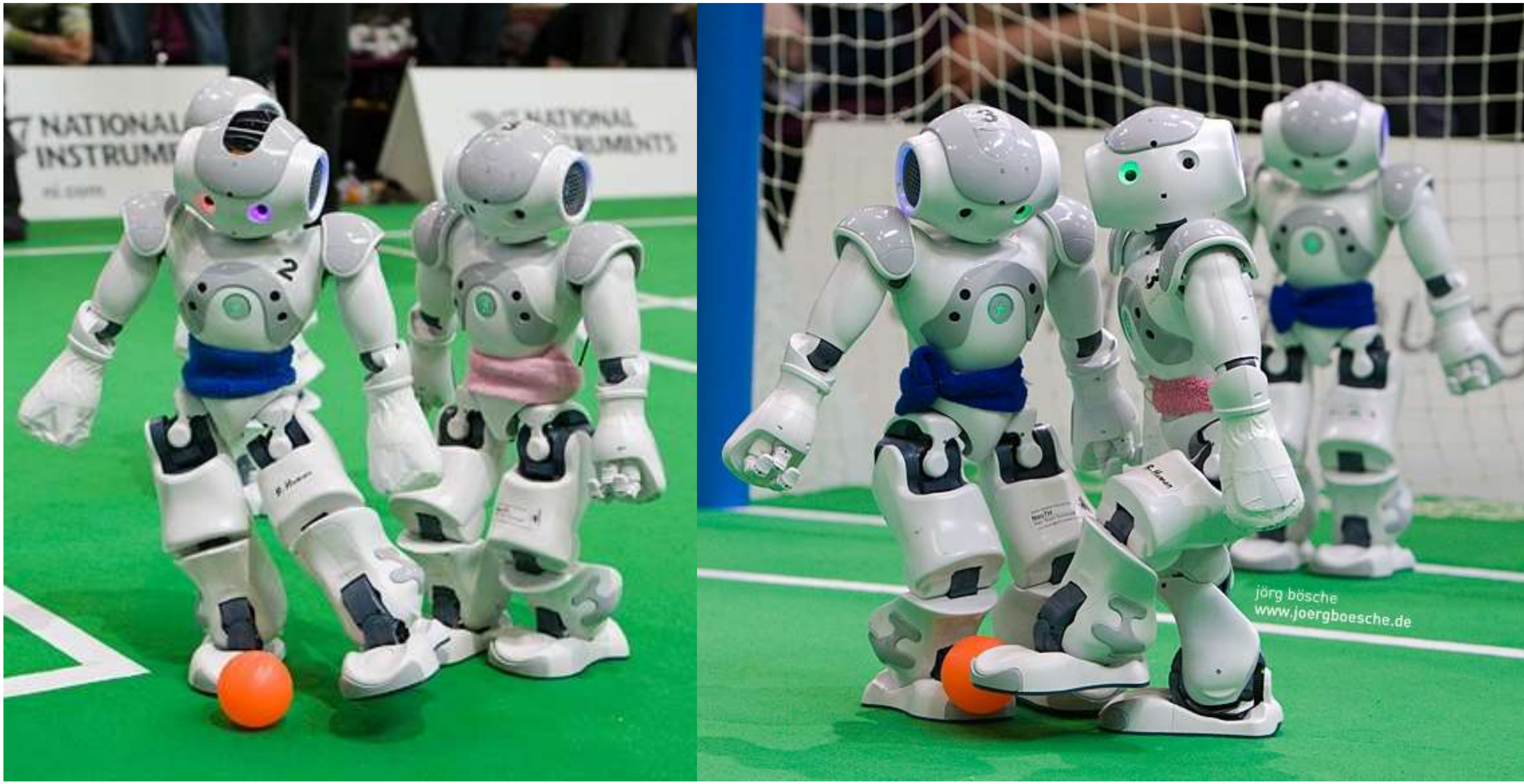
Search-Based Footstep Planning

Armin Hornung

University of Freiburg, Germany

Joint work with J. Garimort, A. Dornbush, D. Maier, C. Lutz, M. Likhachev, M. Bennewitz

Motivation



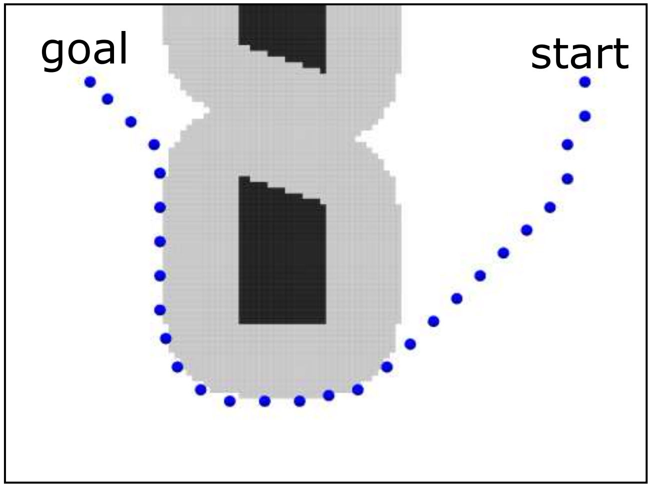
BHuman vs. Nimbro, RoboCup German Open 2010

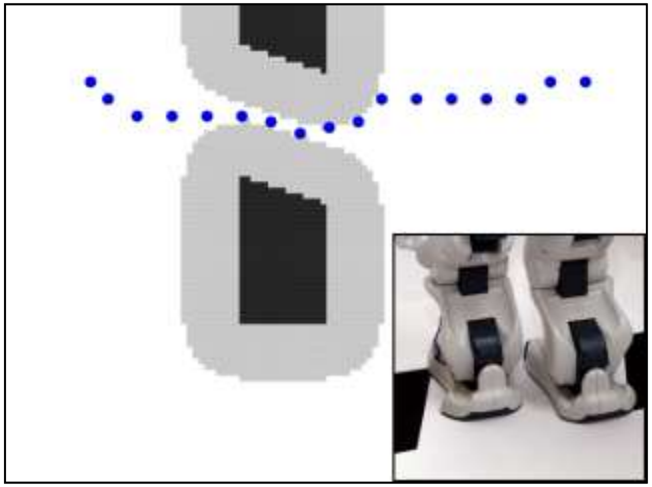
Photo by J. Bösche, www.joergboesche.de

Previous approaches: 2D Path Planning

- Compute collision-free 2D path first, then footsteps in a local area
  [Li et al. '03, Chestnutt & Kuffner '04]

- Problem: 2D planner cannot consider all capabilities of the robot



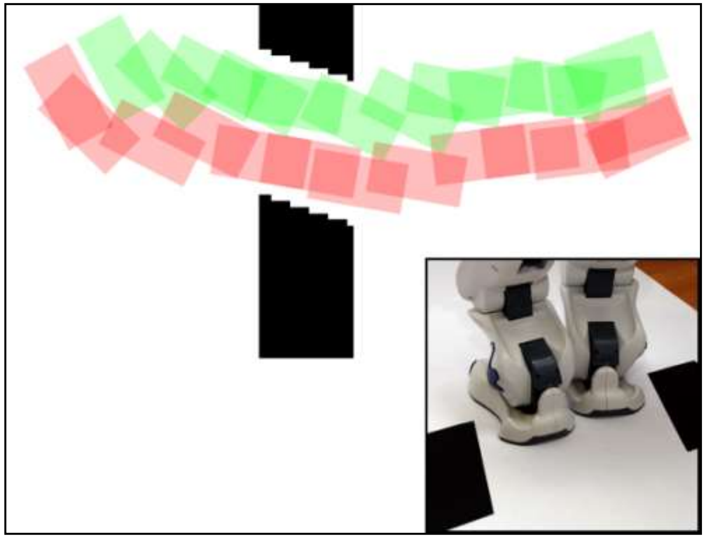


Path Planning for Humanoids

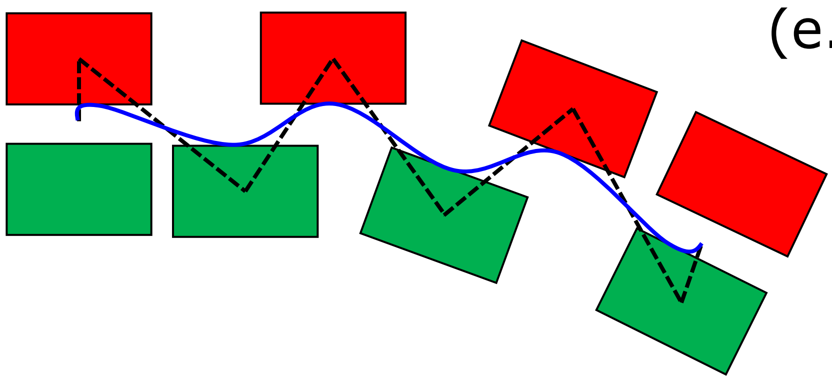
- Humanoids can avoid obstacles by stepping over or close to them
- However, planning whole-body motions has a high computational complexity

[Hauser et al. ‘07, Kanoun ’10, …]

- Planning for possible foot locations reduces the problem



Overview: Path Planning for Humanoids



Footsteps

↓

Desired ZMP Trajectory

↓

CoM Trajectory

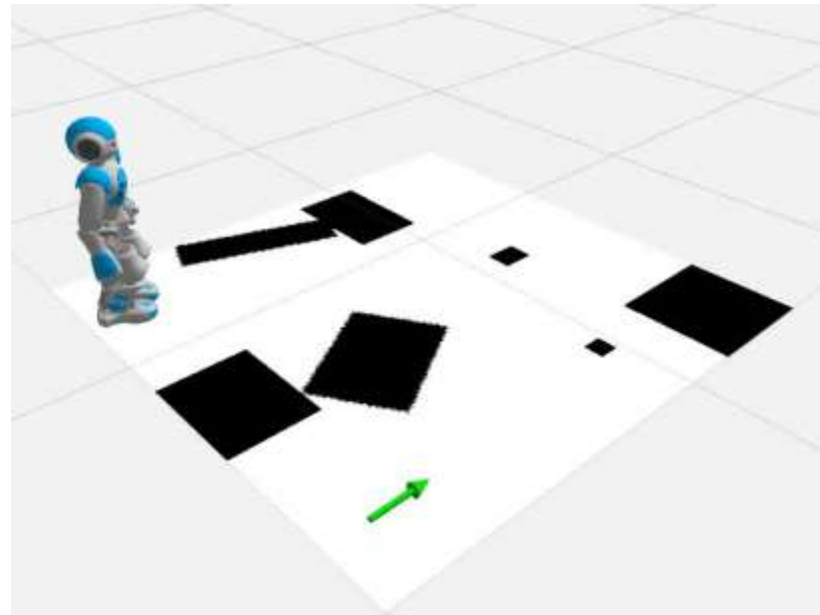
↓

Joint Angles

Pattern Generator (e.g. Kajita et al. 2003)

