16-350
Planning Techniques for Robotics

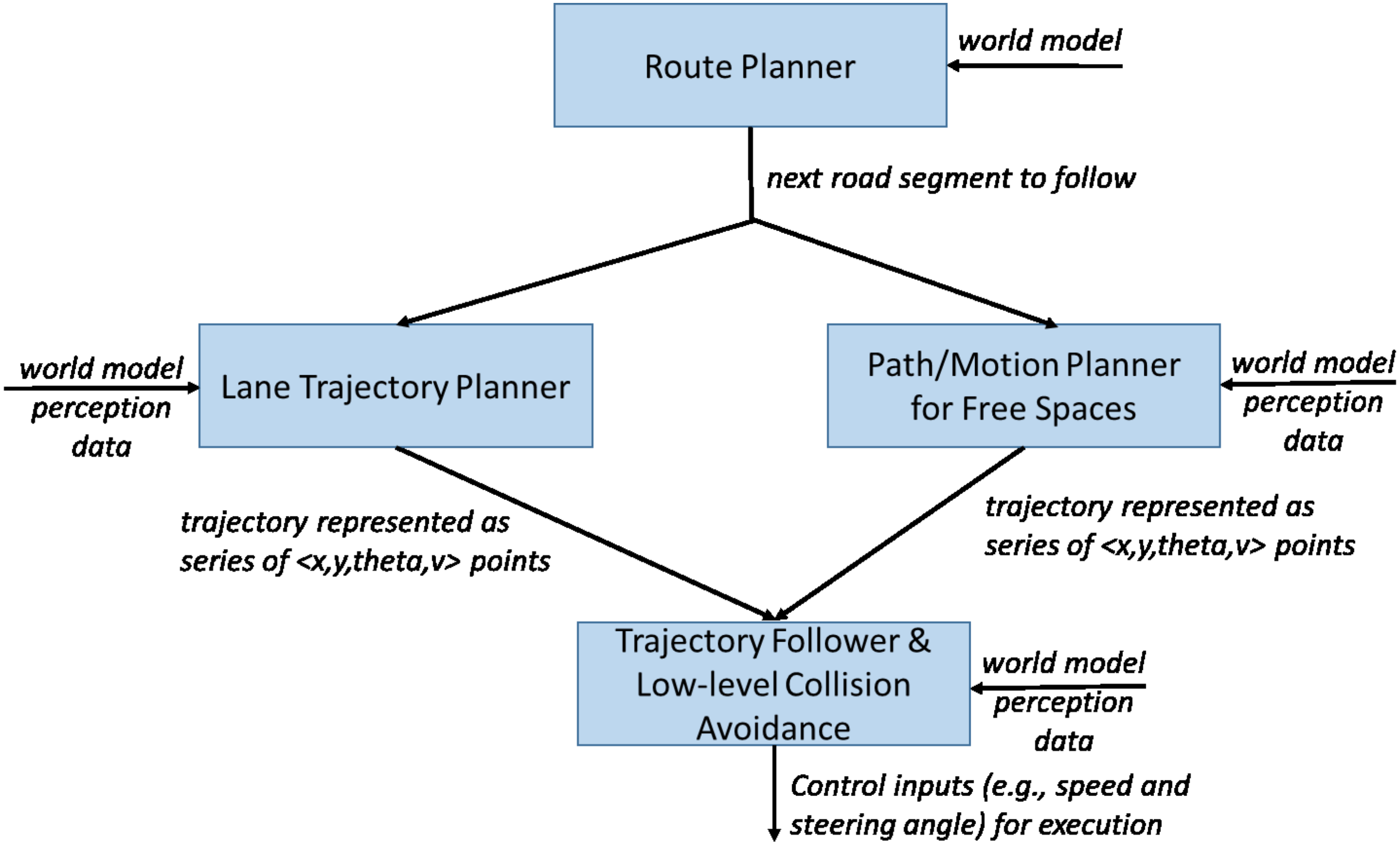
Case Study:
Planning for Autonomous Driving

Maxim Likhachev

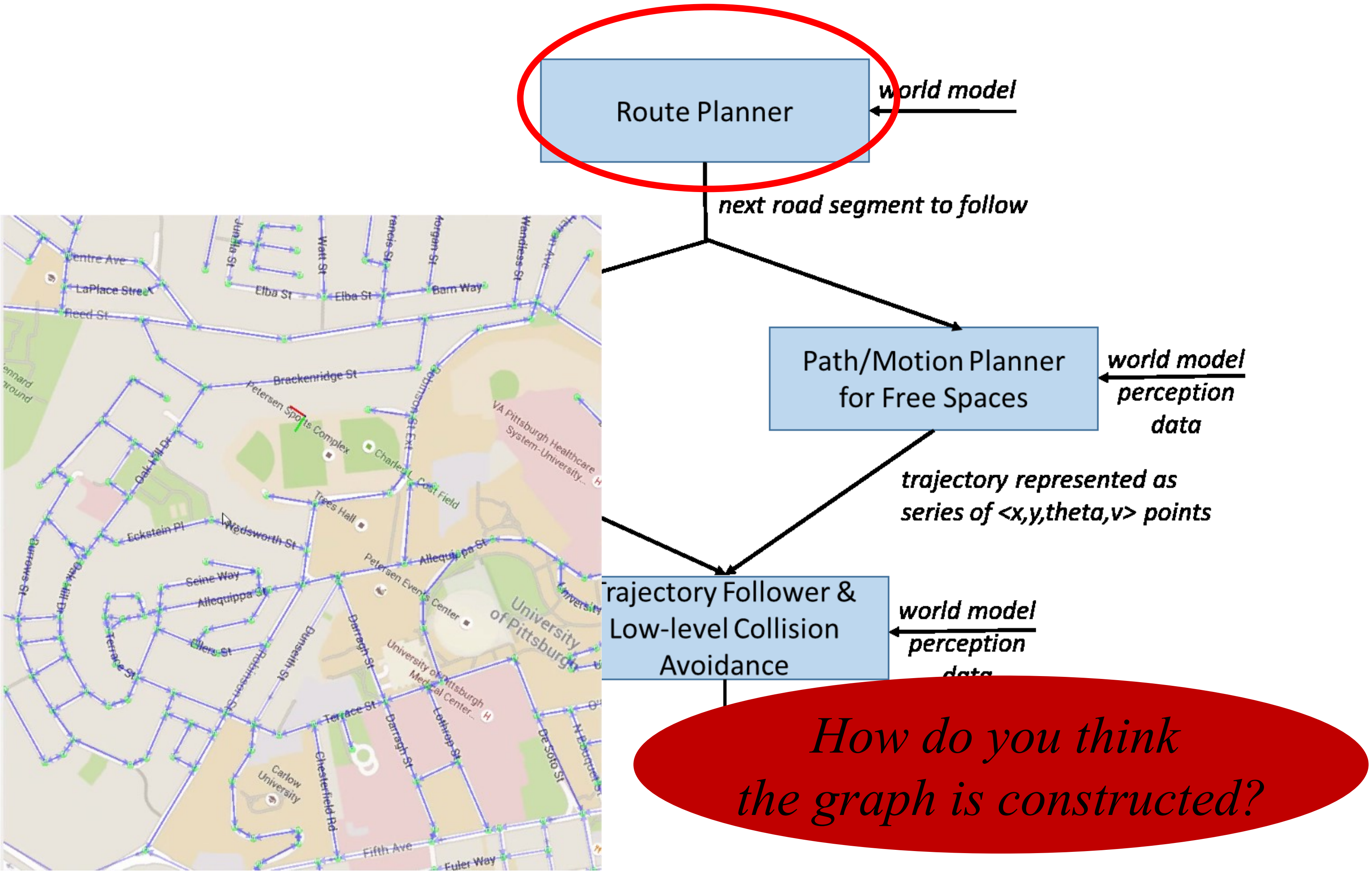
Robotics Institute

Carnegie Mellon University

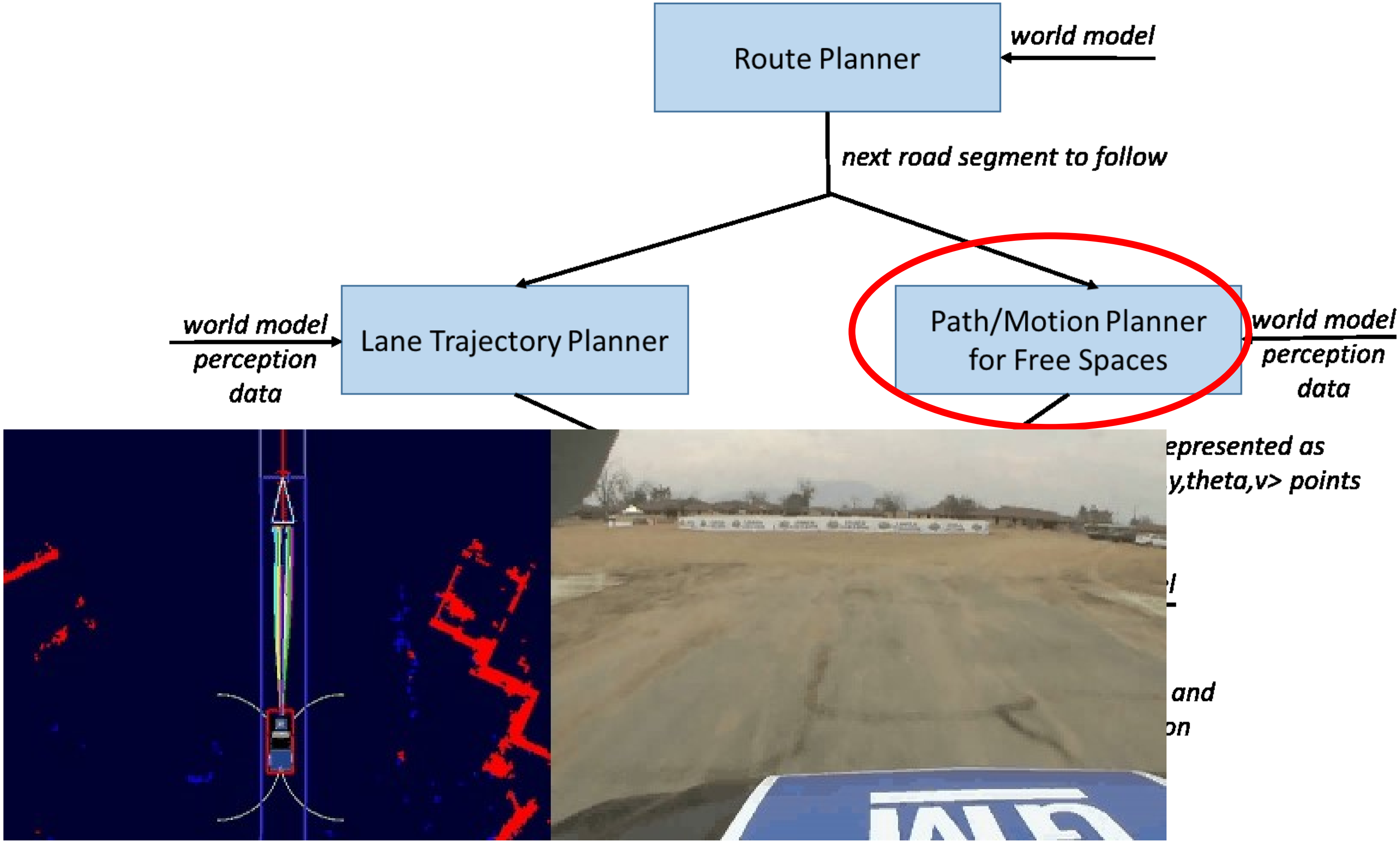
Typical Planning Architecture for Autonomous Vehicle



