

***16-782***

***Planning & Decision-making in Robotics***

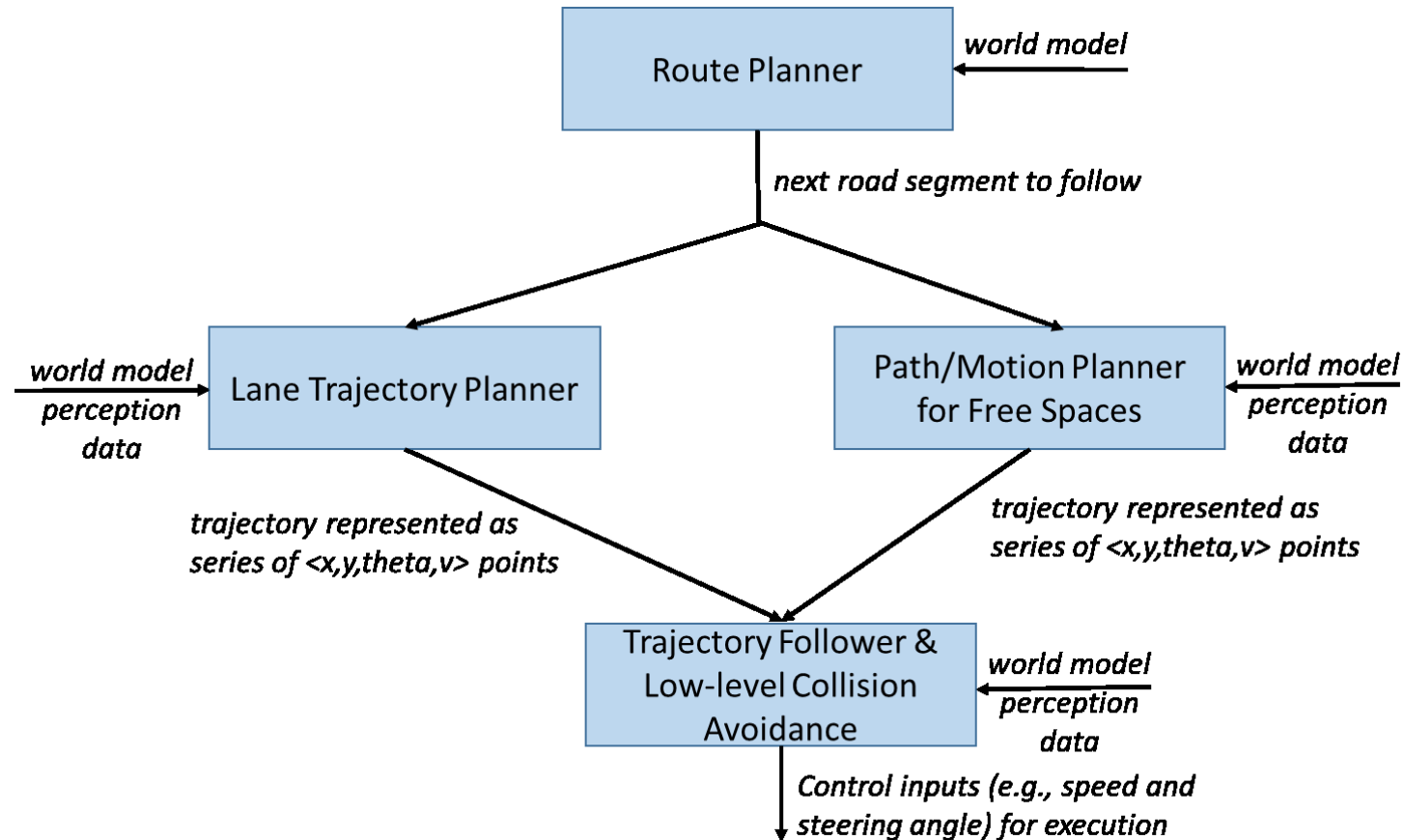
***Case Study:  
Planning for Autonomous Driving***