Footstep planning with A*

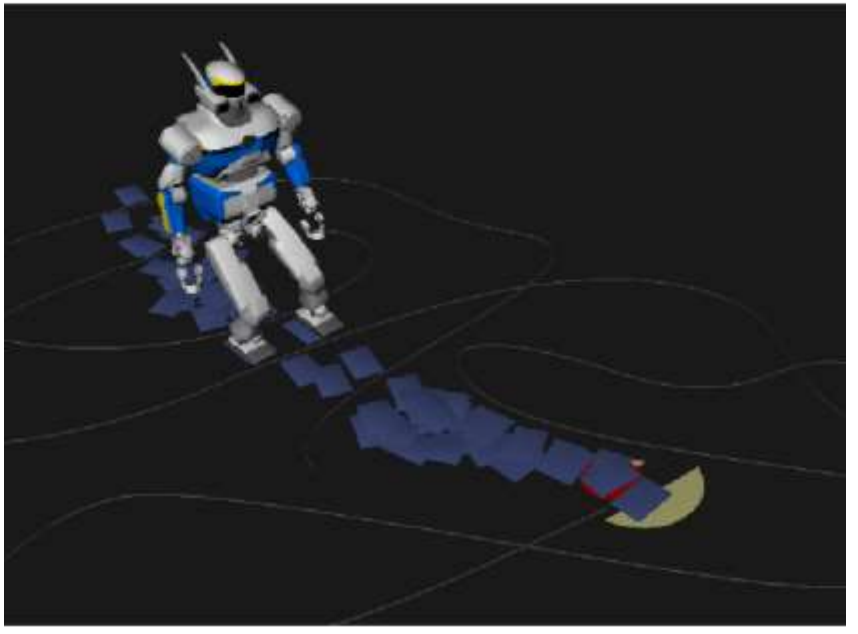
- Search space: (x,y,θ)
- Discrete set of footsteps
- Optimal solution with A*



[Kuffner '01, Chestnutt et al. '05, '07]

Randomized Footstep Planning

- Search space of footstep actions with RRT / PRM
- Fast planning results
- Enables high number of actions
- No guarantees on optimality or completeness

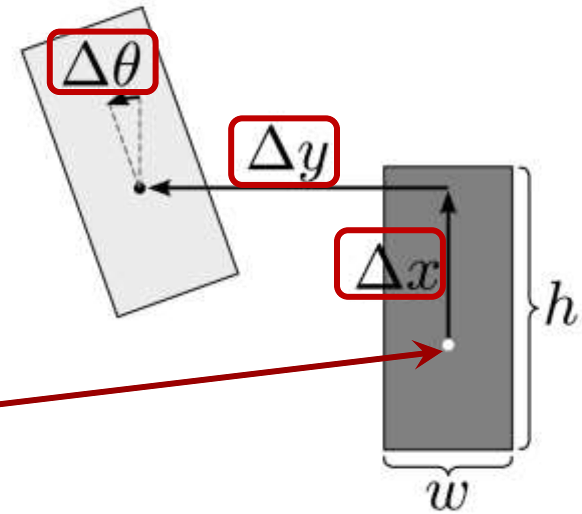


[Perrin et al. '11]

A* Heuristic Search

- Best-first search to find a cost-optimal path to a goal state
- Expands states according to the evaluation function $f(s)=g(s)+h(s)$
- $g(s)$: Costs from start to current state
- $h(s)$: Heuristic, estimated costs to the goal
- Heuristic must be admissible: it may never overestimate the costs to the goal

Footstep Planning



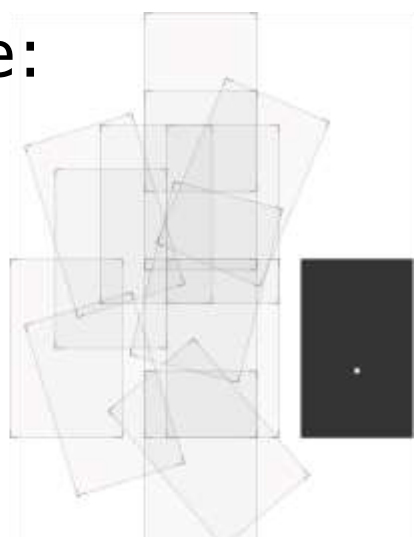
- State $s = (x, y, \theta)$
- Footstep action $a = (\Delta x, \Delta y, \Delta\theta)$
- Fixed set of footstep actions $F = \{a_1, \ldots, a_n\}$
- Successor state $s' = t(s, a)$
- Transition costs reflect execution time:

$$c(s, s') = \|(\Delta x, \Delta y)^\top\| + k + d(s')$$

Euclidean distance

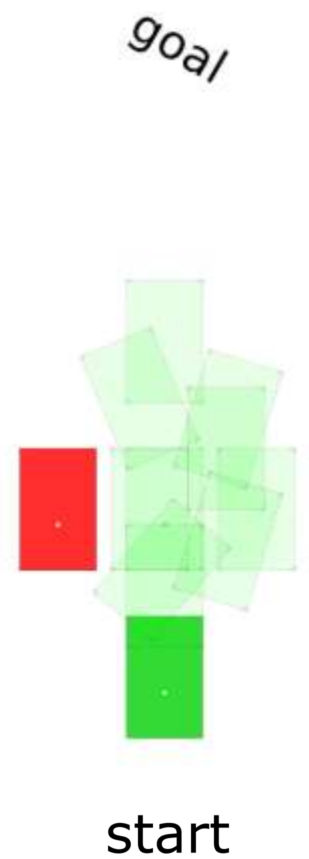
constant step cost

costs based on the distance to obstacles

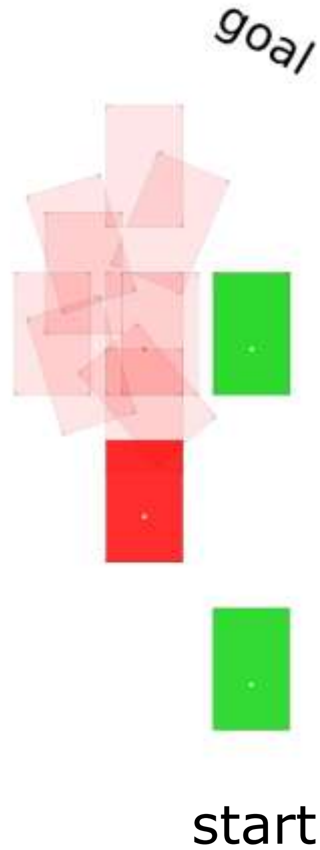


Footstep Planning

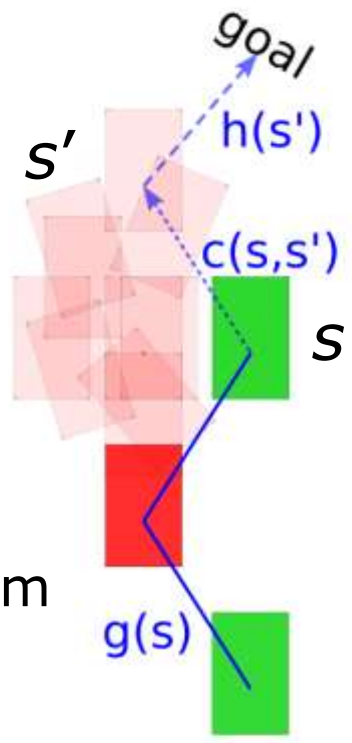
Footstep Planning



Footstep Planning



Footstep Planning



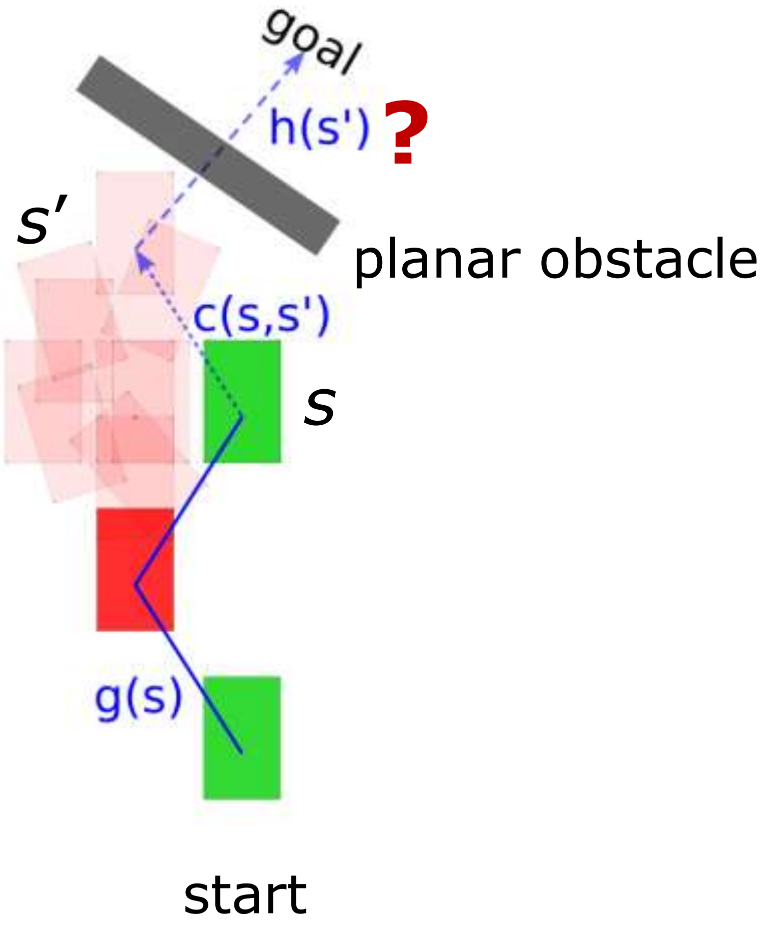
estimated costs from *s*′ to goal

transition costs

path costs from start to *s*

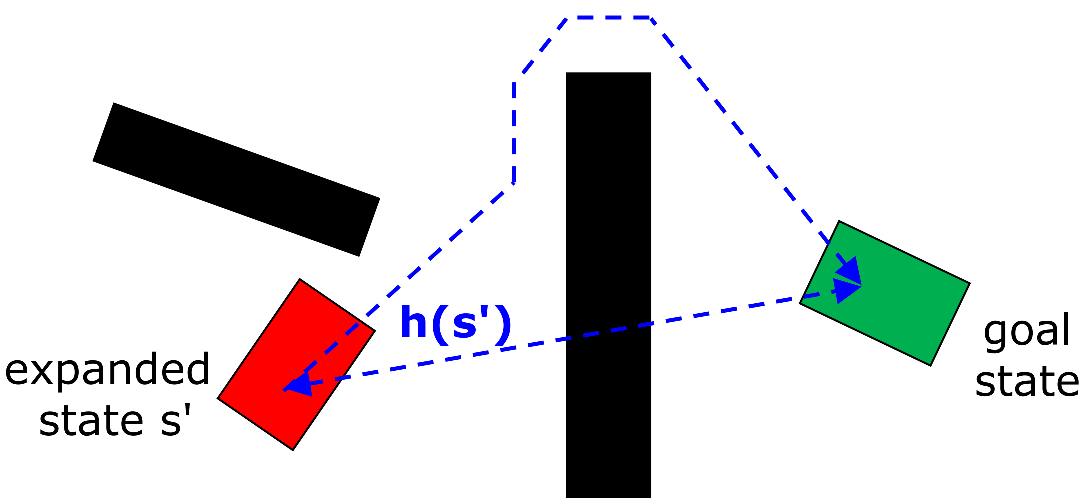
start

Footstep Planning



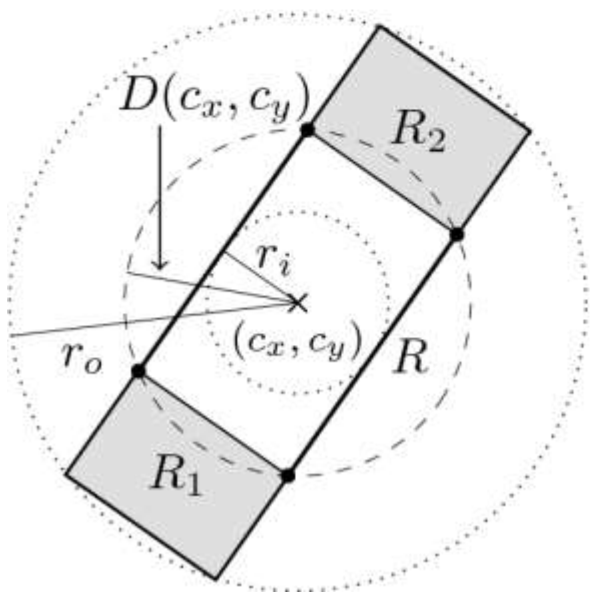
Heuristic

- Estimates the costs to the goal
- Critical for planner performance
- Usual choices:
  - Euclidean distance
  - 2D Dijkstra path