Typical Planning Architecture for Autonomous Vehicle

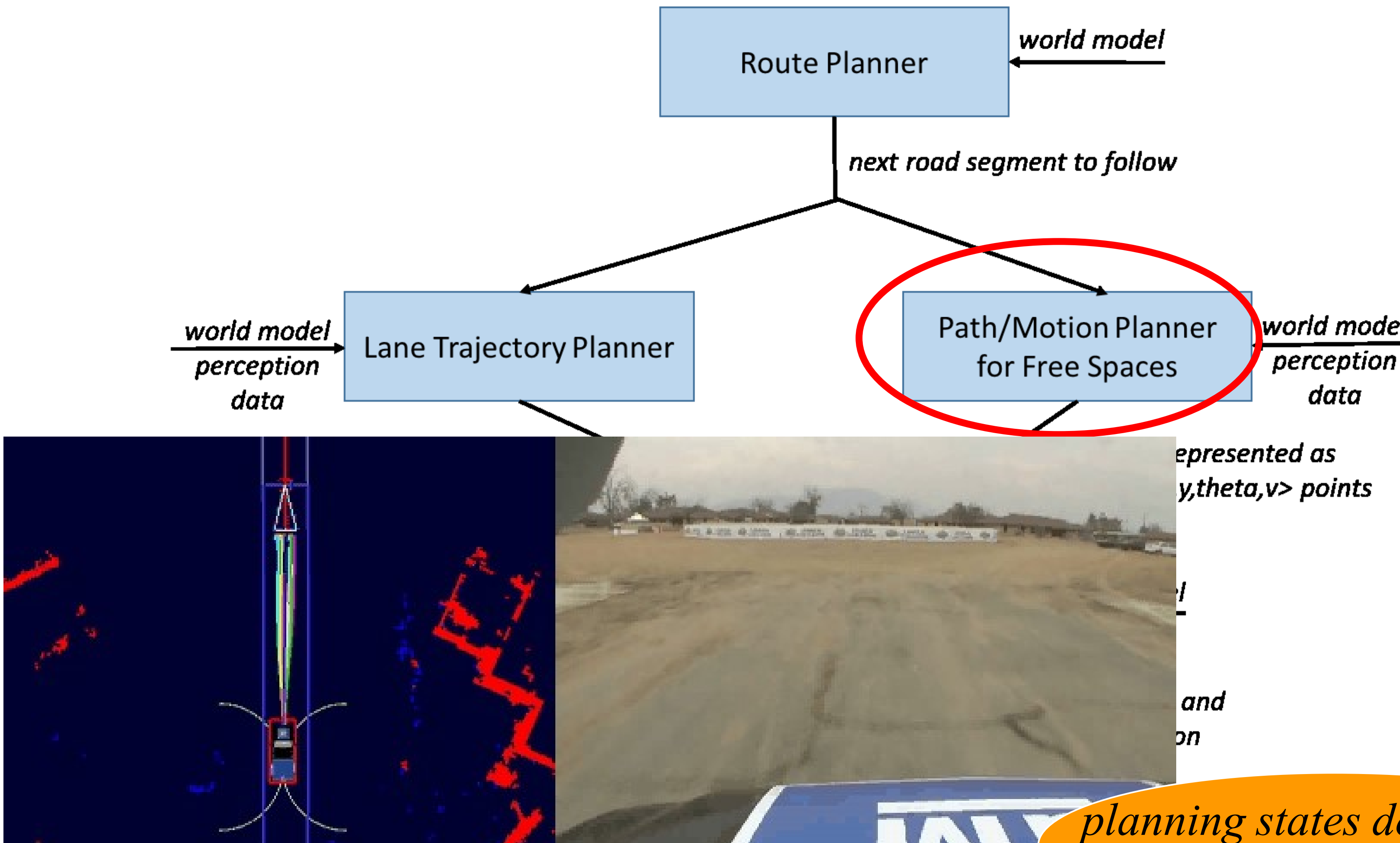


Typical Planning Architecture for Autonomous Vehicle



Tartanracing, CMU

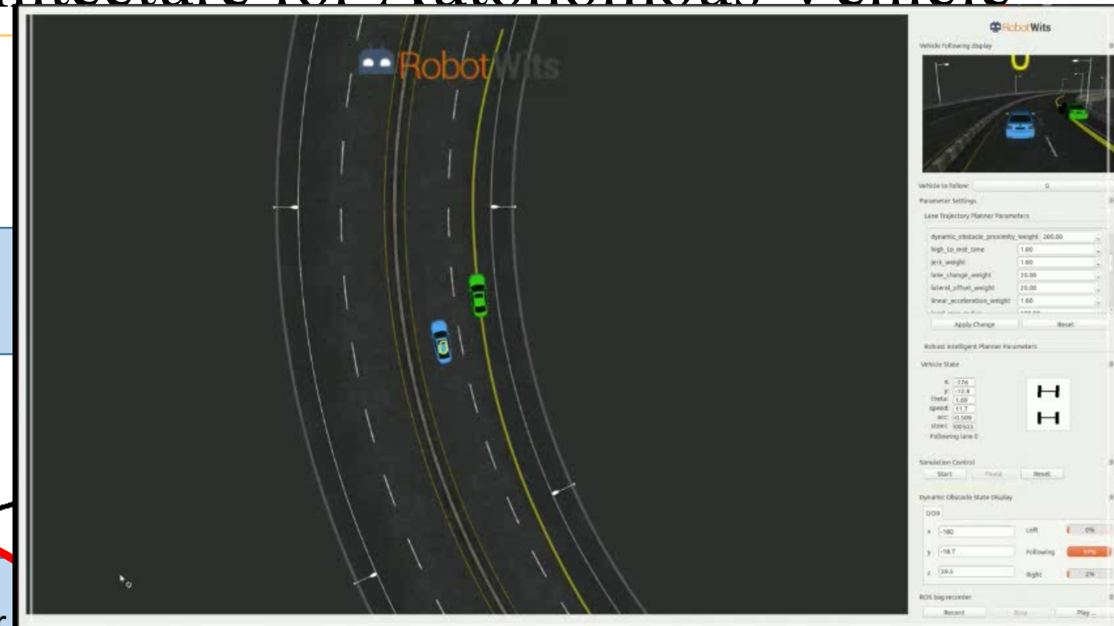
Typical Planning Architecture for Autonomous Vehicle



Tartanracing, CMU

planning states defined by:
x,y,Θ,v

Typical Planning Architecture for Autonomous Vehicle



world mode[l]
perception
data

Lane Trajectory Planner

for Free Spaces

perception
data

trajectory represented as
series of <x,y,theta,v> points

trajectory represented as
series of <x,y,theta,v> points

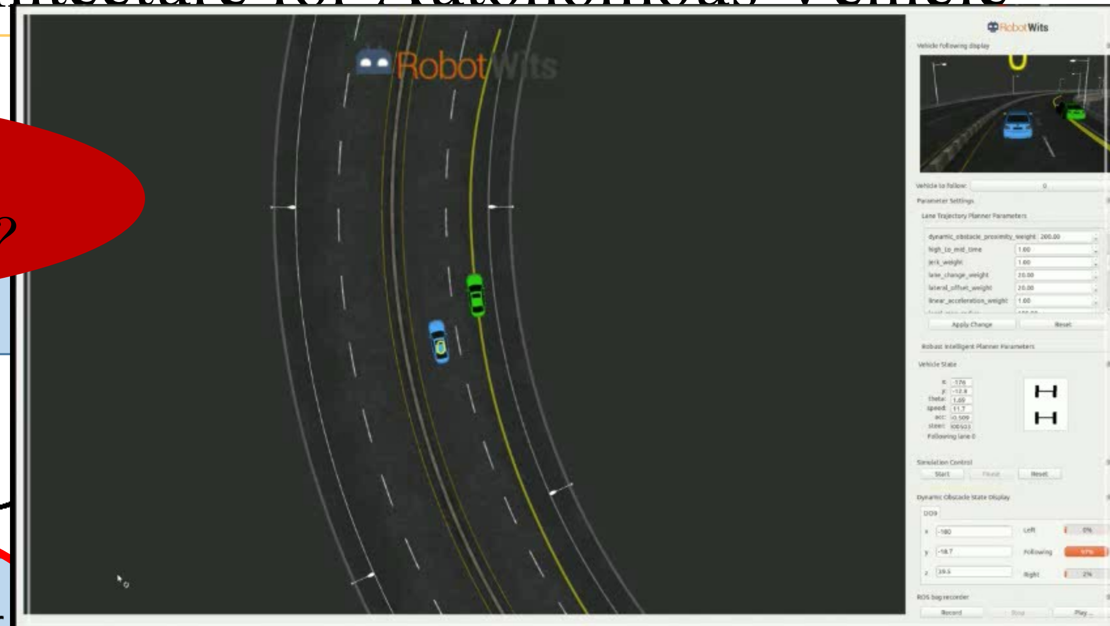
Trajectory Follower & Low-level Collision Avoidance

world model
perception
data

Control inputs (e.g., speed and
steering angle) for execution

Typical Planning Architecture for Autonomous Vehicle

How do you think the graph is constructed?



world model
perception data

Lane Trajectory Planner

for Free Spaces

perception data

trajectory represented as series of <x,y,theta,v> points

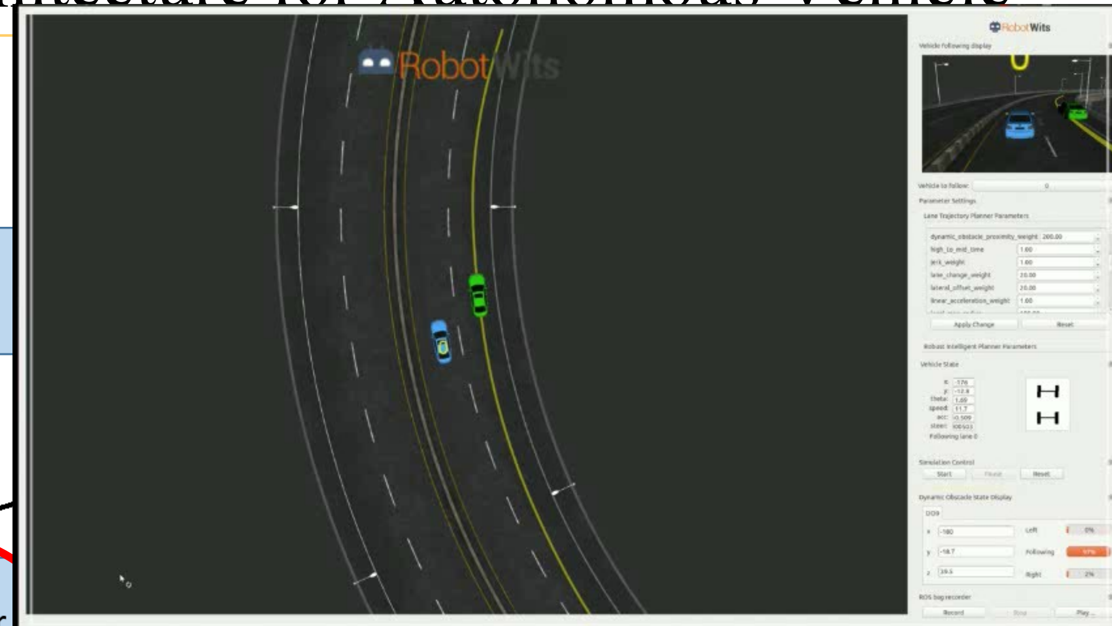
trajectory represented as series of <x,y,theta,v> points