*Maxim Likhachev*

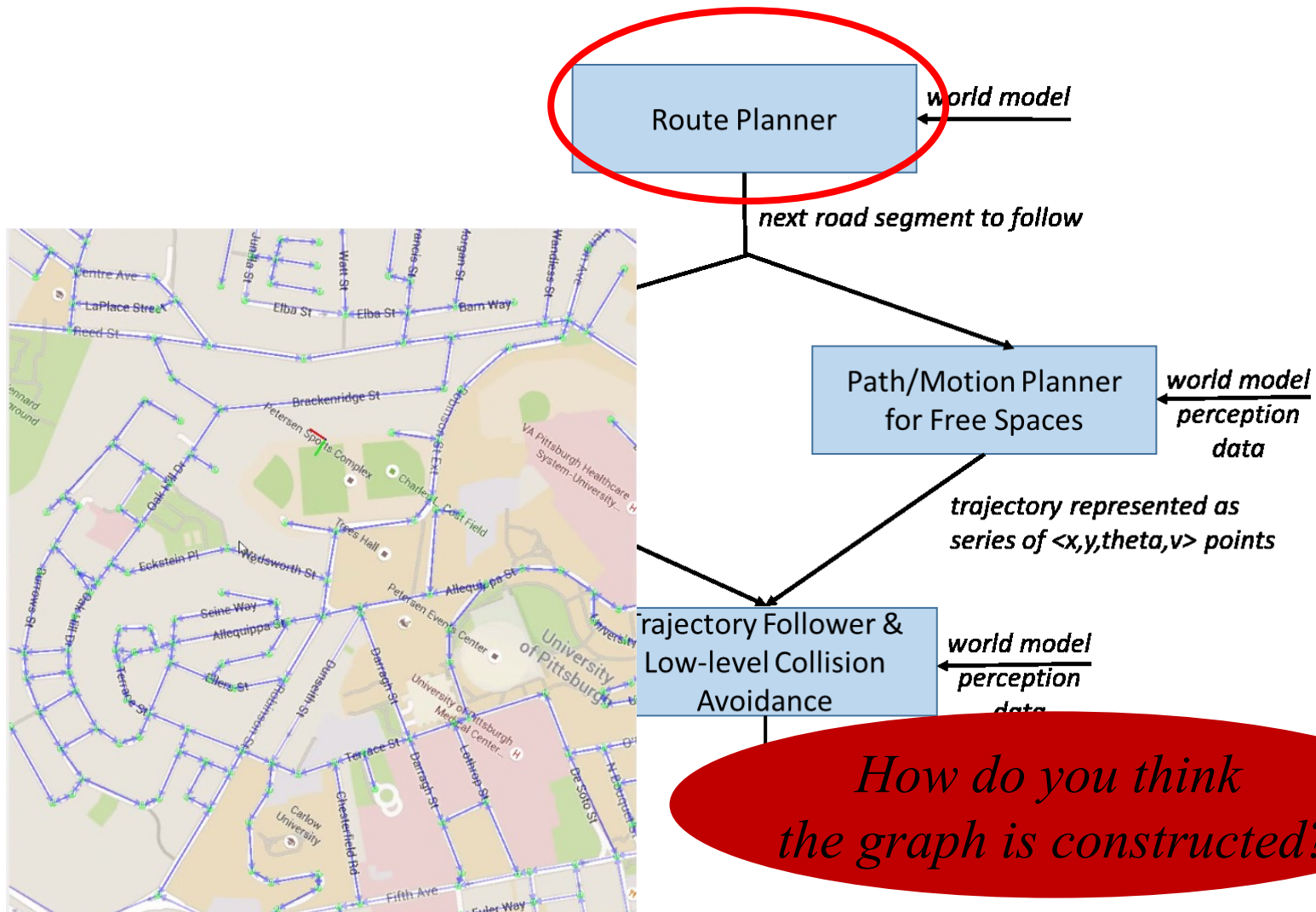
*Robotics Institute*

*Carnegie Mellon University*

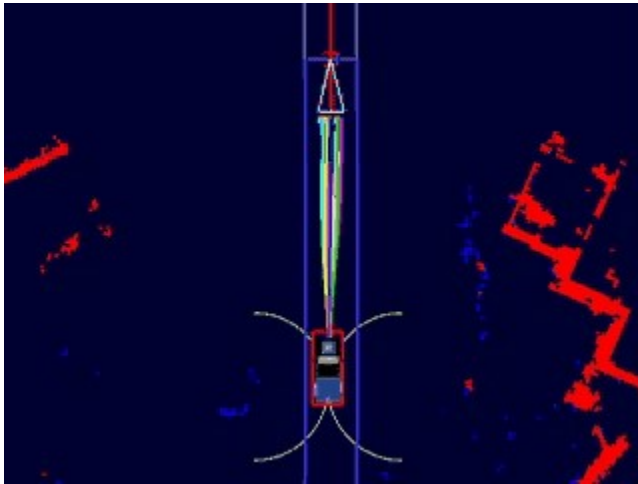
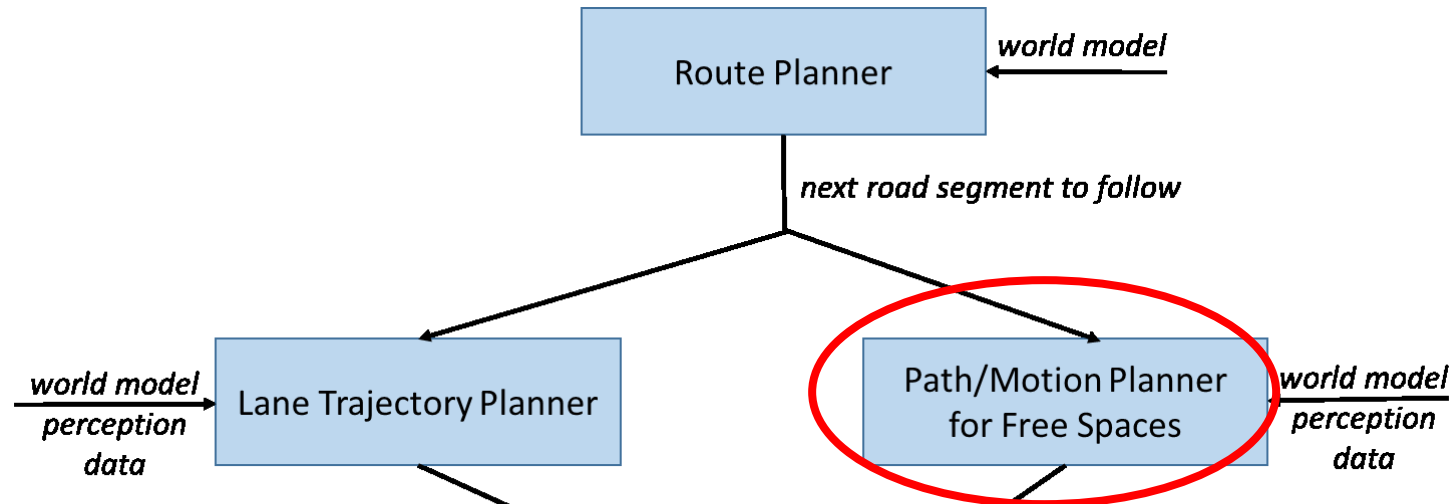
# Typical Planning Architecture for Autonomous Vehicle