Collision Checking in 2D

- Footprint is rectangular with arbitrary orientation
- Evaluating the distance between foot center and the closest obstacle may not yield correct or optimal results
- Recursively subdivide footstep shape

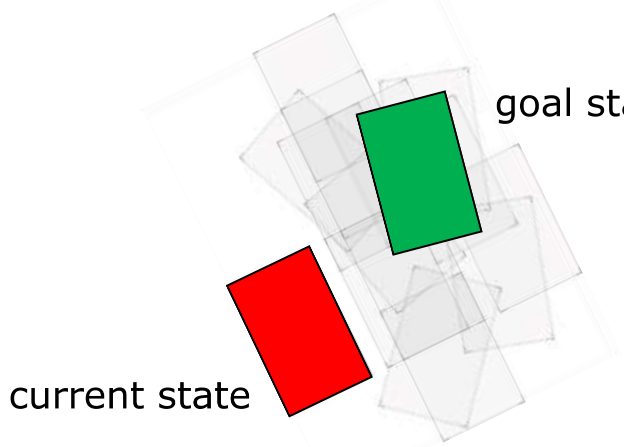


$D(c_x, c_y)$ = distance to the closest obstacle (precomputed map)

[Sprunk et al. (ICRA '11)]

Search-Based Footstep Planning

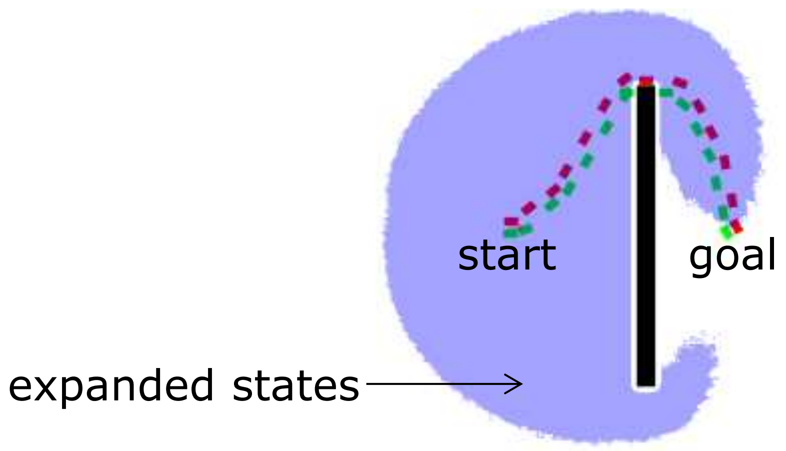
- Concatenation of footstep actions builds a lattice in the global search space
- Only valid states after a collision check are added
- Goal state may not be exactly reached, but it is sufficient to reach a state close by (within the motion range)



Search-Based Footstep Planning

- We can now apply heuristic search methods on the state lattice
- Search-based planning library: www.ros.org/wiki/sbpl
- Footstep planning implementation based on SBPL: www.ros.org/wiki/footstep_planner

Local Minima in the Search Space



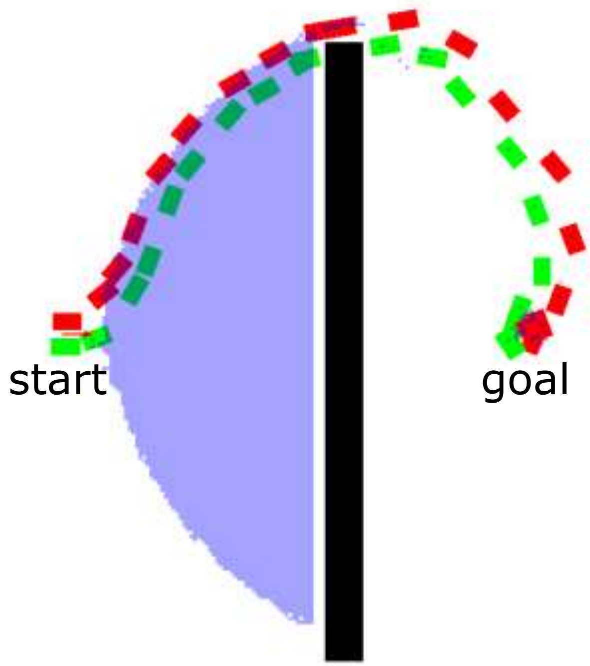
- A* will search for the optimal result
- Initially sub-optimal results are often sufficient for navigation
- Provable sub-optimality instead of randomness yields more efficient paths

Anytime Repairing A* (ARA*)

- Heuristic inflation by a factor w allows to efficiently deal with local minima: weighted A* (wA*)
- ARA* runs a series of wA* searches, iteratively lowering w as time allows
- Re-uses information from previous iterations

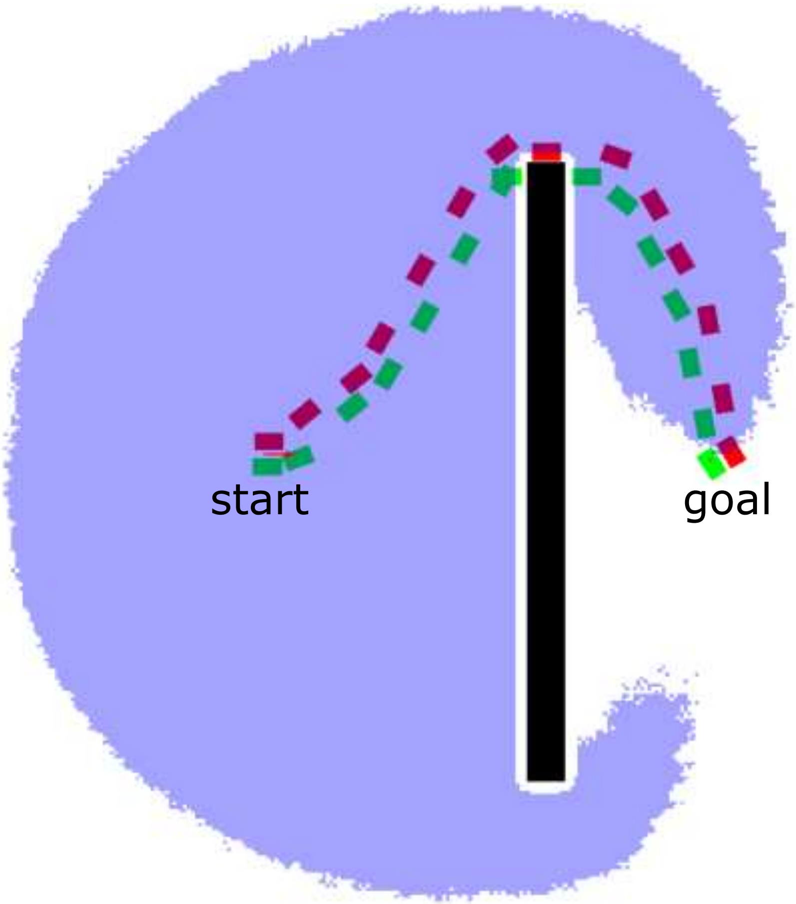
[Likhachev et al. (NIPS 2004), Hornung et al. (Humanoids 2012)]

ARA* with Euclidean Heuristic



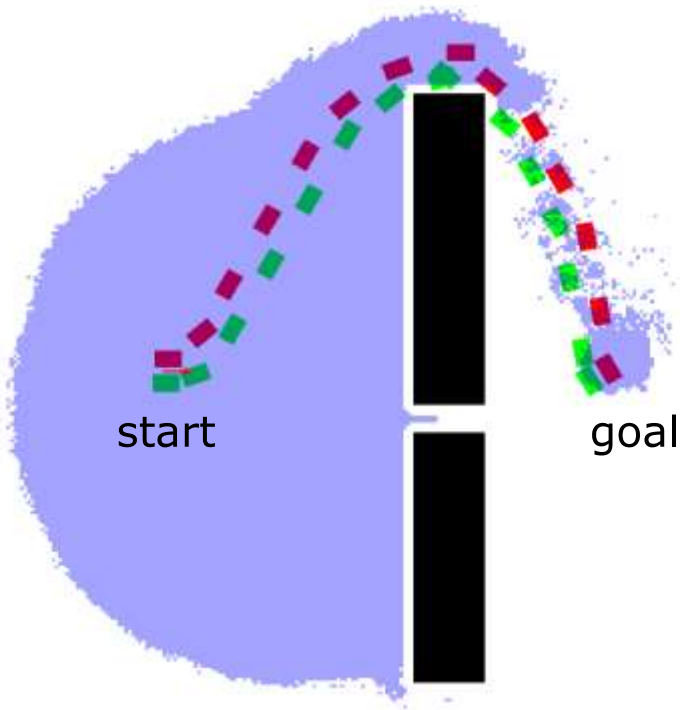
w = 10

ARA* with Euclidean Heuristic



w = 1

ARA* with Dijkstra Heuristic

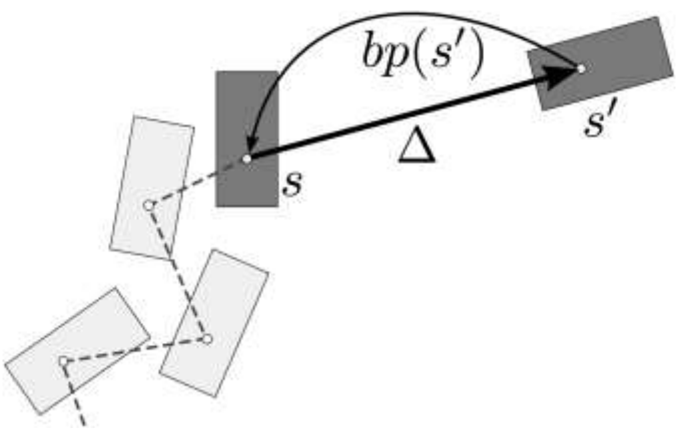


w = 1

Performance depends on well-designed heuristic

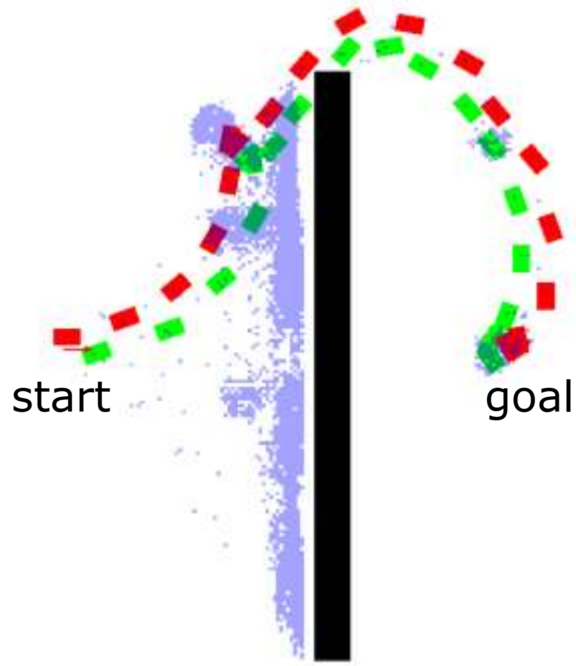
Randomized A* (R*)