Trajectory Follower & Low-level Collision Avoidance

world model
perception data

Control inputs (e.g., speed and steering angle) for execution

Typical Planning Architecture for Autonomous Vehicle



world model perception data

Lane Trajectory Planner

for Free Spaces

perception

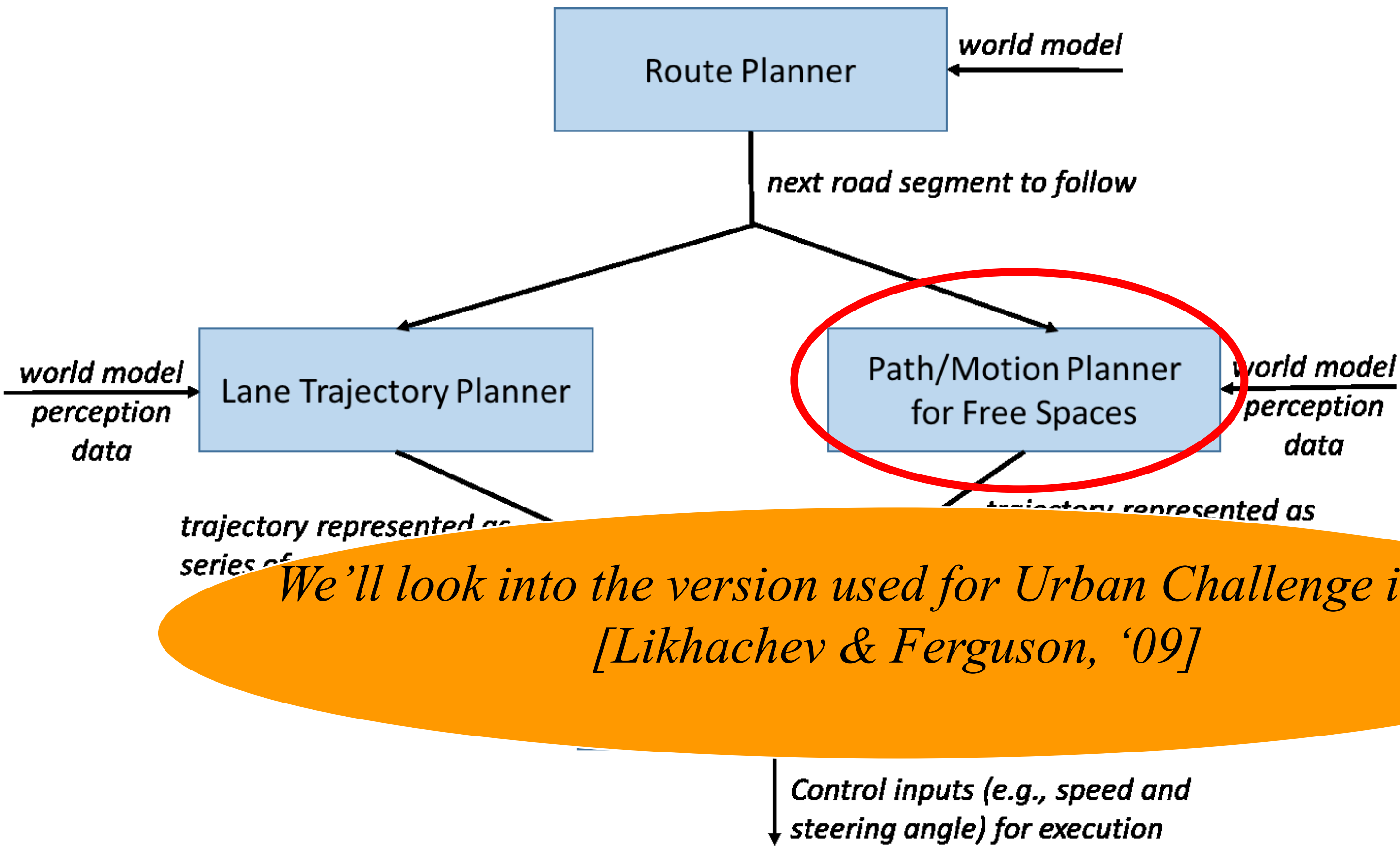
planning states defined by:
discretization along a lane (=x) and perpendicular to it (=y), lane ID, v, time

Low-level Collision Avoidance

world model perception data

Control inputs (e.g., speed and steering angle) for execution

Typical Planning Architecture for Autonomous Vehicle



We'll look into the version used for Urban Challenge in '07
[Likhachev & Ferguson, '09]

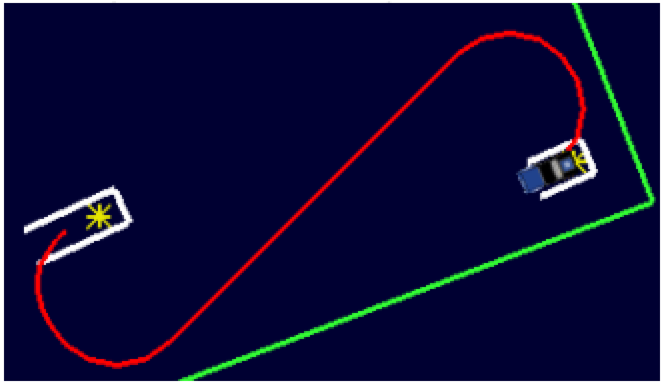
Motivation

- Planning **long complex maneuvers** for the Urban Challenge vehicle from CMU (Tartanracing team)

- Planner suitable for
  - autonomous parking in very large (200m by 200m) cluttered parking lots
  - navigating in off-road conditions
  - navigating cluttered intersections/driveways

Desired Properties

- Generate a path that can be tracked well (at up to 5m/sec):



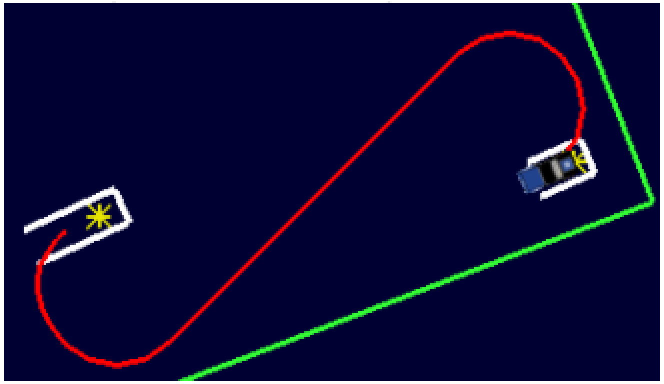
  – path is a 4-dimensional trajectory:

$$(x,\ y,\ \theta,\ v)$$

orientation *speed*

Desired Properties

- Generate a path that can be tracked well (at up to 5m/sec):



  – path is a 4-dimensional trajectory:

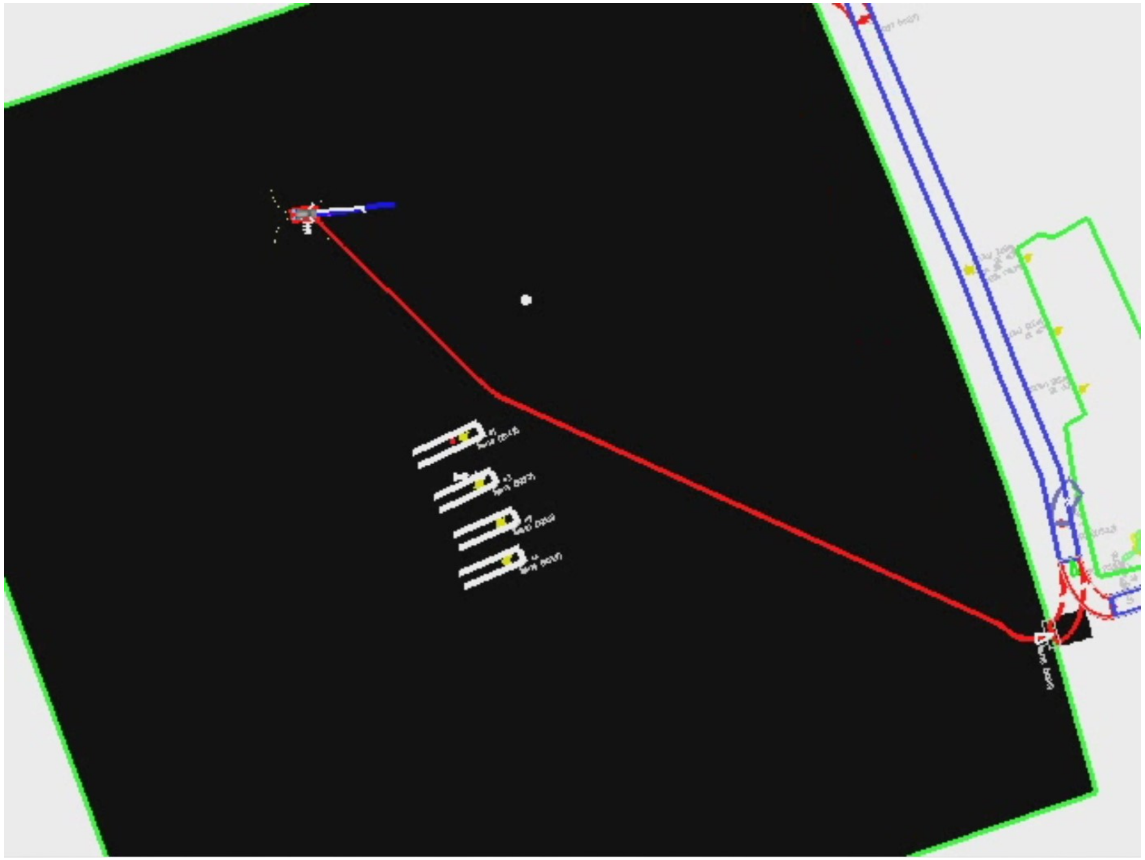
$$(x,\ y,\ \theta,\ v)$$

orientation *speed*

Orientation of the wheels is not included.
When will that be a problem?

Desired Properties

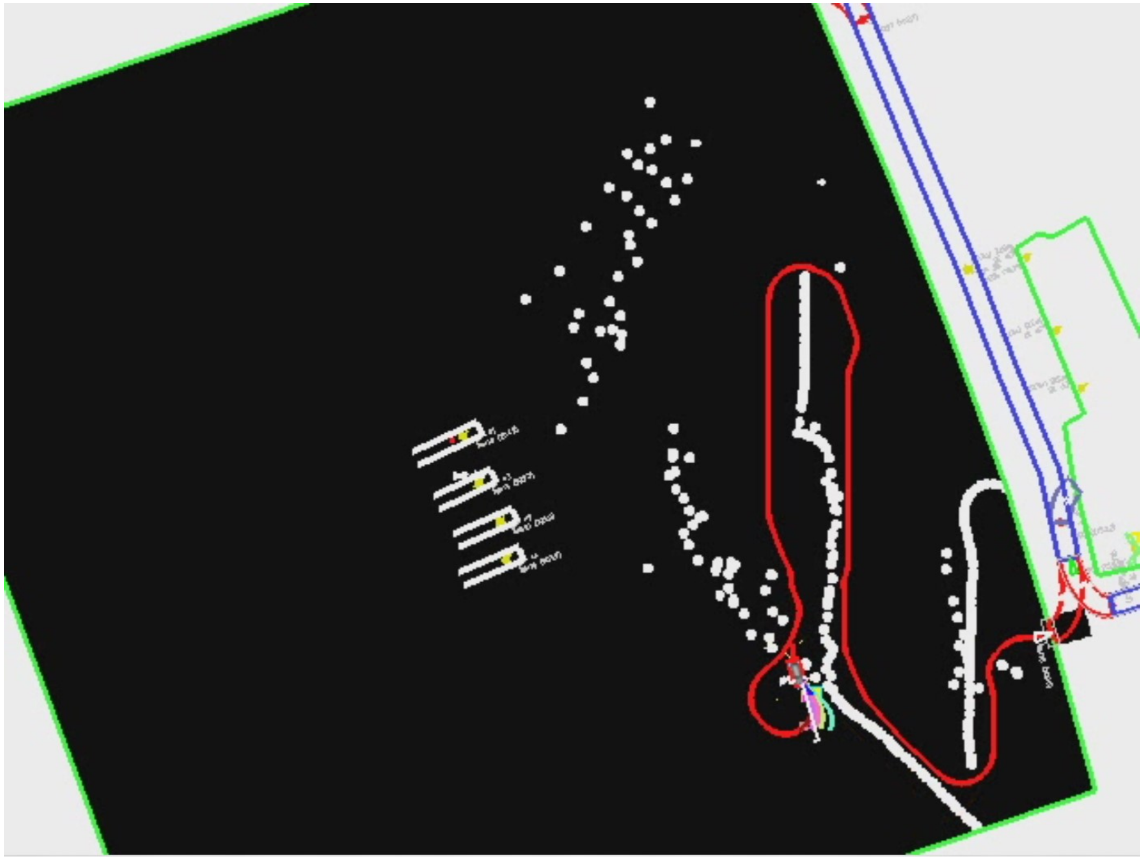
- Fast (2D-like) planning in trivial environments:



200 by 200m parking lot

Desired Properties

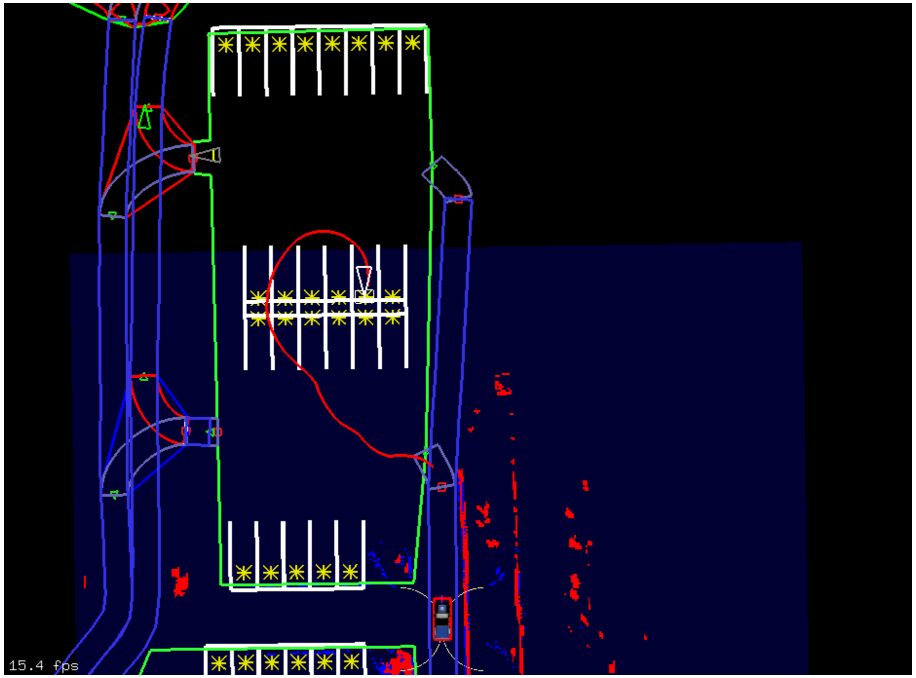
- But can also handle large non-trivial environments:



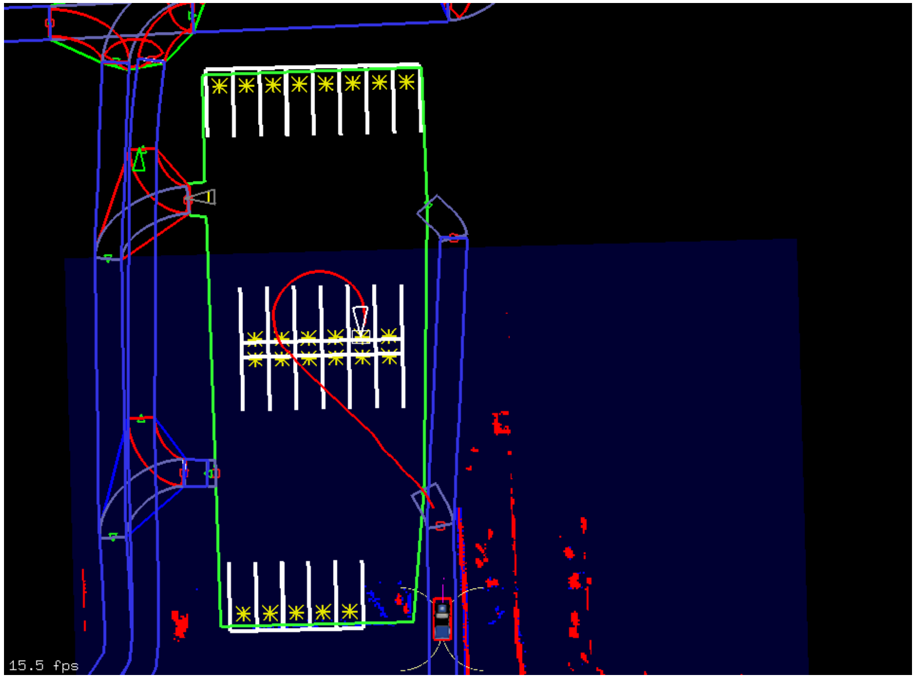
200 by 200m parking lot

Desired Properties

- Anytime property: finds the best path it can within X secs and then improves the path while following it



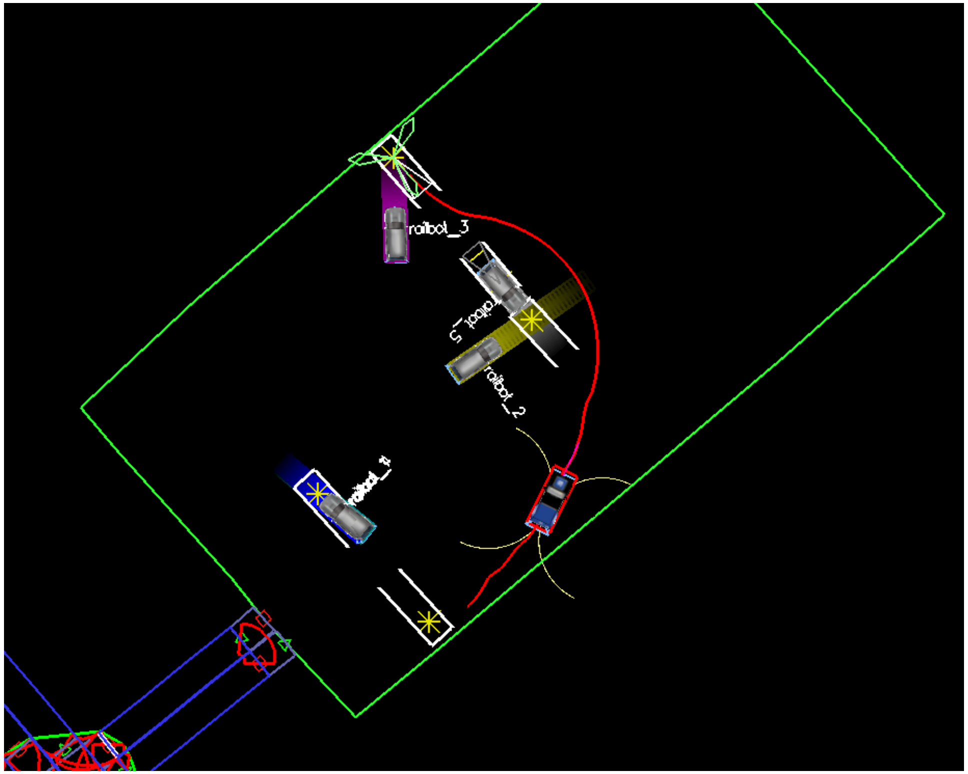
initial path



converged (to optimal) path

Desired Properties

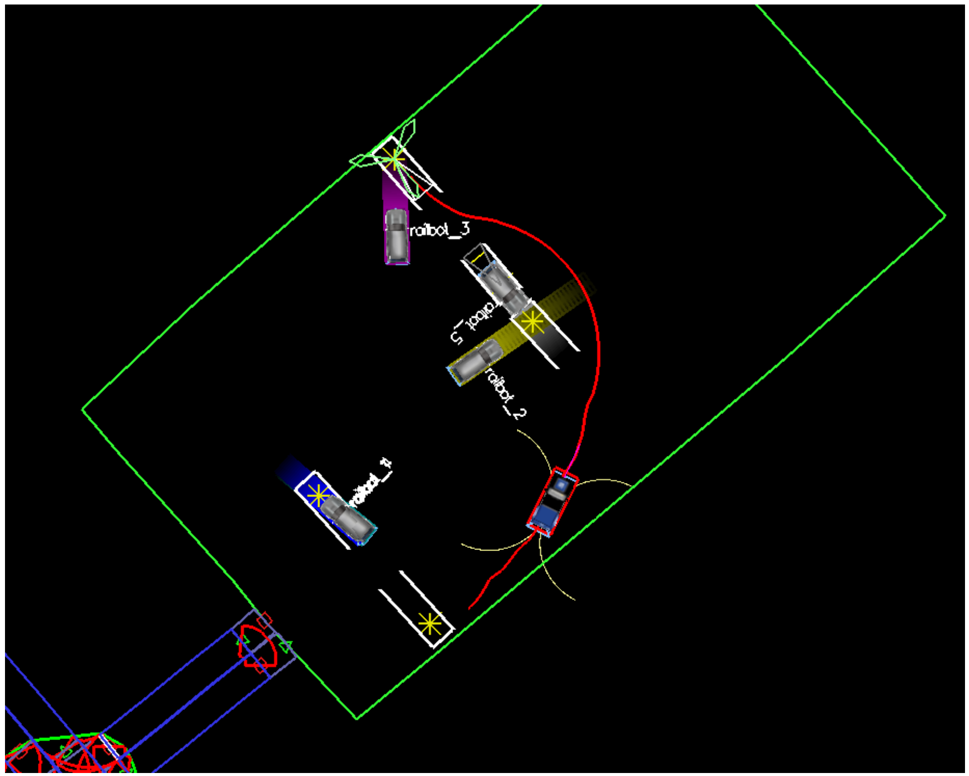
- Fast replanning, especially since we need to avoid other vehicles



planning a path that avoids other vehicles

Desired Properties

- Fast replanning, especially since we need to avoid other vehicles



ther vehicles

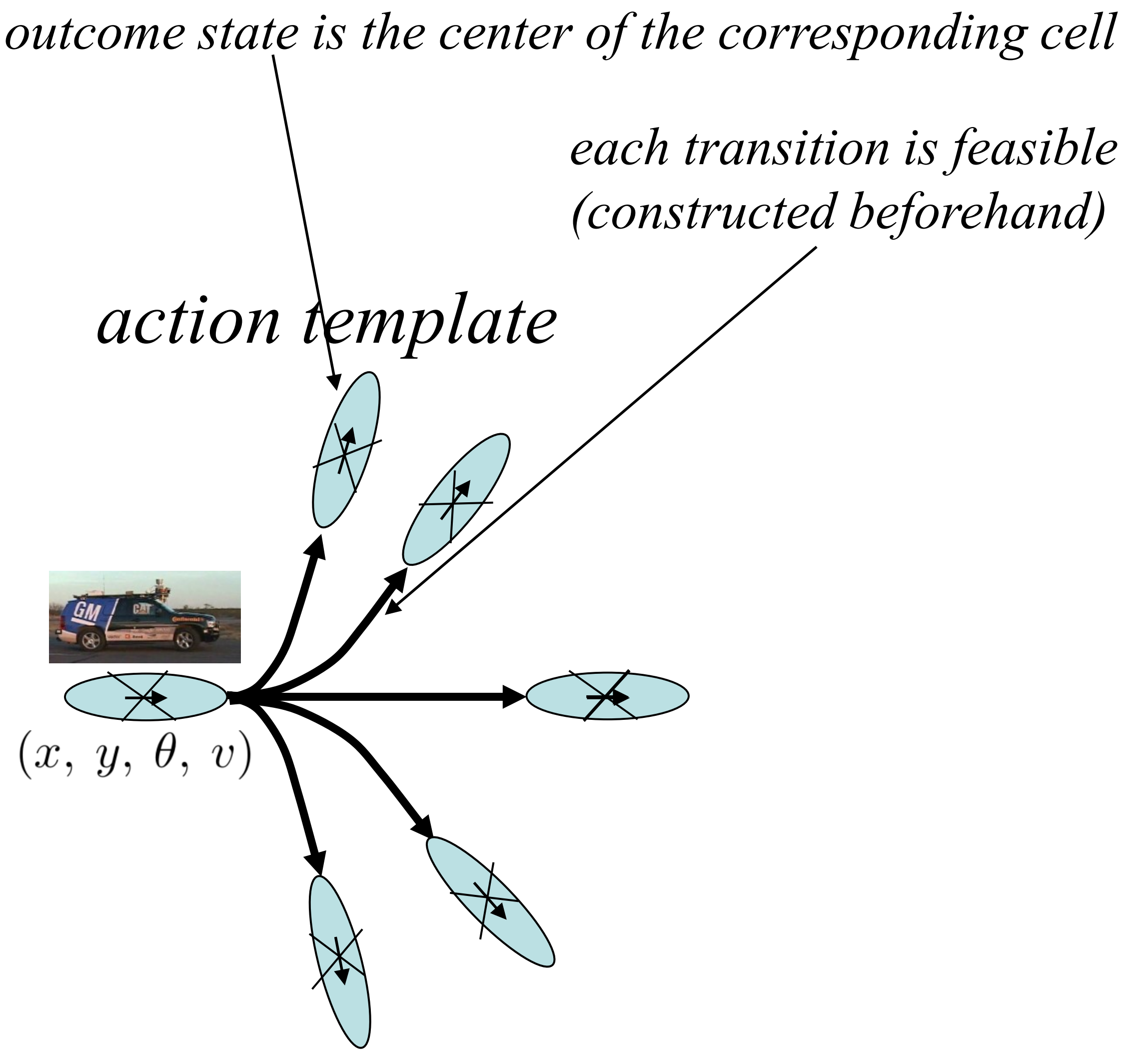
Time is not part of the state-space.
When will that be a problem?