# Typical Planning Architecture for Autonomous Vehicle



# Typical Planning Architecture for Autonomous Vehicle

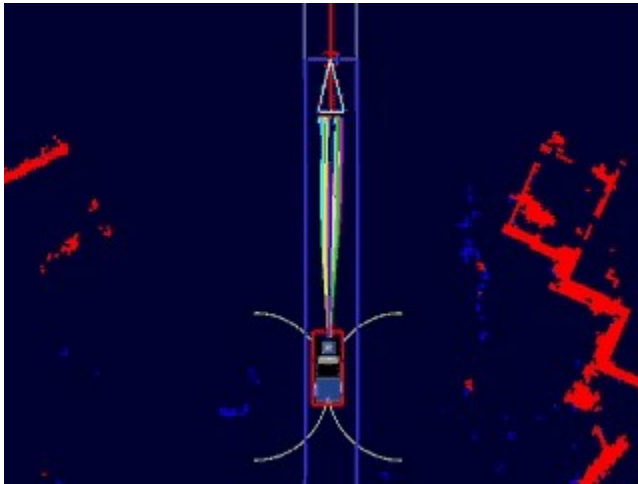
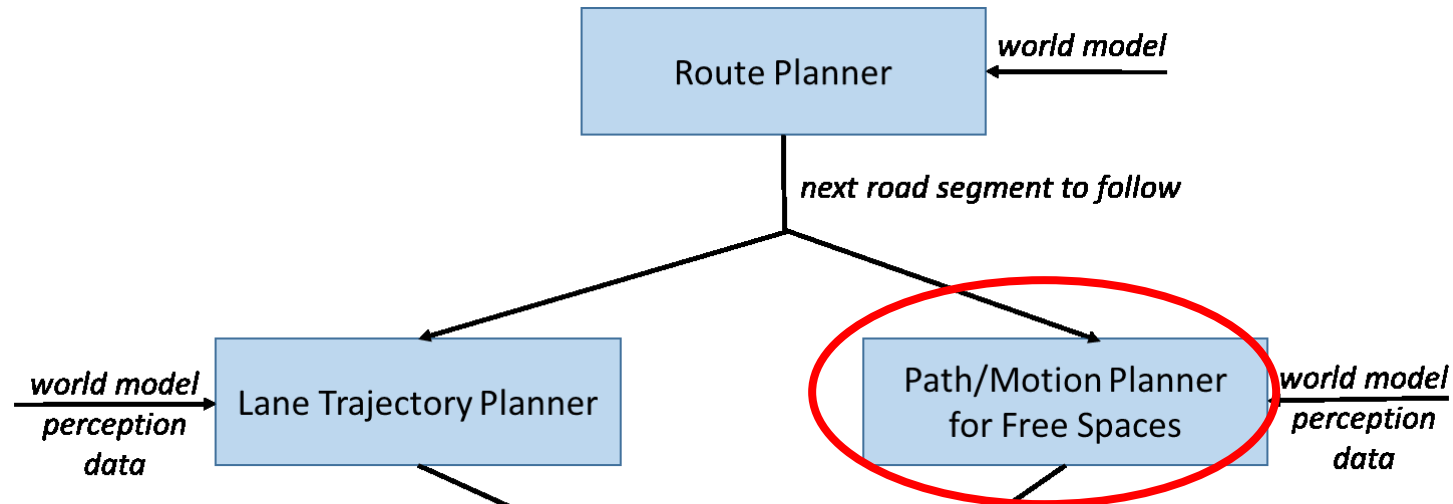