- Iteratively constructs a graph of sparsely placed randomized sub-goals (exploration)
- Plans between sub-goals with wA*, preferring easy-to-plan sequences
- Iteratively lowers w as time allows



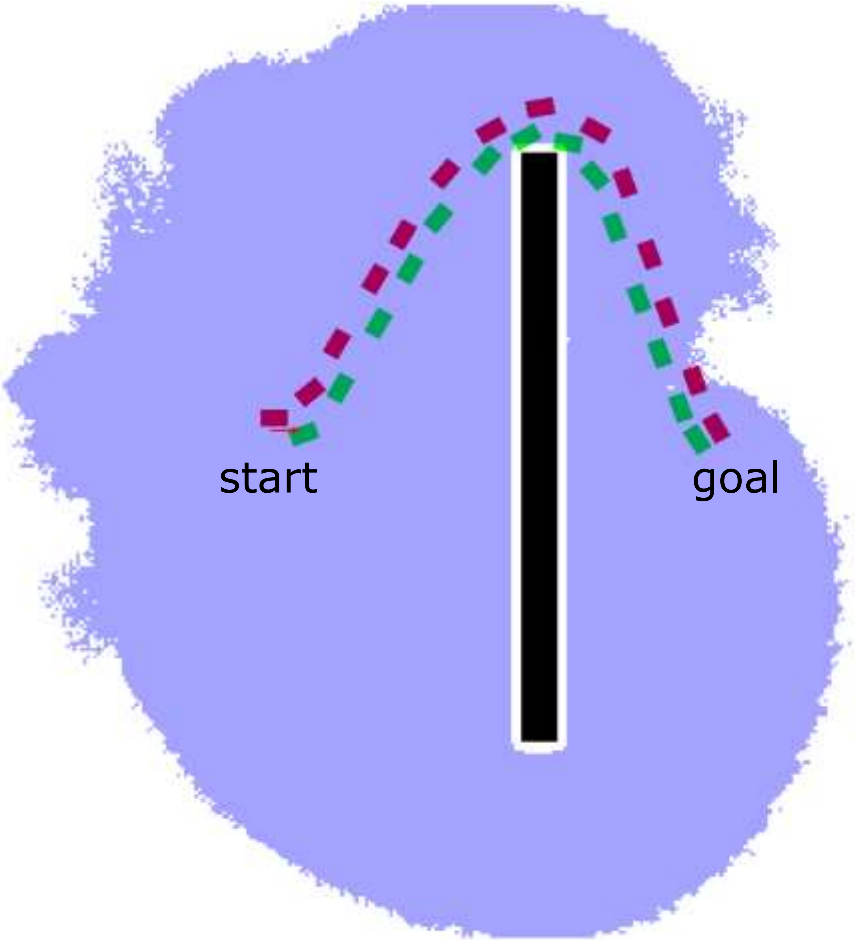
[Likhachev & Stentz (AAAI 2008), Hornung et al. (Humanoids 2012)]

R* with Euclidean Heuristic



w = 10

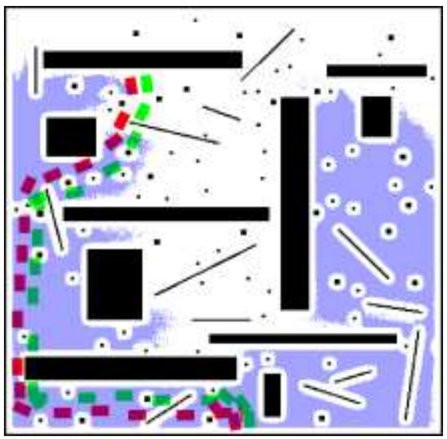
R* with Euclidean Heuristic



w = 1

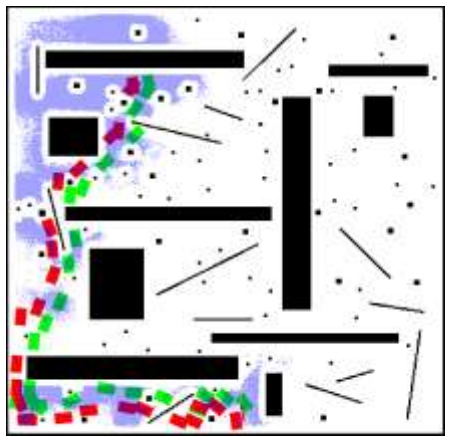
Planning in Dense Clutter Until First Solution

A*
Euclidean heur.



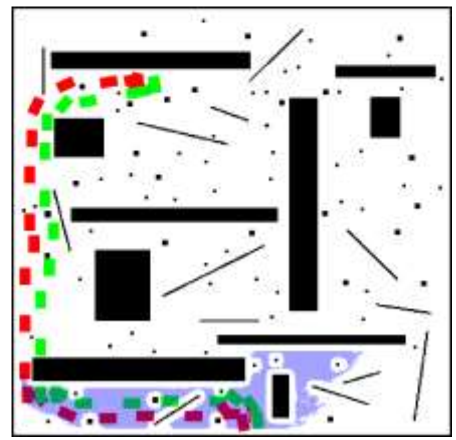
11.9 sec.

R*
Euclidean heur.



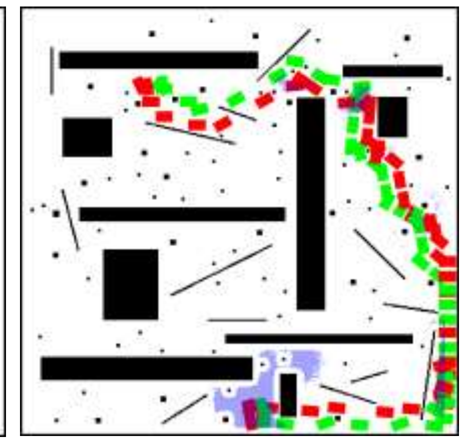
0.4 sec.

ARA*
Euclidean heur.



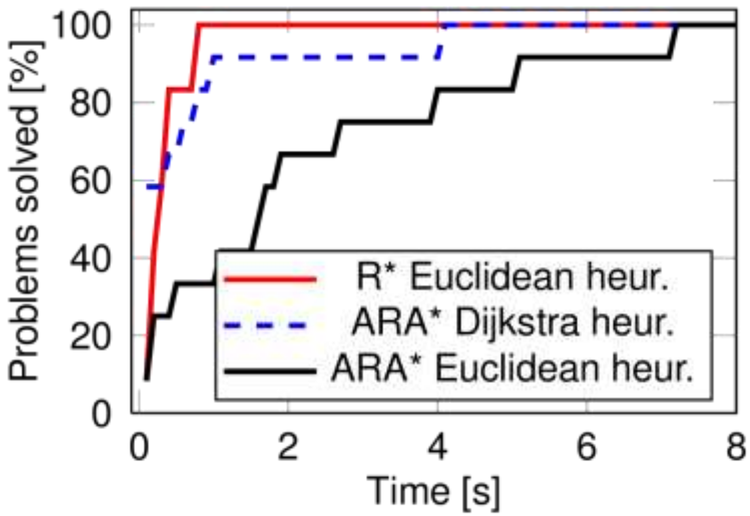
2.7 sec.

ARA*
Dijkstra heur.



0.7 sec.

Planning in Dense Clutter Until First Solution



Planner	Heuristic	Planning time [s]	Path costs
R* (w=5)	Euclidean	0.32 ± 0.23	16.45 ± 3.16
ARA* (w=5)	Euclidean	2.15 ± 2.21	13.57 ± 1.15
ARA* (w=5)	2D Dijkstra	0.56 ± 1.13	20.41 ± 5.08
A* (w=1)	Euclidean	33.31 ± 15.00	11.06 ± 1.20

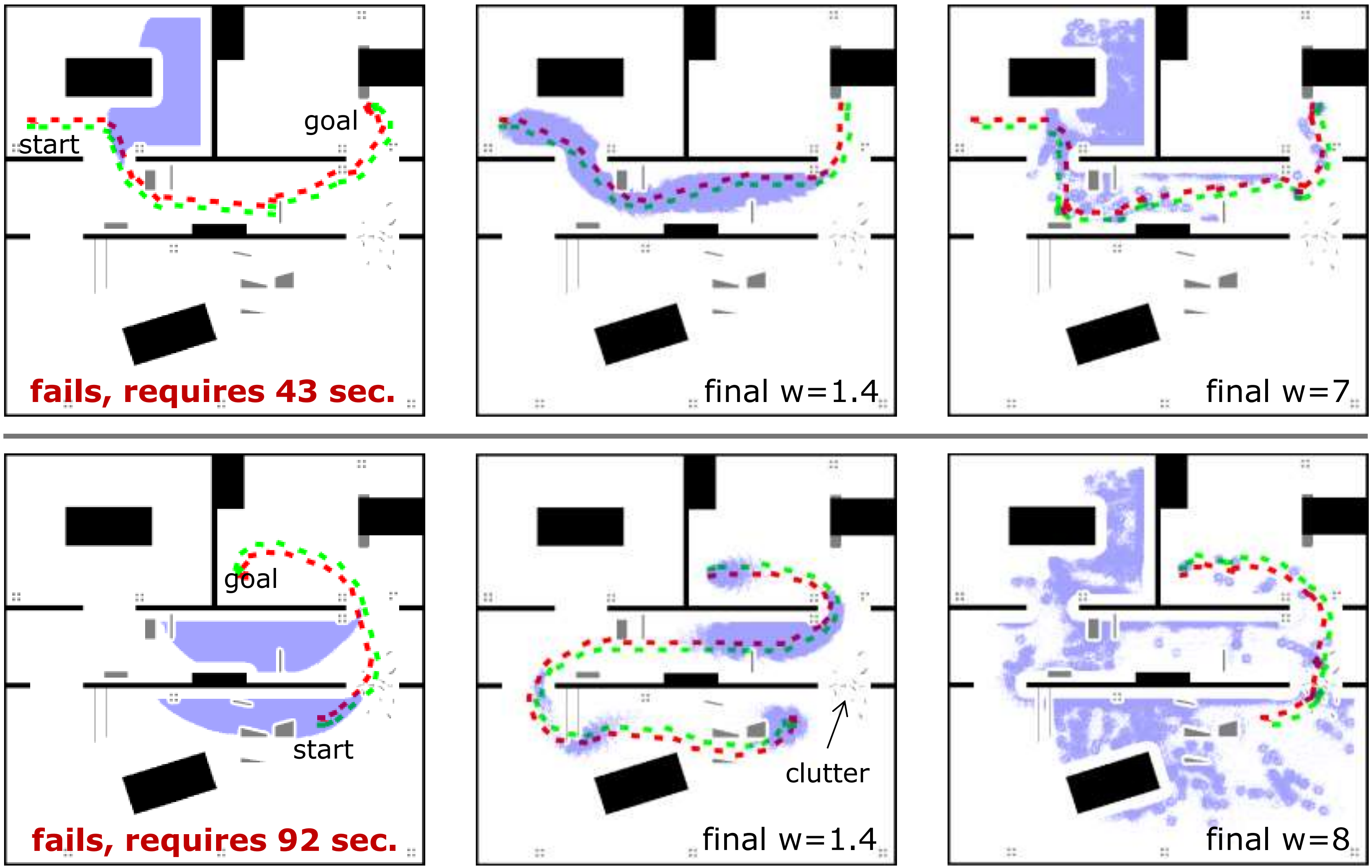
- 12 random start and goal locations
- ARA* finds fast results only with the 2D Dijkstra heuristic, leading to longer paths due to its inadmissibility
- **R* finds fast results even with the Euclidean heuristic**