The Approach

- Define an **Implicit** graph
  - multi-resolution version of a lattice graph
- Search the graph for a least-cost path
  - Anytime D* (ARA* + D* Lite)

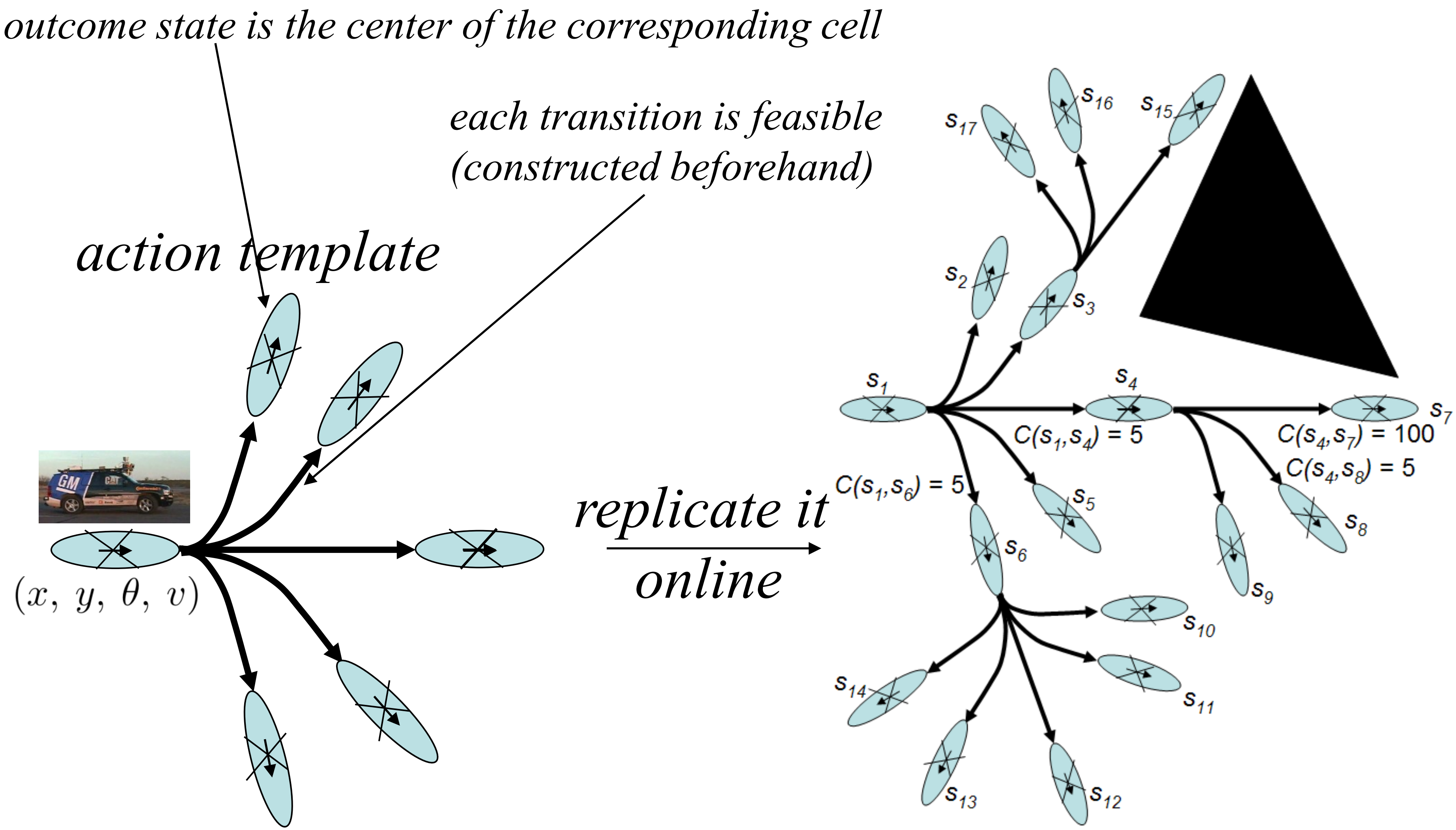
Building the Graph

- Lattice-based graph:



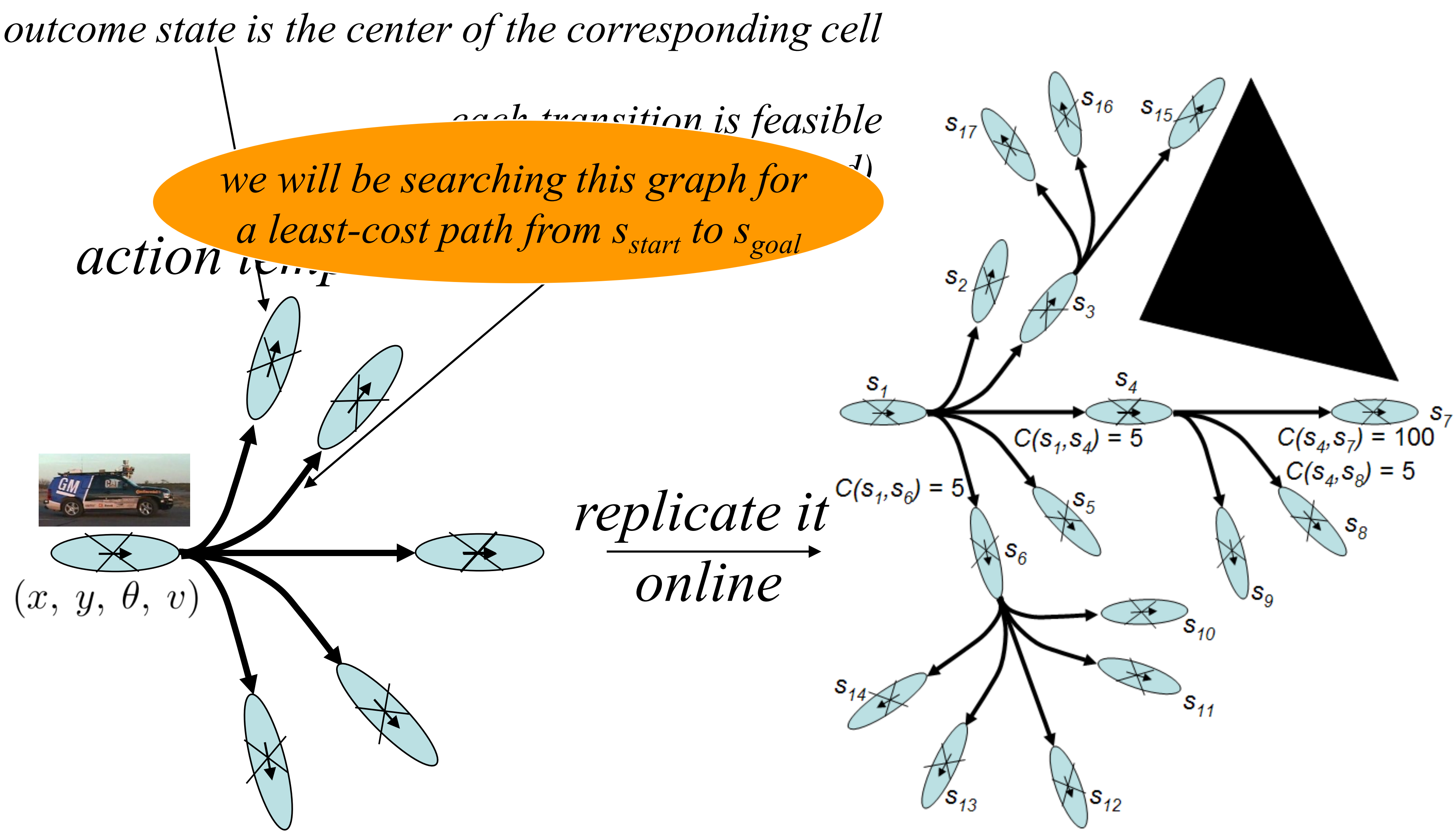
Building the Graph

- Lattice-based graph:



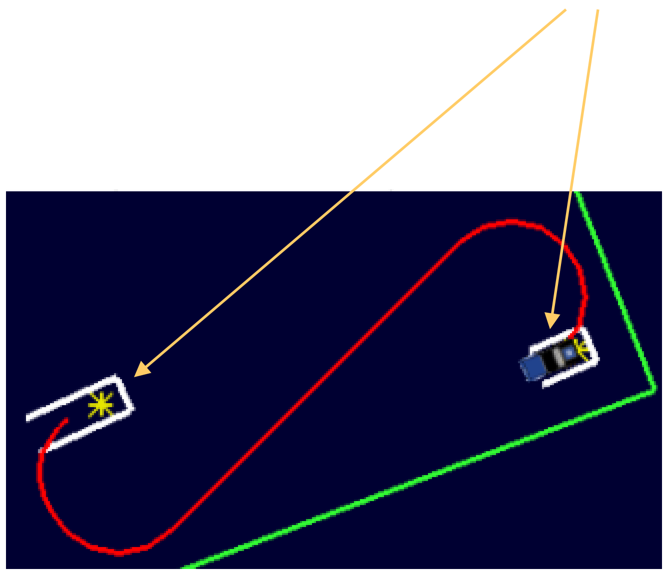
Building the Graph

- Lattice-based graph:



Building the Graph

- Multi-resolution lattice:
  – high density in the most constrained areas (e.g., around start/goal)
  – low density in areas with higher freedom for motions



Building the Graph

- The construction of multi-resolution lattice:
  - the action space of a low-resolution lattice is a strict subset of the action space of the high-resolution lattice

reduces the branching factor for the low-res. lattice

Building the Graph

- The construction of multi-resolution lattice:
  - the action space of a low-resolution lattice is a strict subset of the action space of the high-resolution lattice

reduces the branching factor for the low-res. lattice

  - the state-space of a low-resolution lattice is discretized to be a subset of the possible discretized values of the state variables in the high-resolution lattice

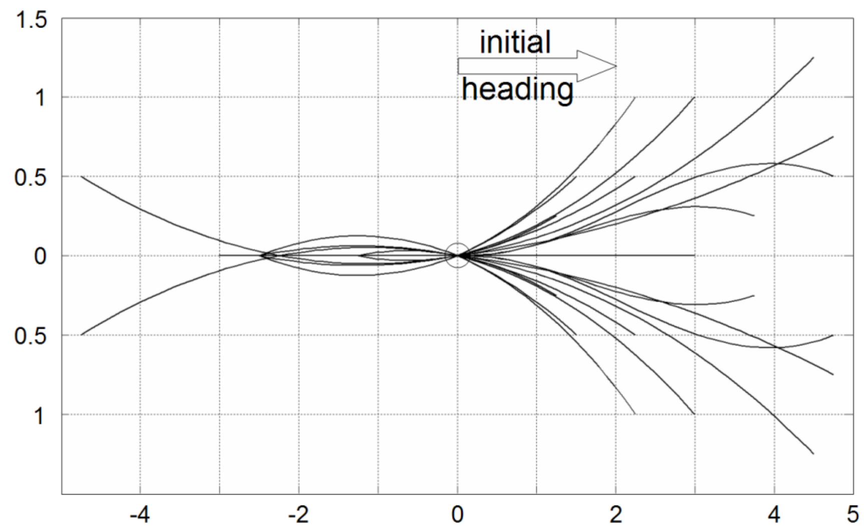
reduces the size of the state-space for the low-res. lattice