represented as  
 $y, \theta, v$  points

and  
on

*Tartanracing, CMU*

# Typical Planning Architecture for Autonomous Vehicle



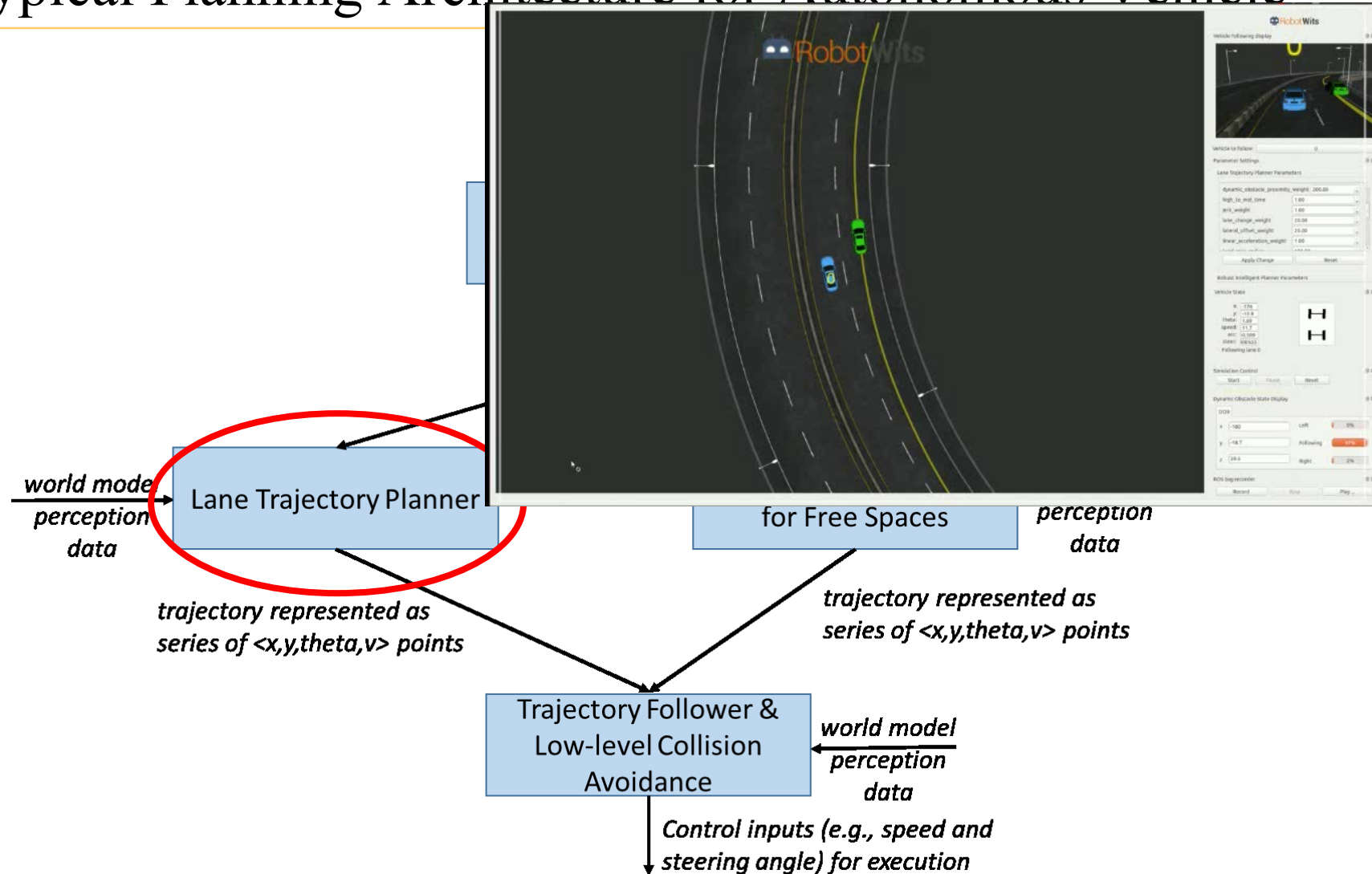
represented as  
 $y, \theta, v$  points

l  
-  
and  
on

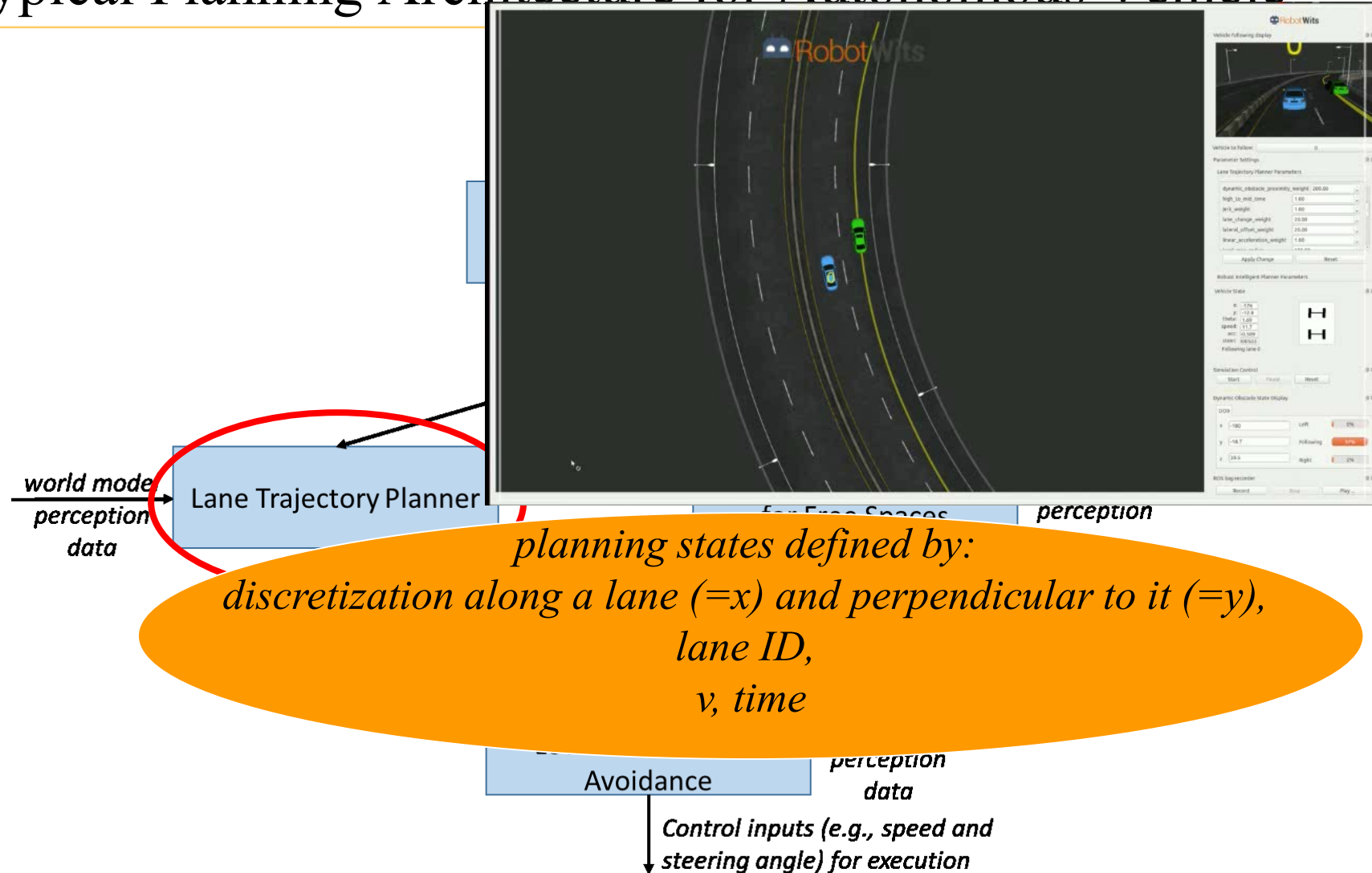
planning states defined by:  
 $x, y, \theta, v$