Planning with Time Limit 5s

ARA*
Euclidean heuristic

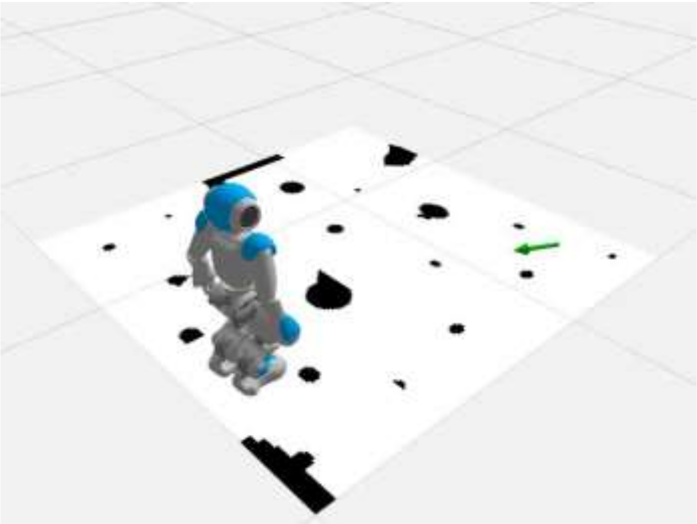
ARA*
Dijkstra heuristic

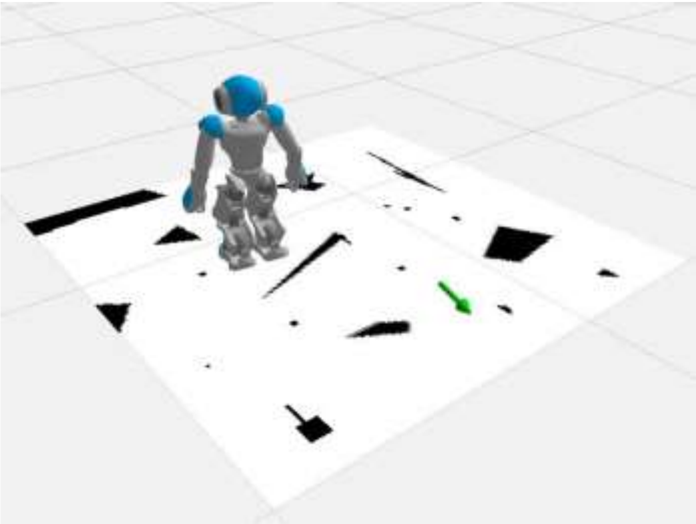
R*
Euclidean heuristic



Anytime Planning Results

- Performance of ARA* depends on well-designed heuristic
- Dijkstra heuristic may be inadmissible and can lead to wrong results
- R* with the Euclidean heuristic finds efficient plans in short time



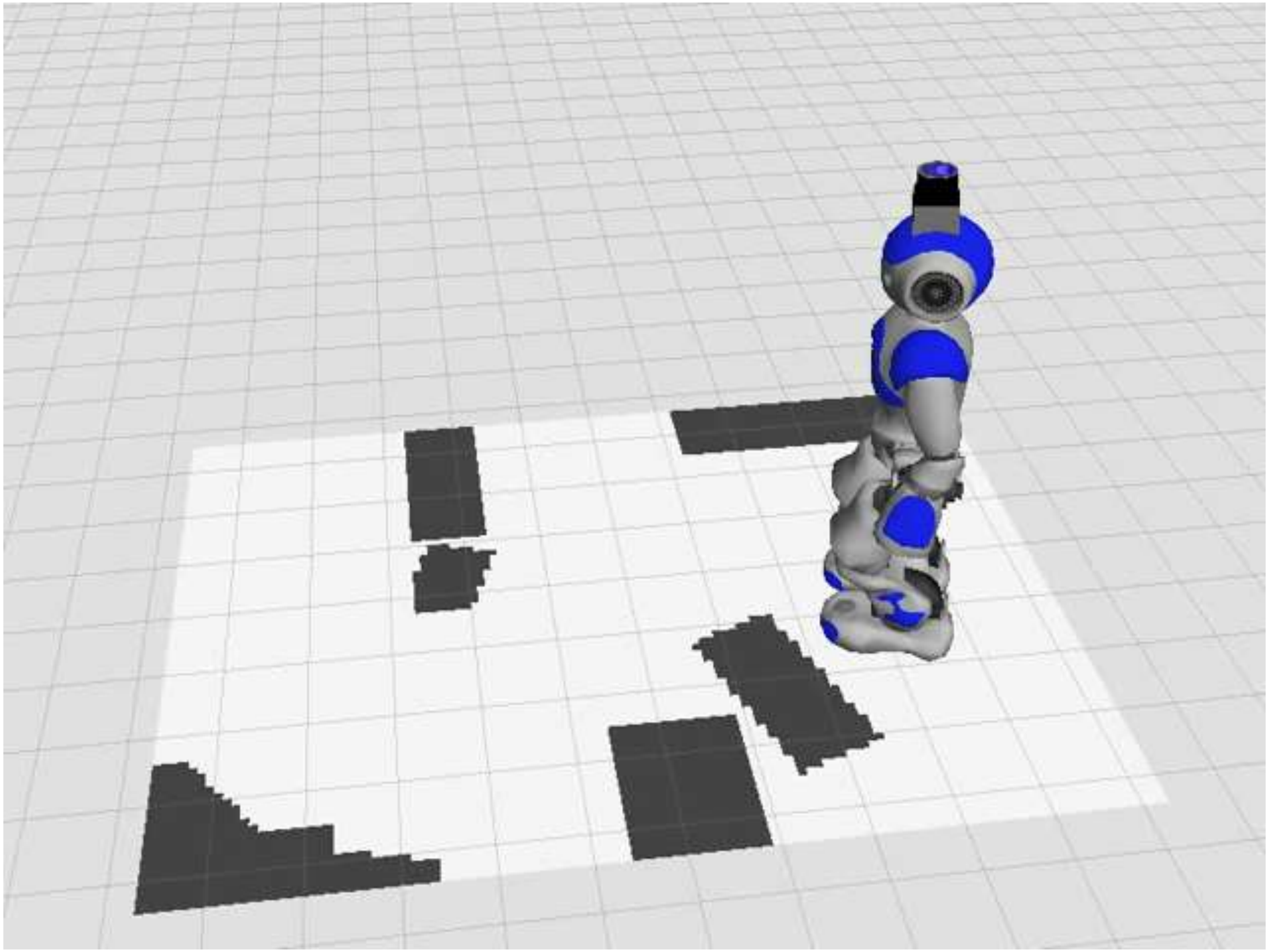


Dynamic A* (D*)

- Allows for efficient re-planning in case of
  - Changes in the environment
  - Deviations from the initial path
- Re-uses state information from previous searches
- Planning backwards increases the efficiency in case of updated localization estimates
- Anytime version: AD*

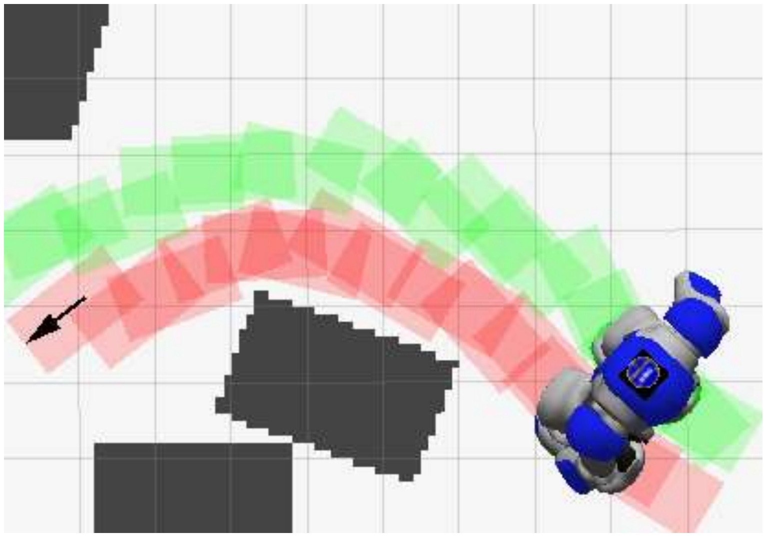
[Koenig & Likhachev (AAAI ‘00), Garimort (ICRA ’11)]

D* Plan Execution with a Nao

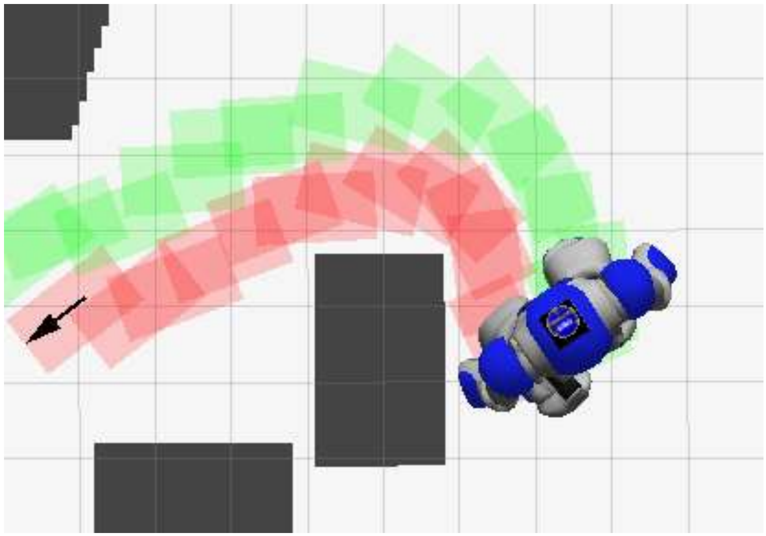


Efficient Replanning

- Plans may become invalid due to changes in the environment
- D* allows for efficient plan re-usage



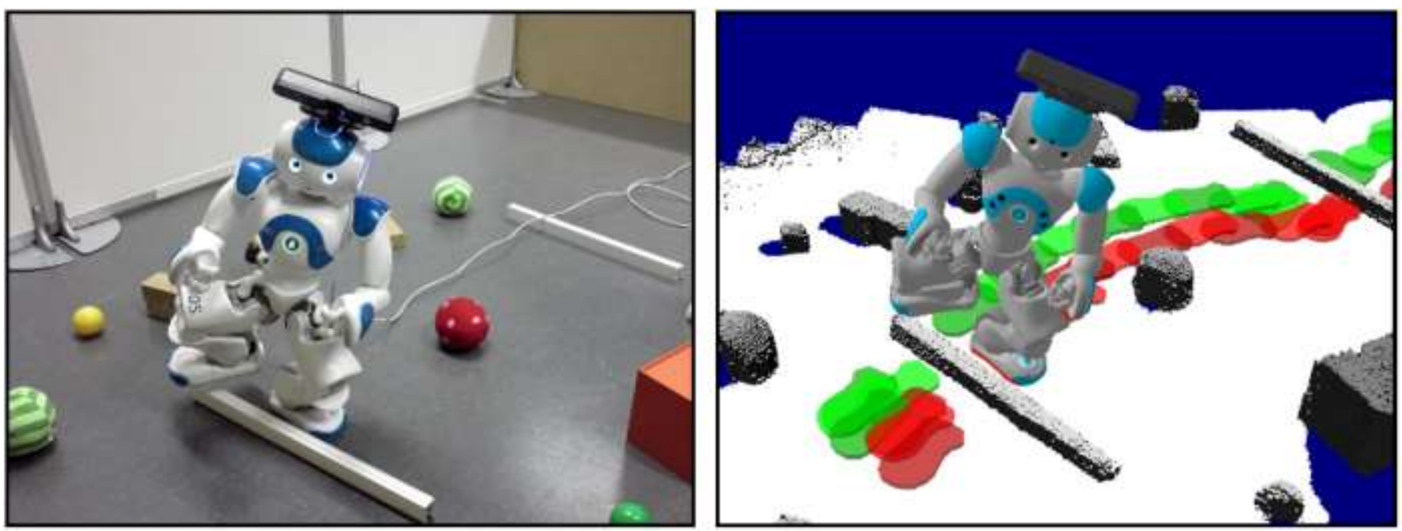
2966 states, 1.05s



956 states, 0.53s

Extension to 3D

- Depth camera for visual perception
- Scan matching to reduce drift of odometry
- Heightmap as environment representation
- Footstep planning and collision-checking on heightmap