both allow for seamless transitions

Building the Graph

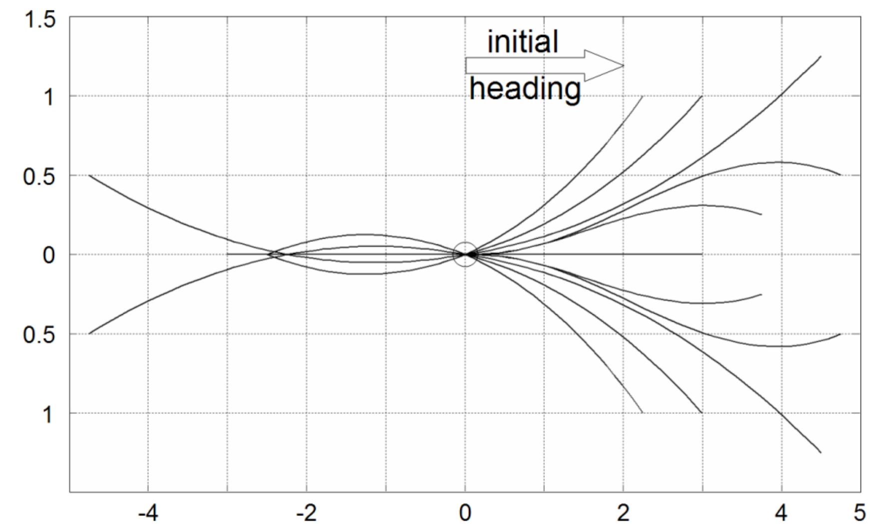
- Multi-resolution lattice used for Urban Challenge:

dense-resolution lattice



36 actions,
32 discrete values of heading
0.25m discretization for x,y

low-resolution lattice



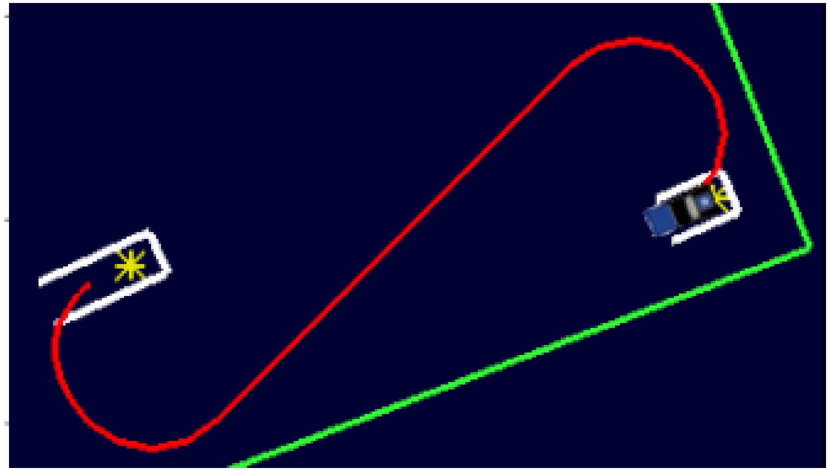
24 actions,
16 discrete values of heading
0.25m discretization for x,y

Building the Graph

- Properties of multi-resolution lattice:
  - ***utilization of low-resolution lattice:*** *every path that uses only the action space of the low-resolution lattice is guaranteed to be a valid path in the multi-resolution lattice*
  - ***validity of paths:*** *every path in the multi-resolution lattice is guaranteed to be a valid path in a lattice that uses only the action space of the high-resolution lattice*

Building the Graph

- Benefit of the multi-resolution lattice:



Lattice	States Expanded	Planning Time (s)
High-resolution	2,933	0.19
Multi-resolution	1,228	0.06

Searching the Graph

- Anytime D*:
  - anytime incremental version of A*
  - **anytime:** computes the best path it can within provided time and improves it while the robot starts execution.
  - **incremental:** it reuses its previous planning efforts and as a result, re-computes a solution much faster

Searching the Graph

- Anytime D*:

desired bound on the

computes a path reusing all of the previous search efforts

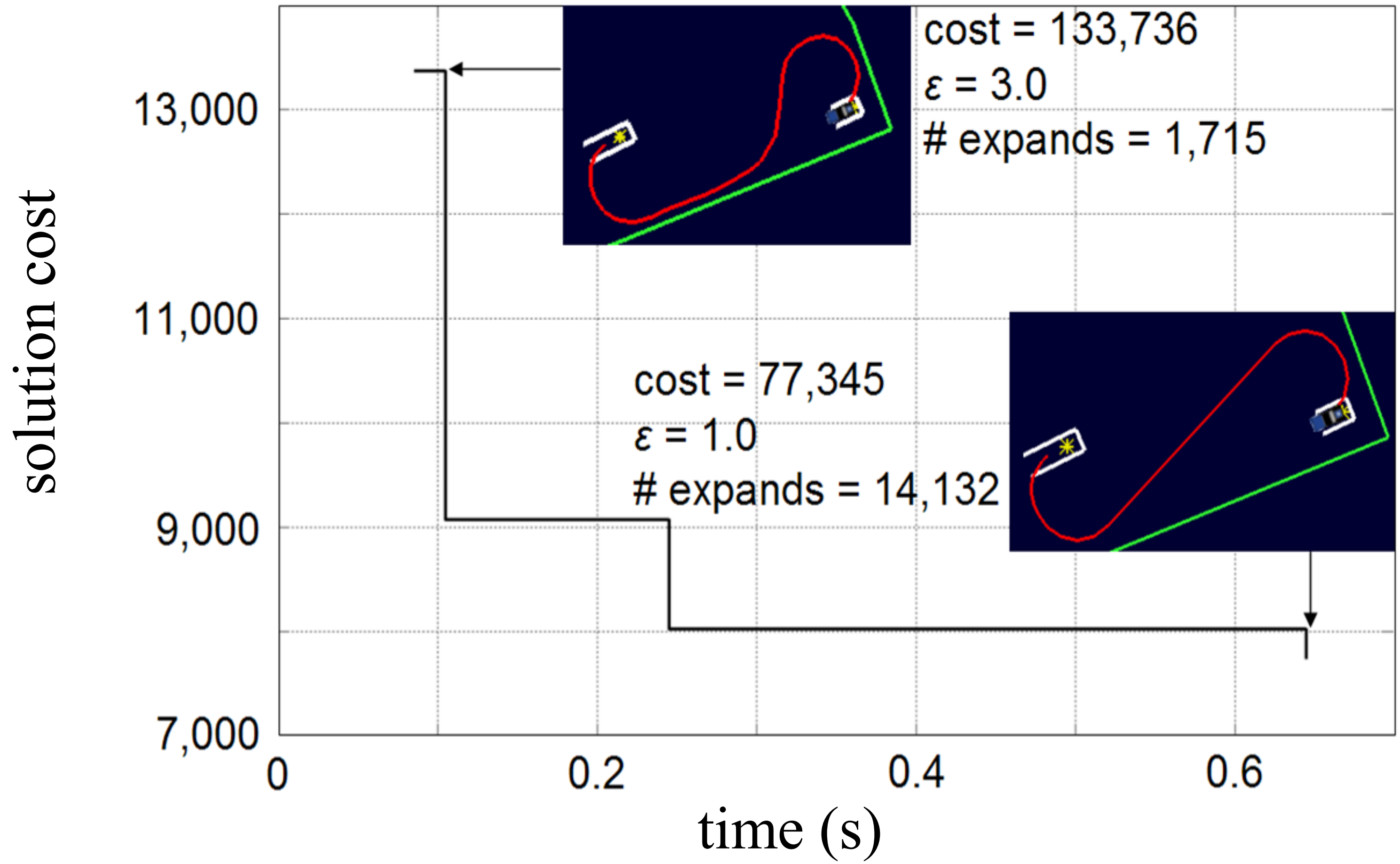
guarantees that cost(path) $\leq \varepsilon$ cost(optimal path)

set ε to large value;
until goal is reached
    ComputePathwithReuse();
    publish ε-suboptimal path for execution;
    update the map based on new sensory information;
    update current state of the agent;
    if significant changes were observed
        increase ε or replan from scratch;
    else
        decrease ε;

makes it improve the solution

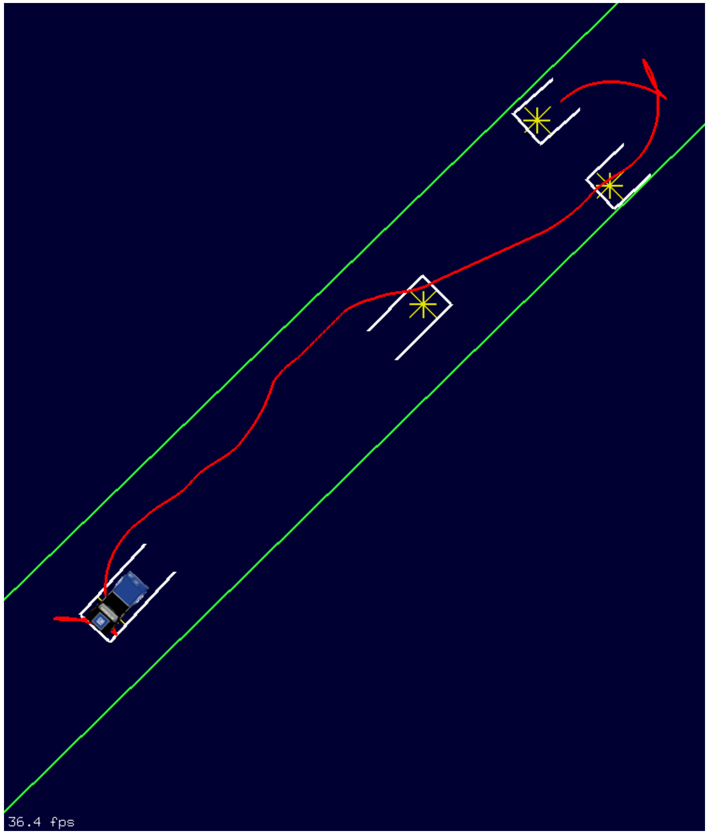
Searching the Graph

- Anytime behavior of Anytime D*:

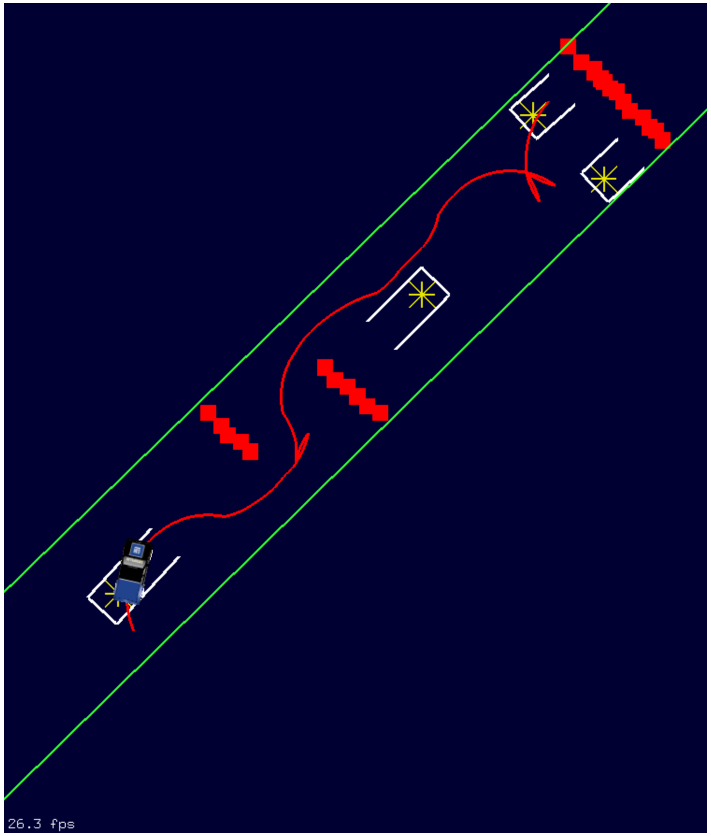


Searching the Graph

- Incremental behavior of Anytime D*:



initial path

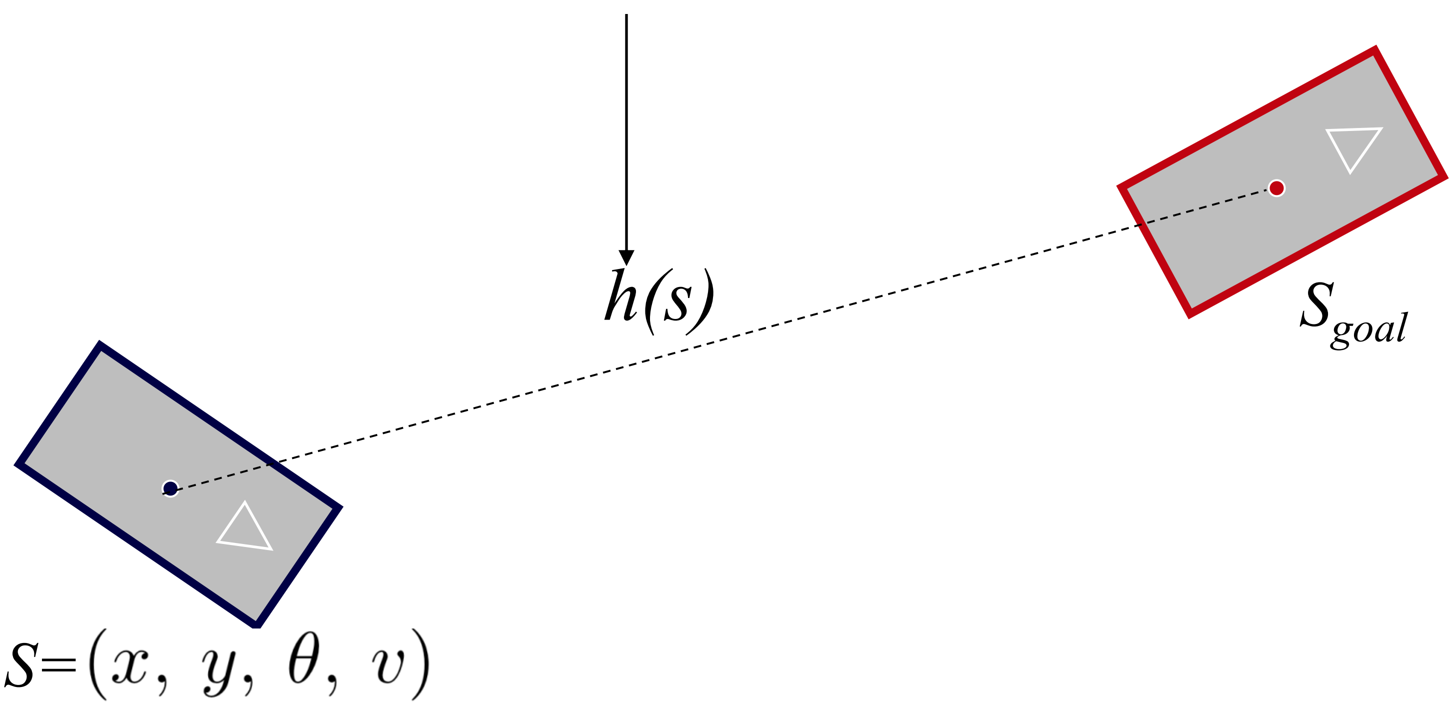


a path after re-planning

Searching the Graph

- Performance of Anytime D* depends strongly on heuristics *h(s)*: estimates of cost-to-goal

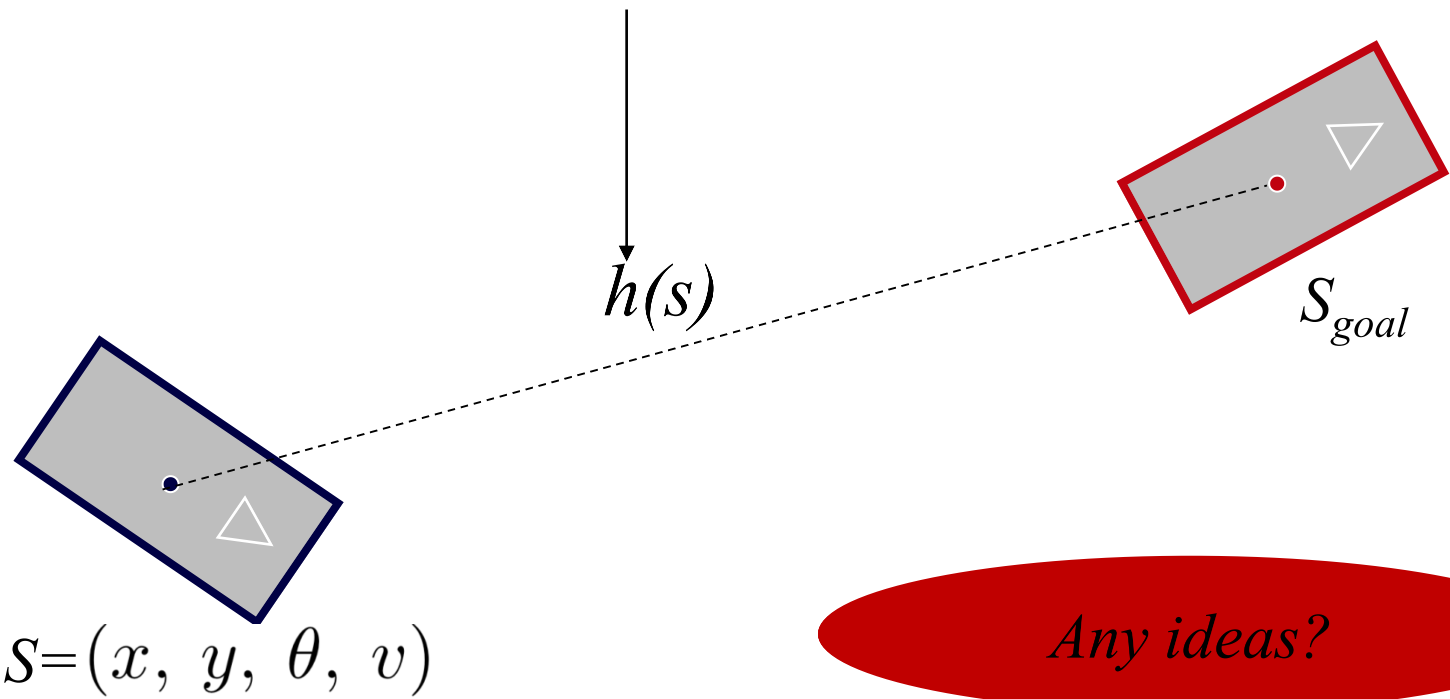
should be consistent and admissible (never overestimate cost-to-goal)



Searching the Graph

- Performance of Anytime D* depends strongly on heuristics *h(s)*: estimates of cost-to-goal

should be consistent and admissible (never overestimate cost-to-goal)

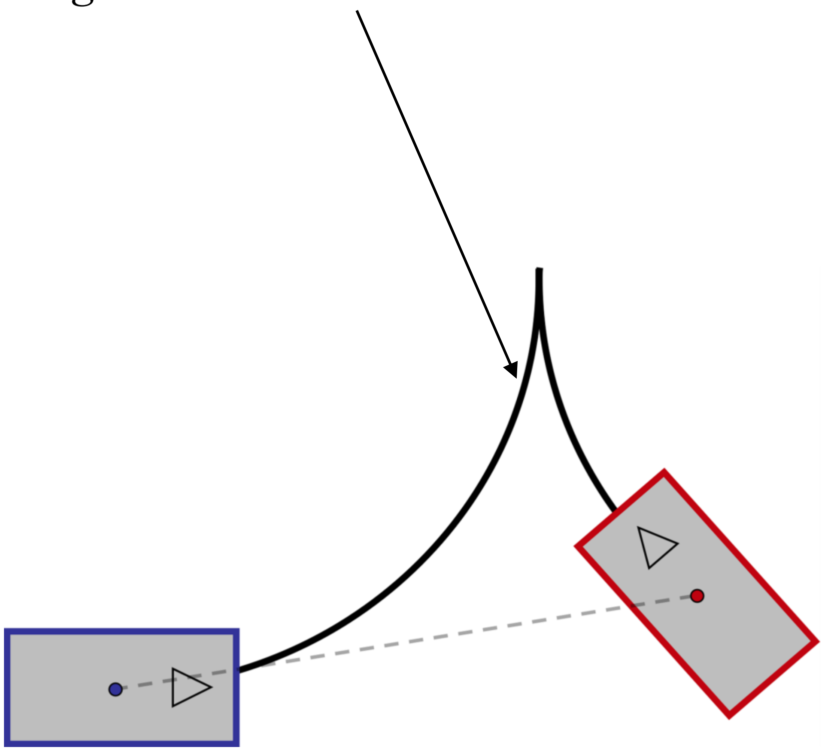


Any ideas?

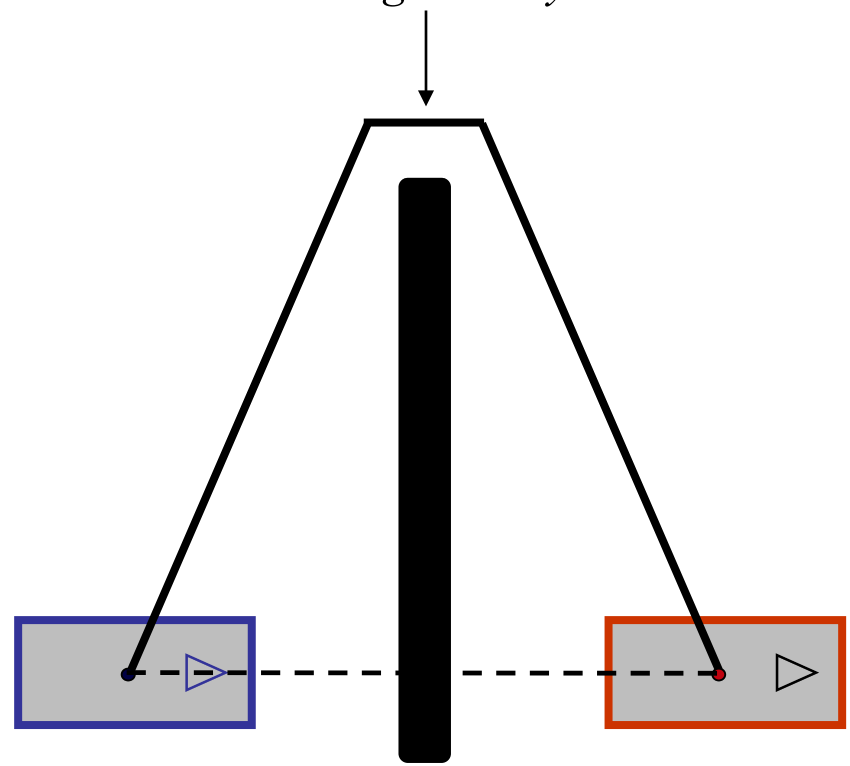
Searching the Graph

- In our planner: $h(s) = max(h_{mech}(s), h_{env}(s))$, where
  - $h_{mech}(s)$ – mechanism-constrained heuristic
  - $h_{env}(s)$ – environment-constrained heuristic

$h_{mech}(s)$ – considers only dynamics constraints and ignores environment



$h_{env}(s)$ – considers only environment constraints and ignores dynamics



Searching the Graph

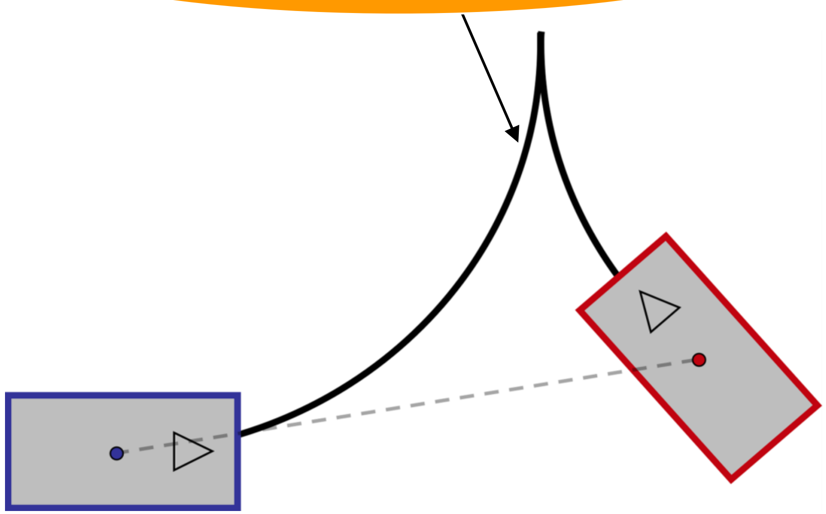
- In our planner: $h(s) = max(h_{mech}(s), h_{env}(s))$, where
  - $h_{mech}(s)$ – mechanism-constrained heuristic
  - $h_{env}(s)$ – environment-constrained heuristic