Tartanracing, CMU

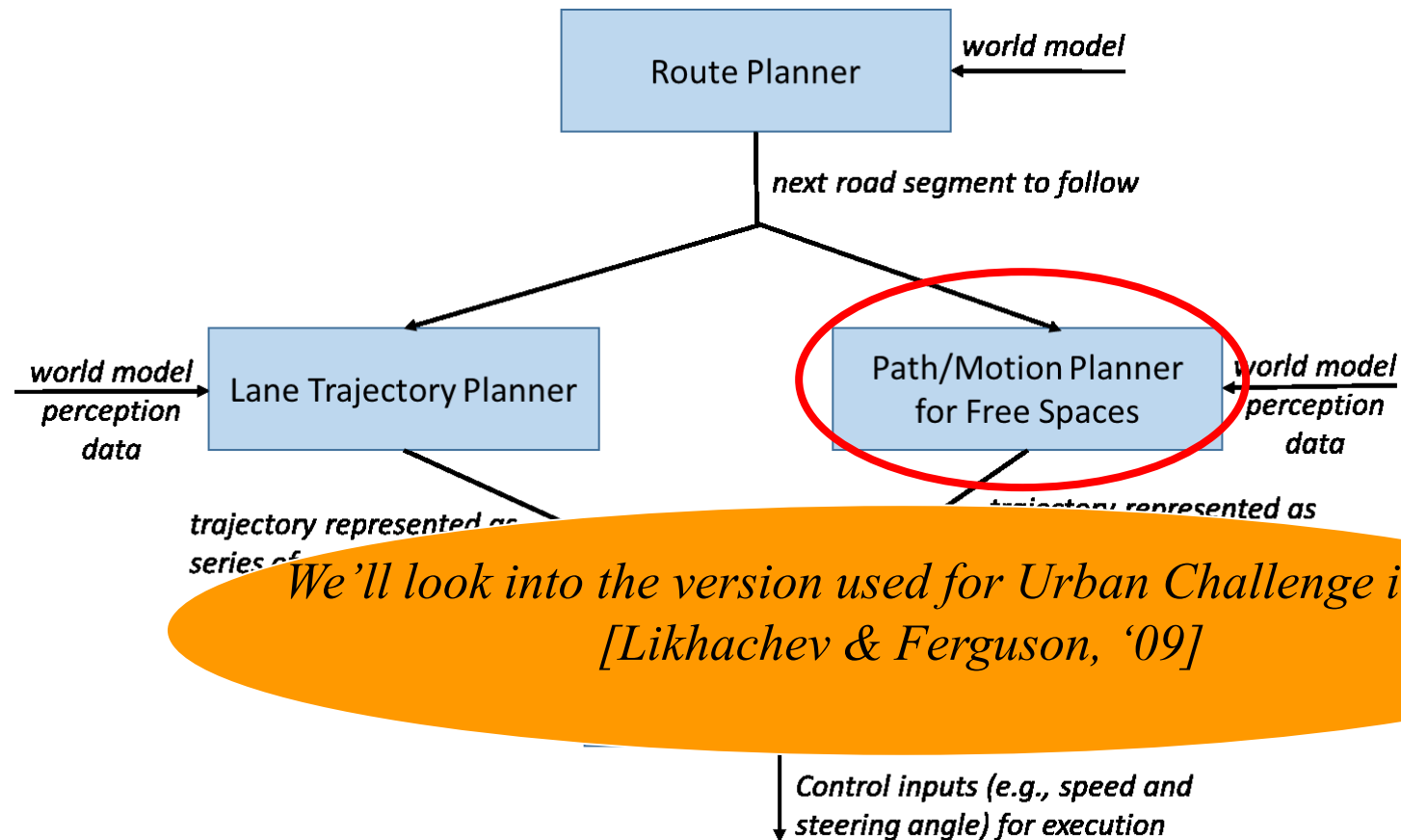
# Typical Planning Architecture for Autonomous Vehicle



# Typical Planning Architecture for Autonomous Vehicle



# Typical Planning Architecture for Autonomous Vehicle





# Motivation

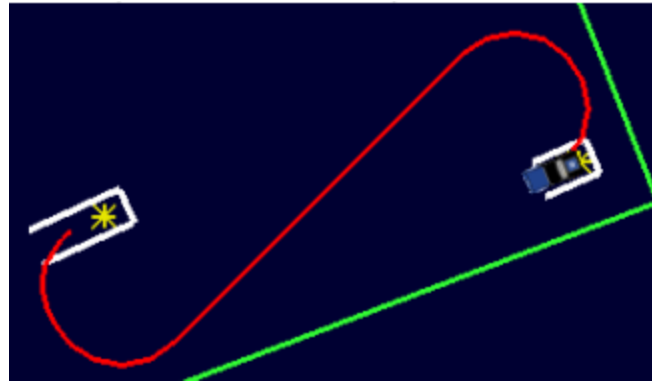
- Planning **long complex maneuvers** for the Urban Challenge vehicle from CMU (Tartanracing team)



- 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):