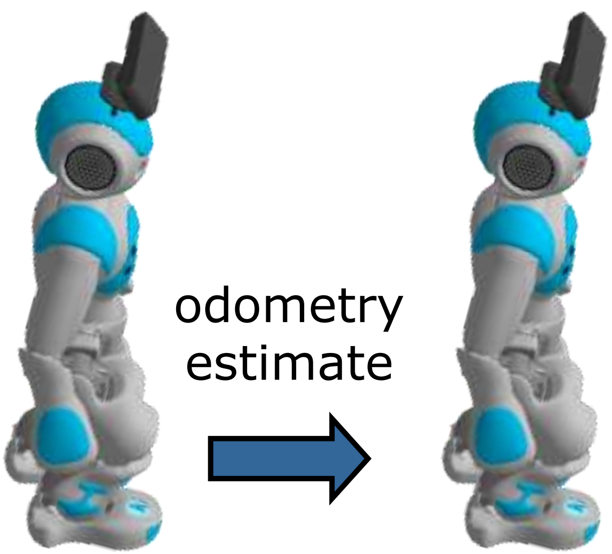
Maier et al. (to appear IROS 2013)]

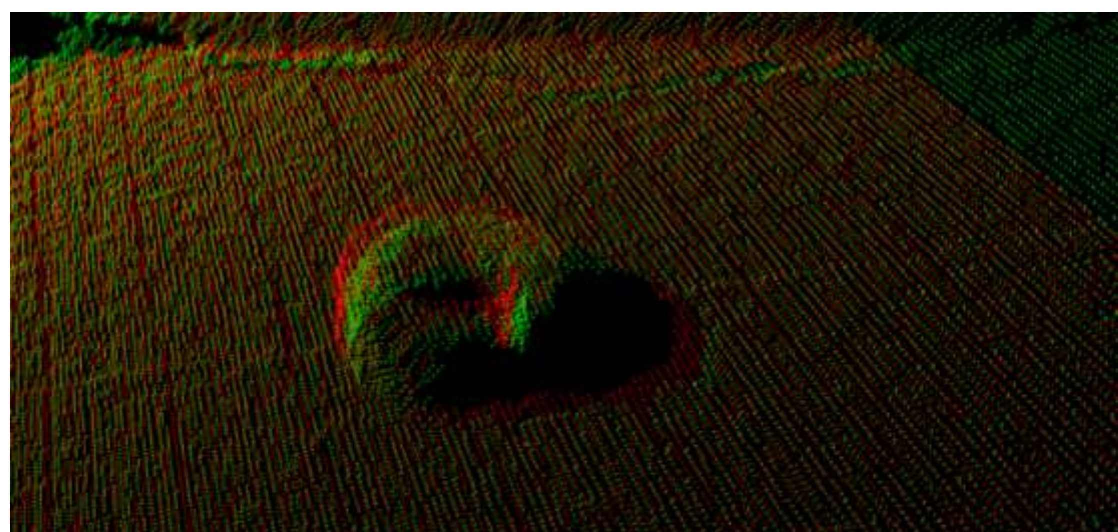
Depth Cameras for Robot Navigation

- Dense depth information
- Lightweight
- Cheap

Pose Estimation

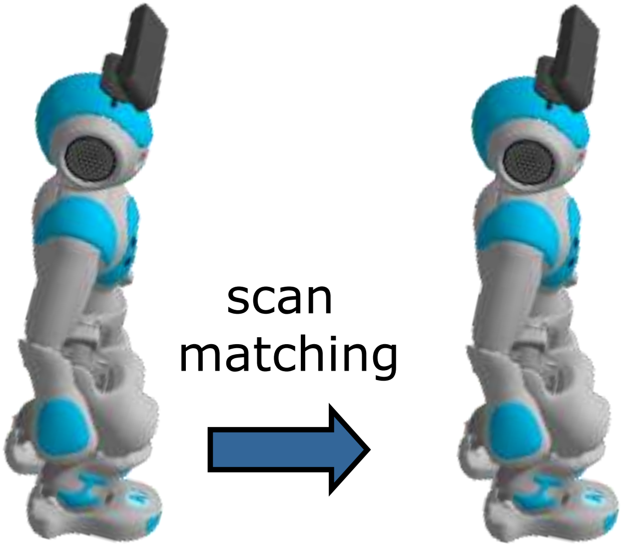
- Odometry estimate is error-prone due to slippage of the feet and noisy sensors
- Accordingly, consecutive depth camera observations may not align
- Error accumulates over time
- Scan matching to reduce the error

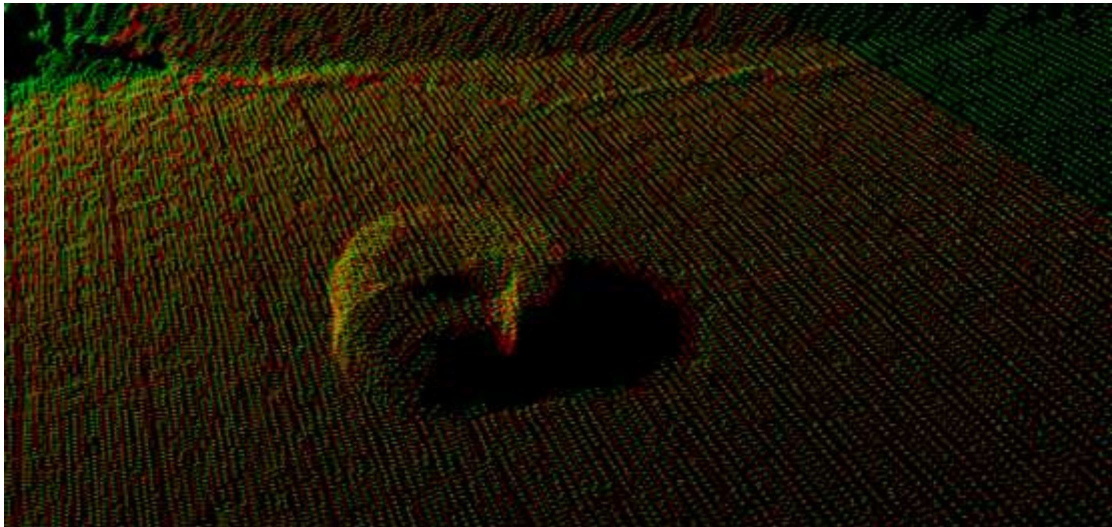




Pose Estimation

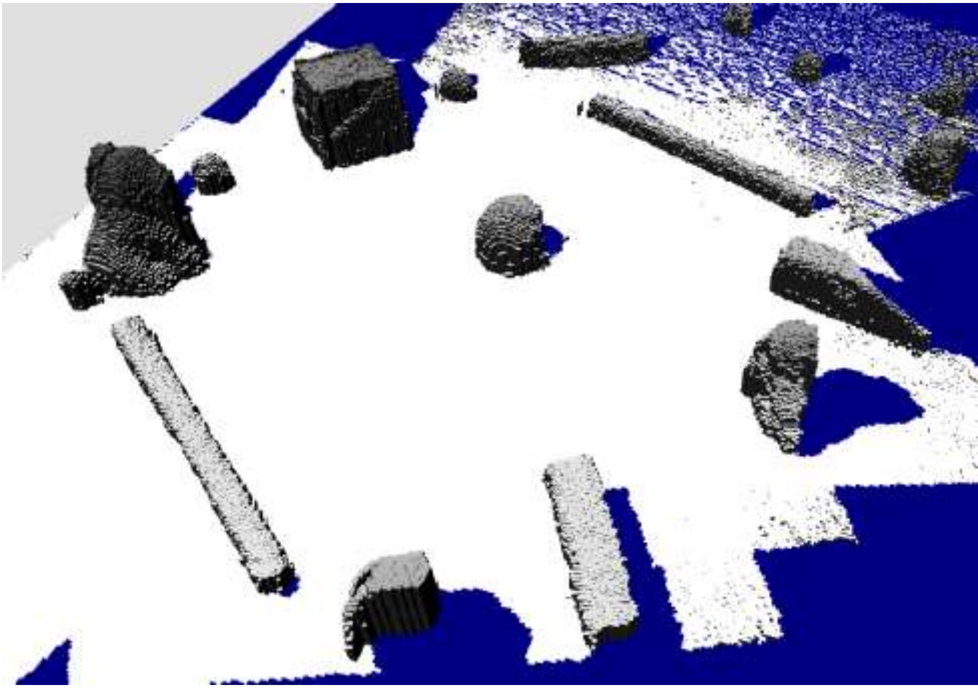
- Odometry estimate is error prone due to slippage of the feet and noisy sensors
- Accordingly, consecutive depth camera observations may not align
- Error accumulates over time
- Scan matching to reduce the error