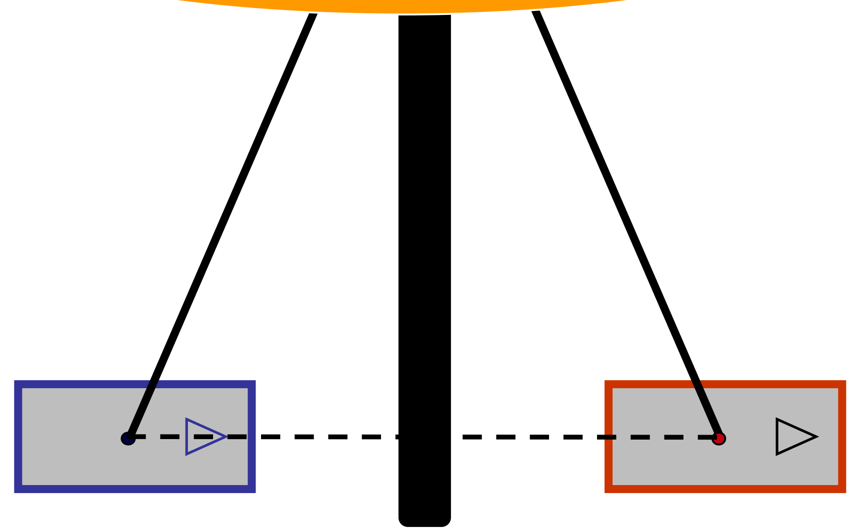
$h_{mech}(s)$ – considers only dynamics constraints and ignores environment

$h_{env}(s)$ – considers only environment constraints and ignores dynamics

pre-computed as a table lookup for high-res. lattice

computed online by running a 2D A with late termination*



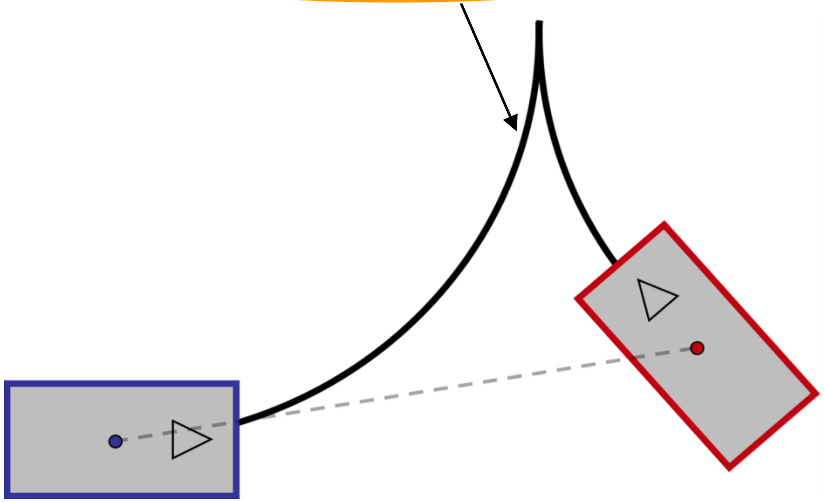


Searching the Graph

- In our planner: $h(s) = max(h_{mech}(s), h_{env}(s))$, where
  - $h_{mech}(s)$ – mechanism-constrained heuristic
  - $h_{env}(s)$ – environment-constrained heuristic

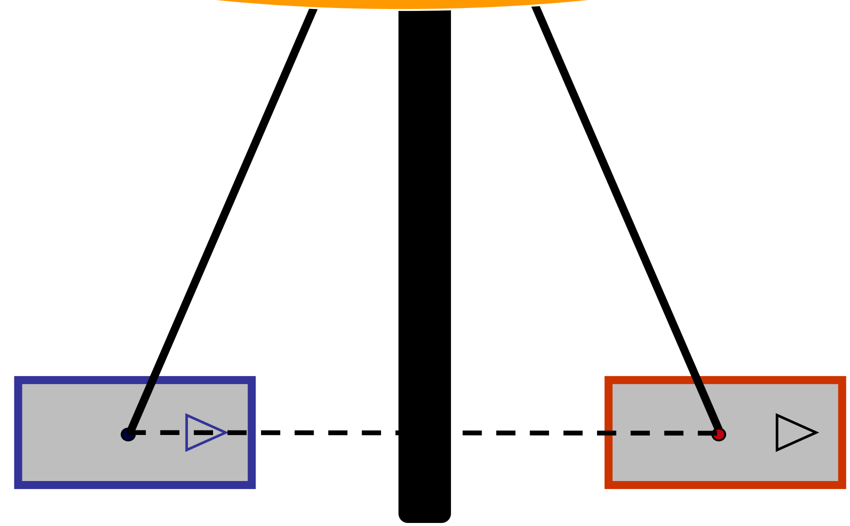
$h_{mech}(s)$ – considers only dynamics constraints and ignores environment

pre-computed as a table lookup for high-res. lattice



$h_{env}(s)$ – considers only environment constraints and ignores dynamics

computed online by running a 2D A with late termination*



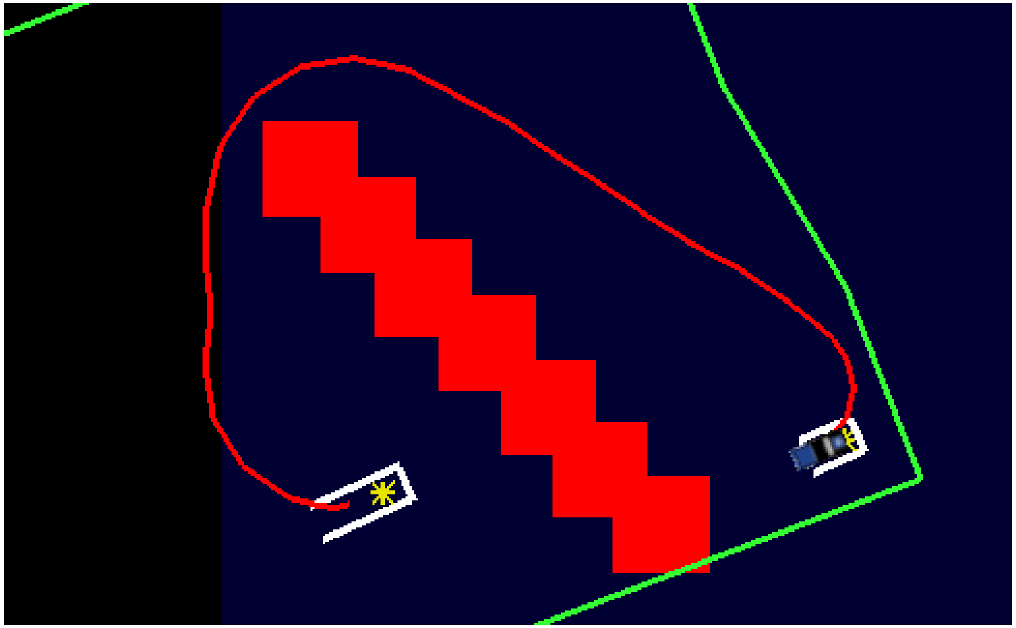
Searching the Graph

- In our planner: $h(s) = max(h_{mech}(s), h_{env}(s))$
- $h_{mech}(s)$ – admissible and consistent
- $h_{env}(s)$ – admissible and consistent
- $h(s)$ – admissible and consistent

Theorem. *The cost of a path returned by Anytime D* is no more than ε times the cost of a least-cost path from the vehicle configuration to the goal configuration using actions in the multi-resolution lattice, where ε is the current value by which Anytime D* inflates heuristics.*

Searching the Graph

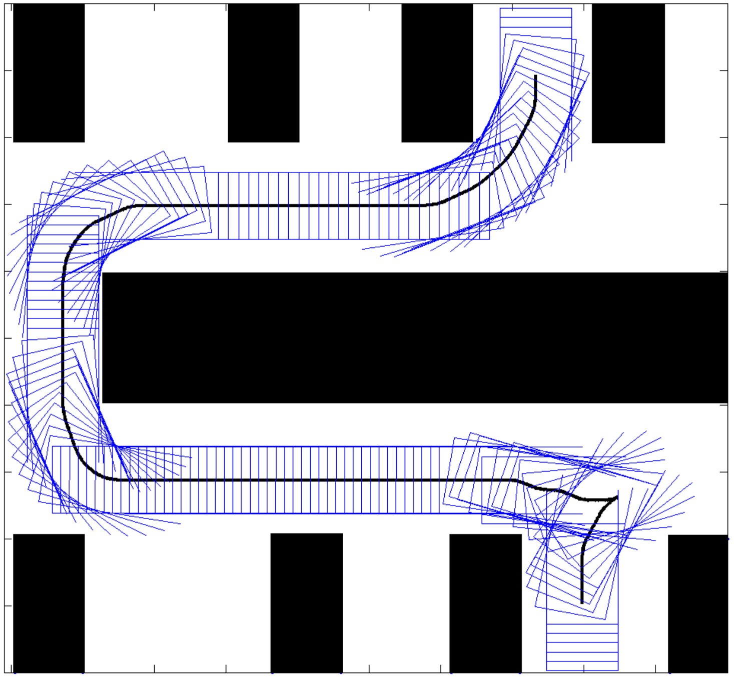
- Benefit of the combined heuristics:



Heuristic	States Expanded	Planning Time (s)
Environment-constrained only	26,108	1.30
Mechanism-constrained only	124,794	3.49
Combined	2,019	0.06

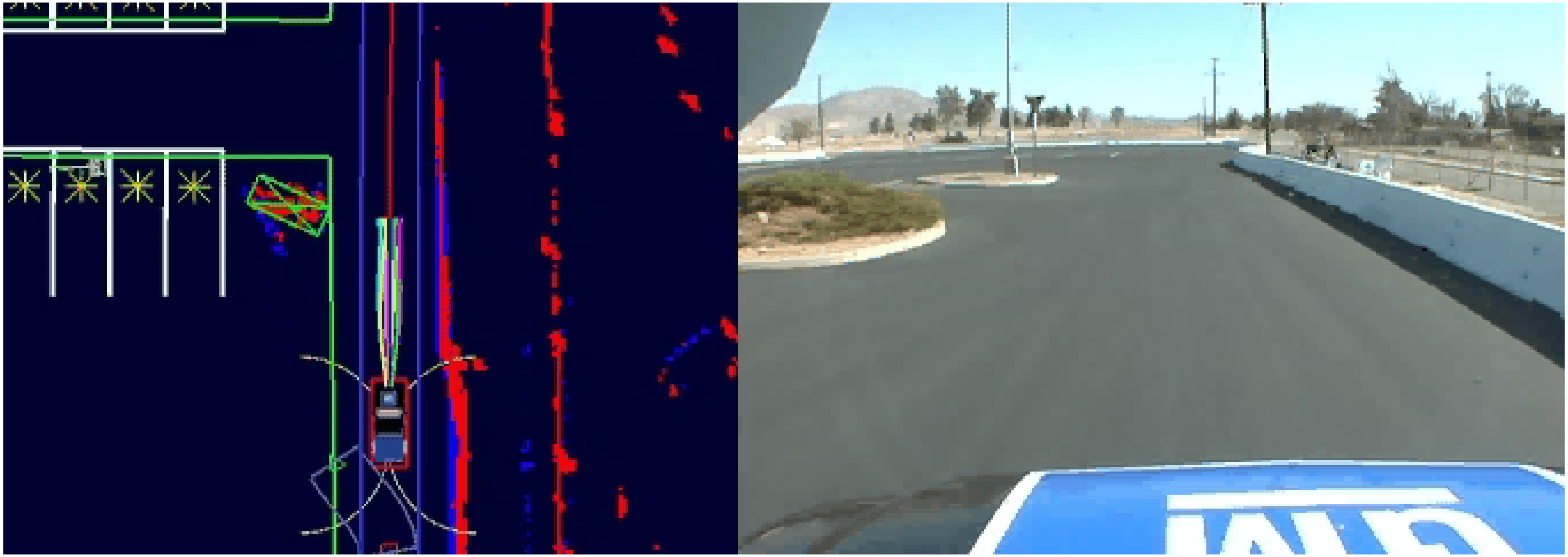
Optimizations

- Pre-compute as much as possible
  - convolution cells for each action for each initial heading



Results

- Plan improvement



Tartanracing, CMU

Results

- Replanning in a large parking lot (200 by 200m)

Tartanracing, CMU

What You Should Know…

- Typical hierarch of planners used in self-driving
- Multi-resolution lattice
- Benefits of anytime and incremental planning
- Ways to generate informative heuristics for motion planning for self-driving