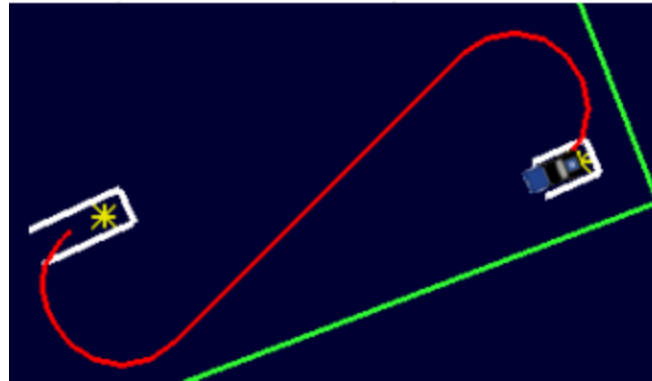
- path is a 4-dimensional trajectory:

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

*orientation* *speed*

# Desired Properties

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



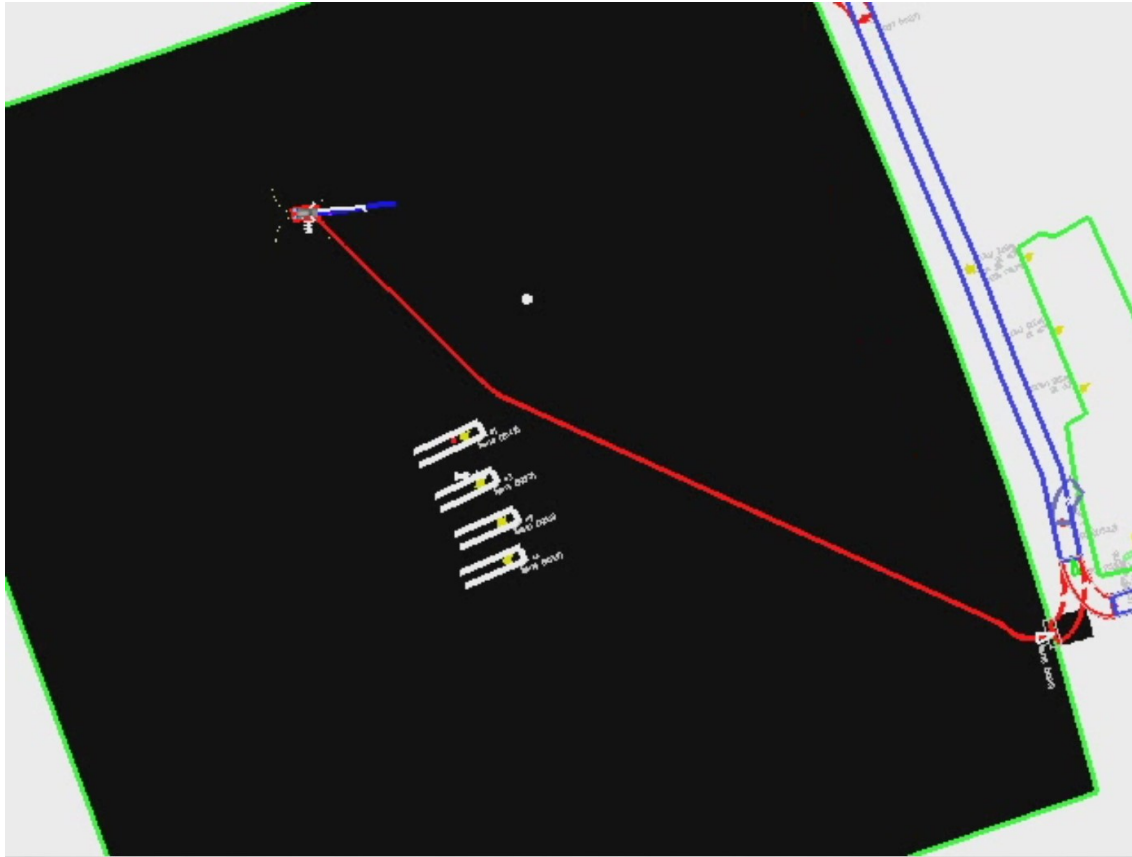
- path is a 4-dimensional trajectory:

$(x, y, \theta, v)$   
                    ↑                    ↑  
                  orientation      speed