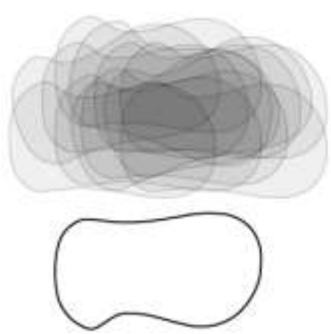


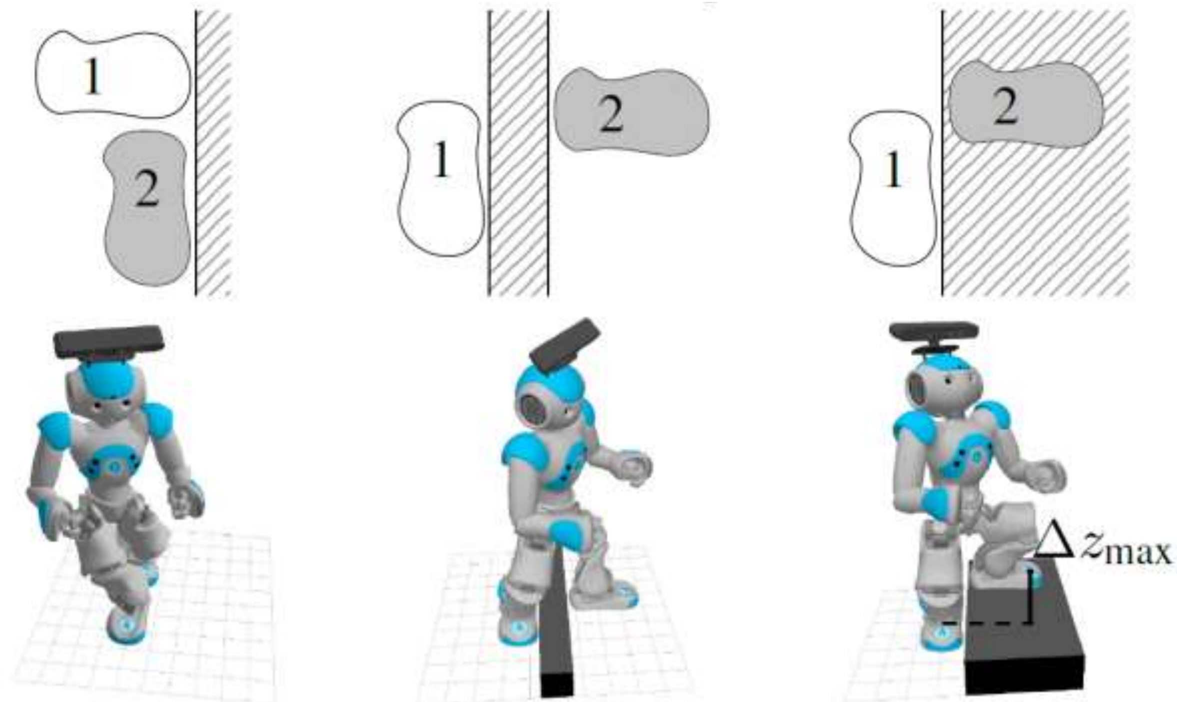
Heightmap Learned from Depth-Camera Observations

- 2D gridmap
- Probabilistic height estimate for each cell
- Conservative updates
- Quick access
- Memory efficient
- High resolution



Action Set for a Nao Humanoid





T-Step step over step onto

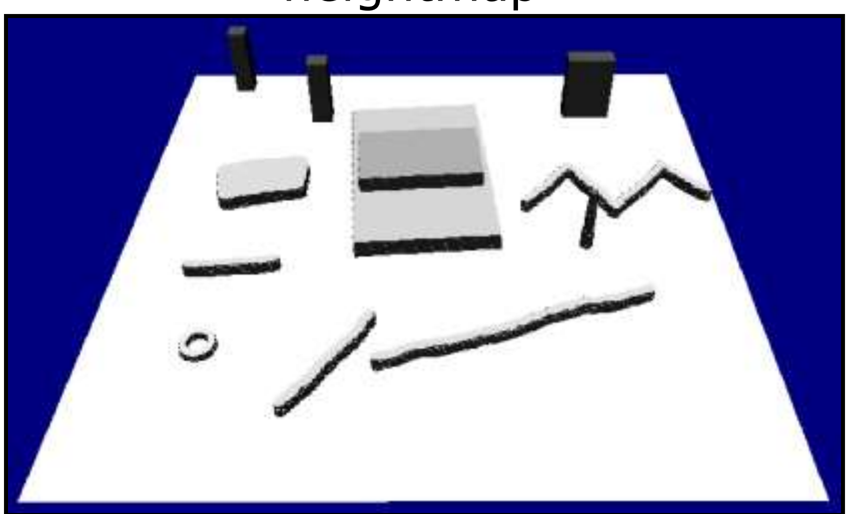
Standard planar steps

Extended 3D stepping capabilities

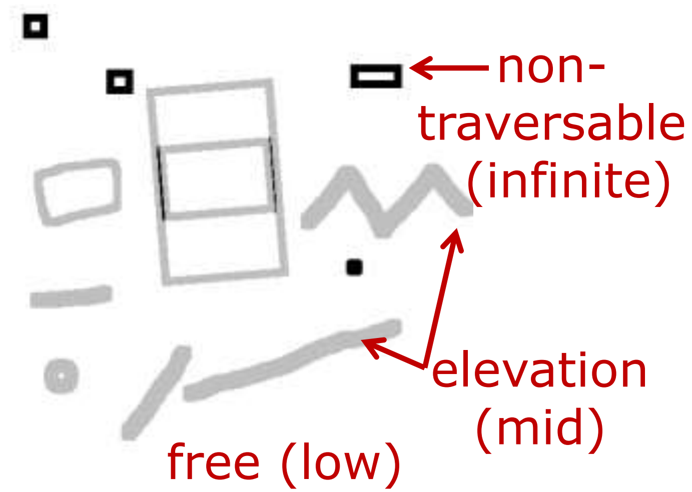
Dijkstra Heuristic for Heightmaps

- Graph G=(V,E)
  - V: discrete locations in the (x,y)-space
  - E: union of 8-neighborhoods in the state space
- Costs of an edge are defined by the height differences in the heightmap
- h(s): shortest path in G to the goal / $v_{\max}$

heightmap

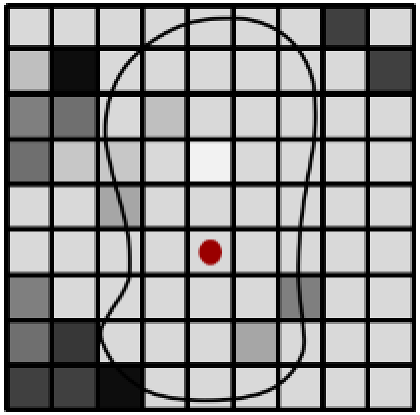


example costs

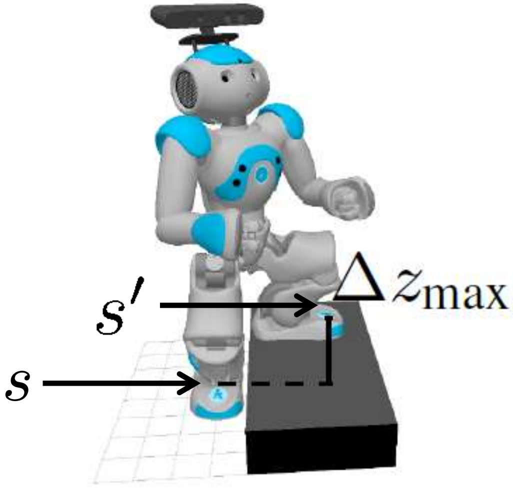


Safe Stepping Actions

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



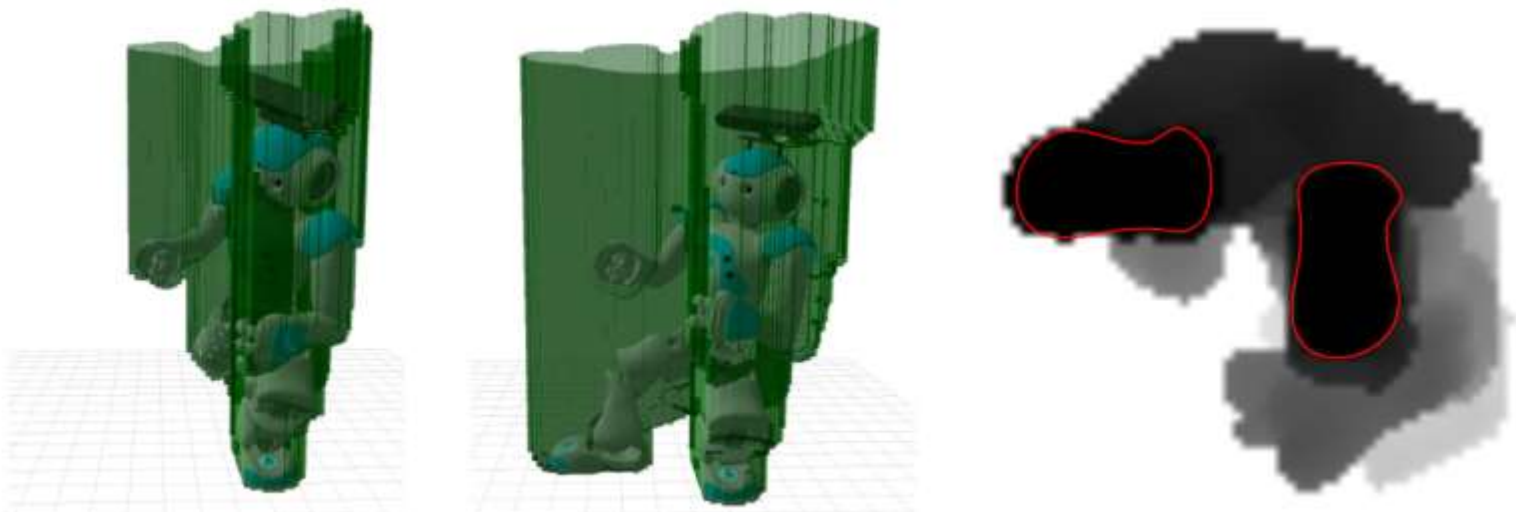
- Height difference between s and $s' = a(s)$ must be within the limits $[\Delta z_{\mathrm{min}}, \Delta z_{\mathrm{max}}]^a$ allowed by the action a



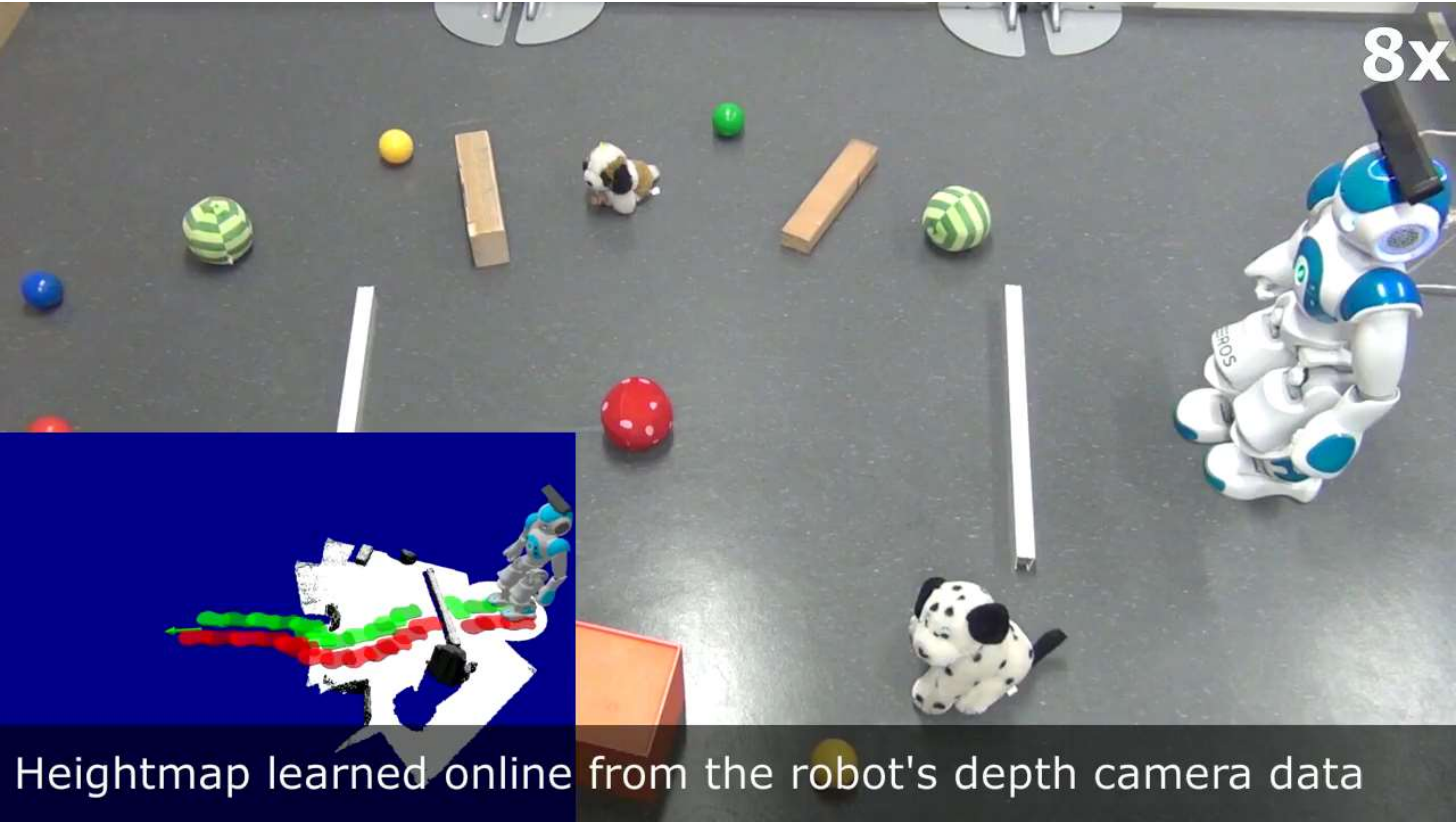
Whole-Body Collision Checking

- Project swept volume of a motion to the ground plane: inverse heightmap (IHM)
- An action a at state s is safe if

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

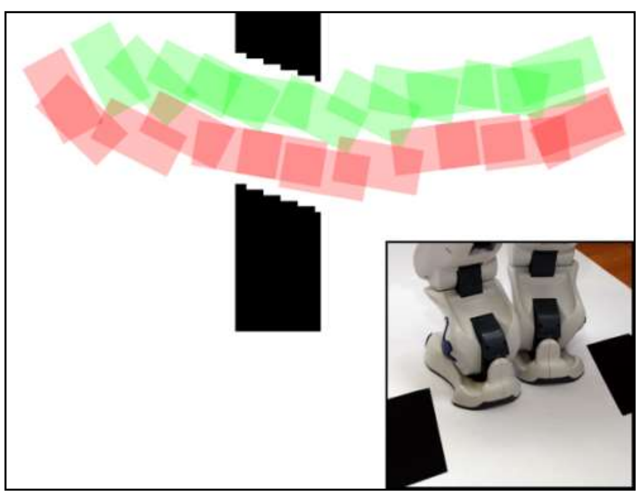


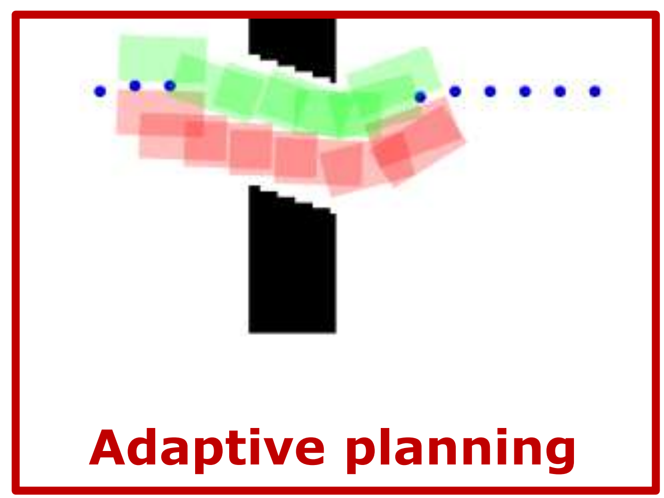
Navigation Experiments



Adaptive Level-of-Detail Planning

- Planning the whole path with footsteps may not always be desired in large open spaces
- Adaptive level-of-detail planning: Combine fast grid-based 2D planning in open spaces with footstep planning near obstacles





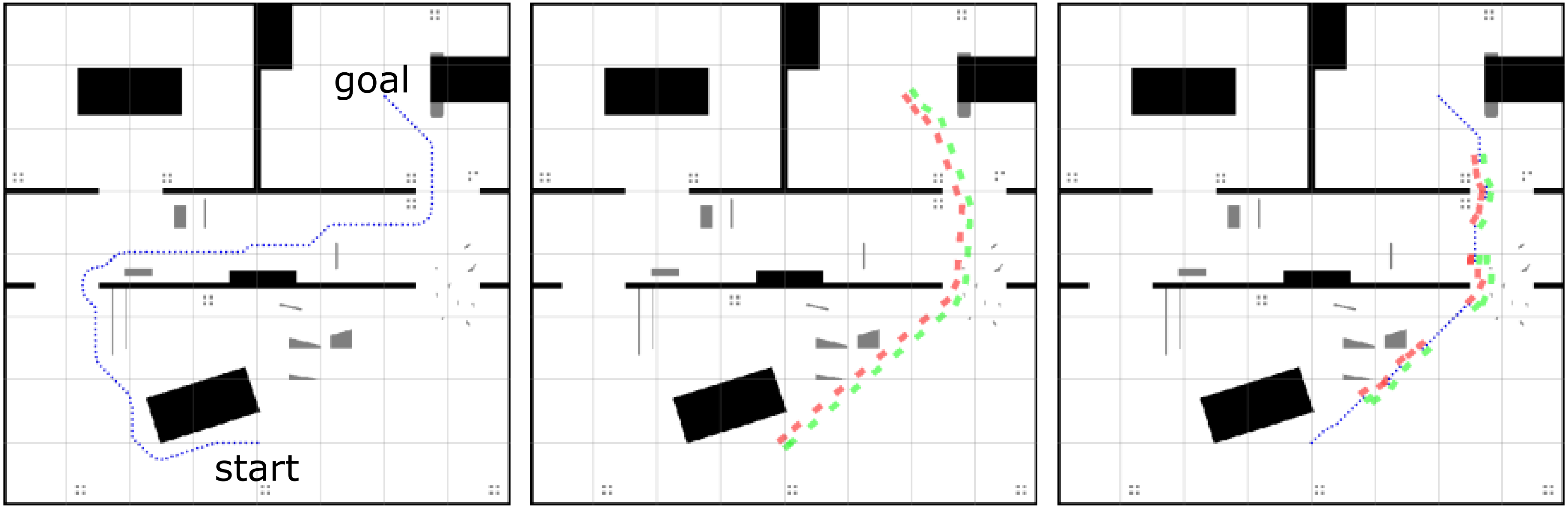
[Hornung & Bennewitz (ICRA ‘11)]

Adaptive Level-of-Detail Planning

- Allow transitions between neighboring cells in free areas and between sampled contour points across obstacle regions
- Traversal costs are estimated from a pre-planning stage or with a learned heuristic
- Every obstacle traversal triggers a footstep plan

Adaptive Planning Results

2D Planning | **Footstep Planning** | **Adaptive Planning**



<1 s planning time
High path costs

29 s planning time

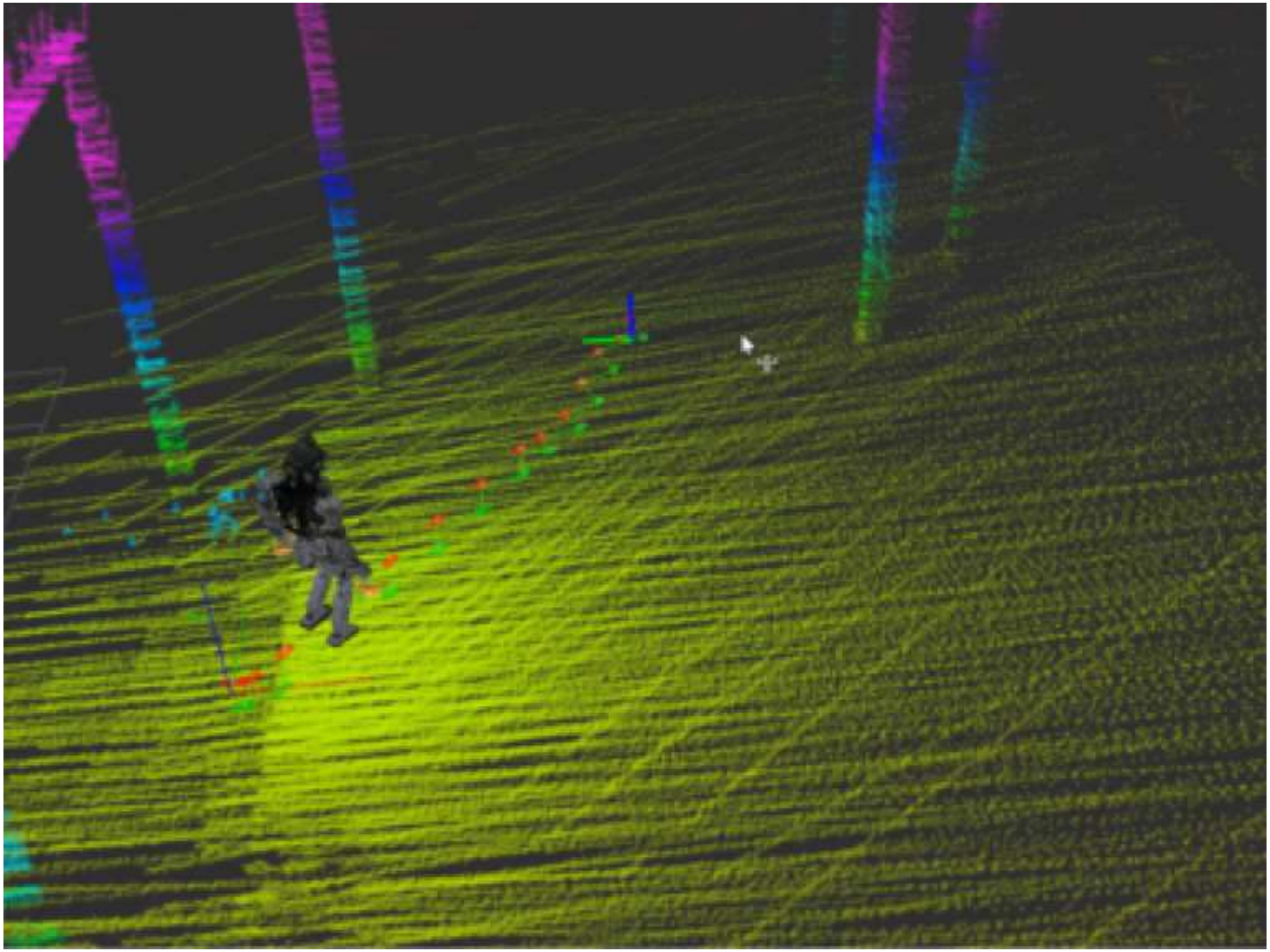
<1s planning time, costs only 2% higher

Fast planning times and efficient solutions with adaptive level-of-detail planning

Summary

- Anytime search-based footstep planning with suboptimality bounds: ARA* and R*
- Replanning during navigation with AD*
- Heuristic influences planner behavior
- Adaptive level-of-detail planning to combine 2D with footstep planning
- Extensions to 3D obstacles
- Available open source in ROS: www.ros.org/wiki/footstep_planner

Example: ATLAS humanoid in DRC (Team ViGIR)



Thank you!

Live Demo

- Install prerequisites:
  `sudo apt-get install ros-groovy-desktop-full python-rosdep python-rosinstall ros-groovy-sbpl`
- Follow rosinstall instructions at http://ros.org/wiki/humanoid_navigation (but don‘t compile)
- Compile with `rosmake footstep_planner`
- Start with `roslaunch footstep_planner footstep_planner_complete.launch`