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

# Desired Properties

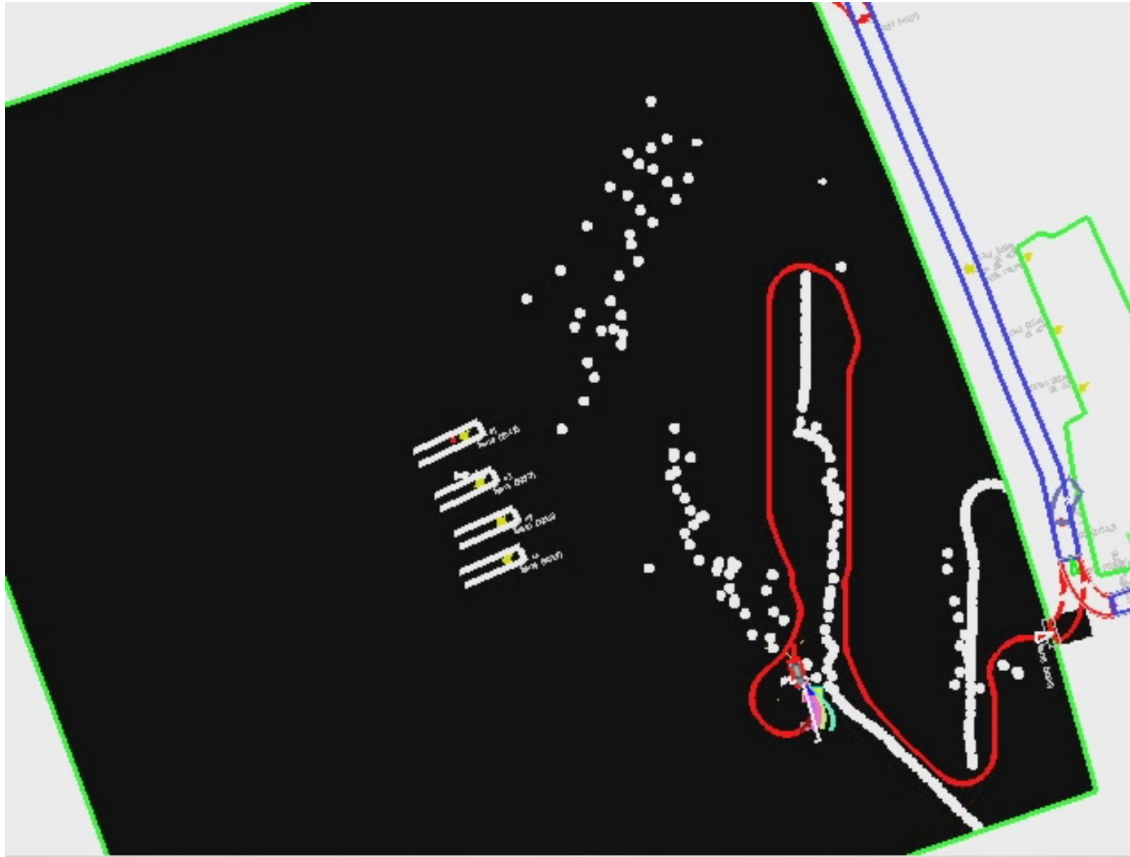
- Fast (2D-like) planning in trivial environments:



*200 by 200m parking lot*

# Desired Properties

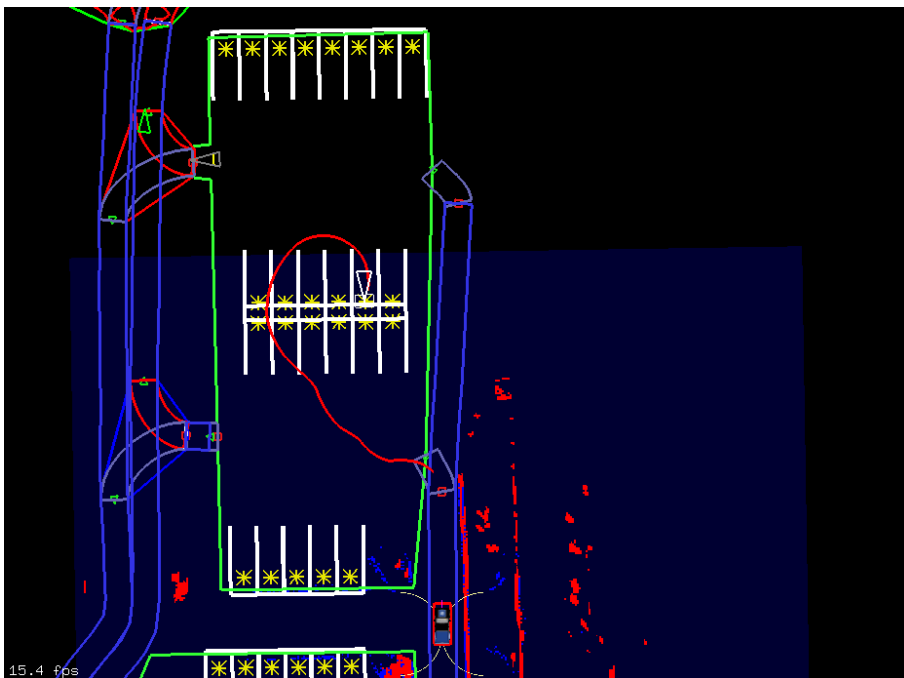
- But can also handle large non-trivial environments:



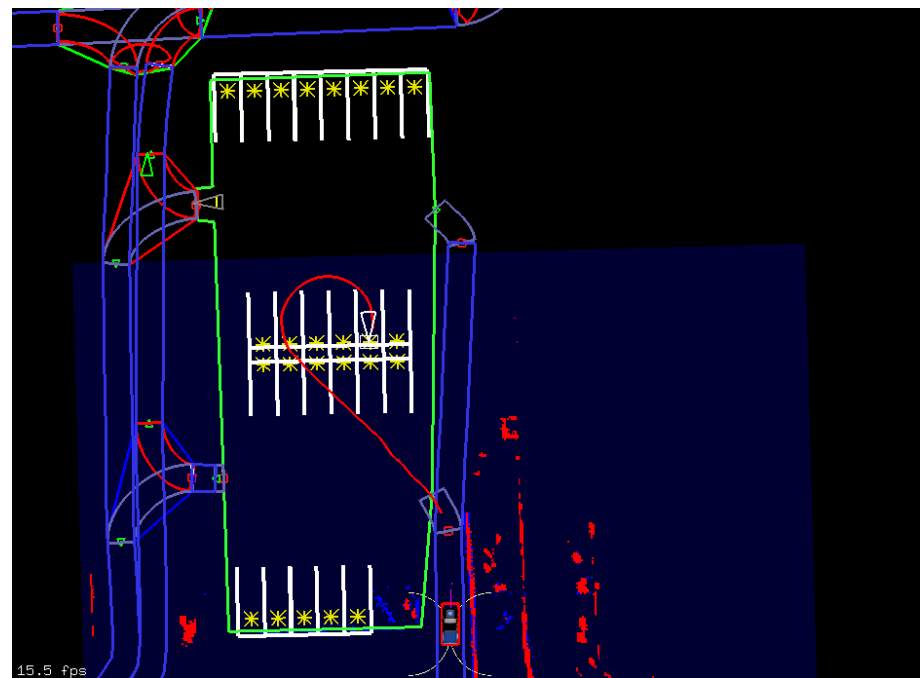
*200 by 200m parking lot*

# Desired Properties

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



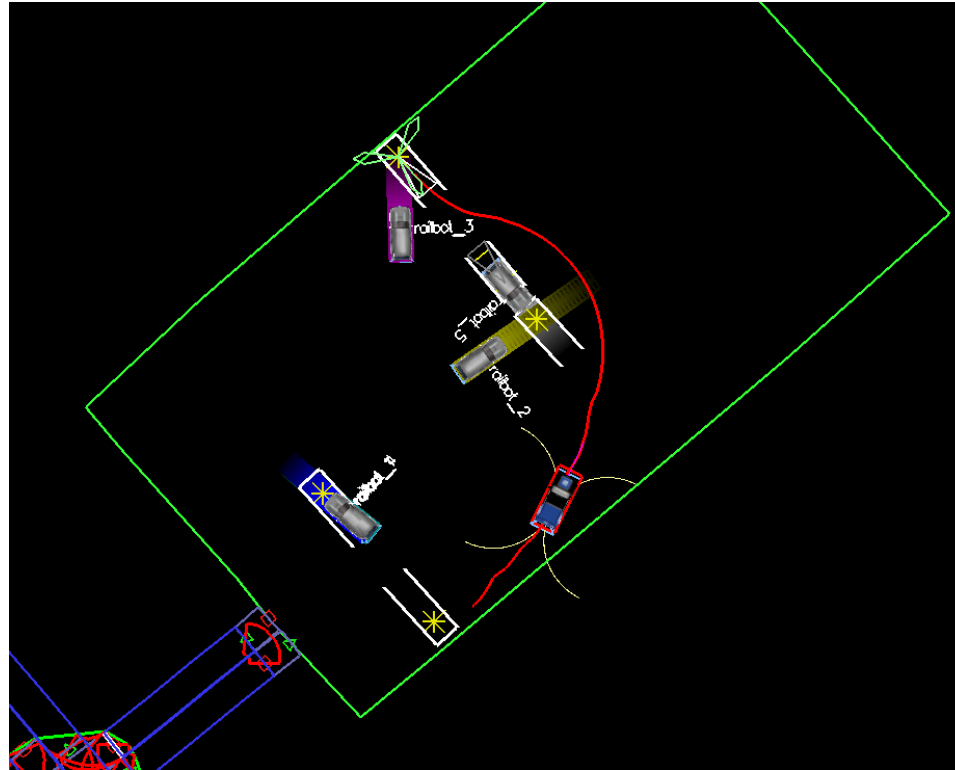
*initial path*



*converged (to optimal) path*

# Desired Properties

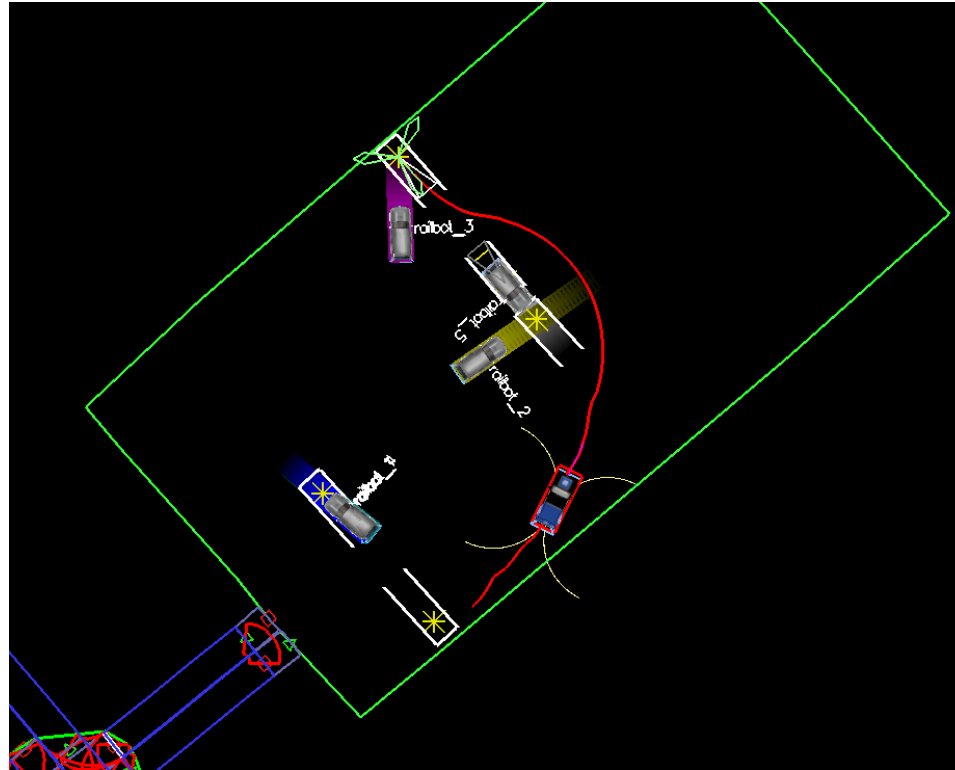
- Fast replanning, especially since we need to avoid other vehicles



*planning a path that avoids other vehicles*

# Desired Properties

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



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



# Our Approach

---

- Build a graph
  - multi-resolution version of a lattice graph
- Search the graph for a least-cost path
  - Anytime D\* [Likhachev et al. '05]

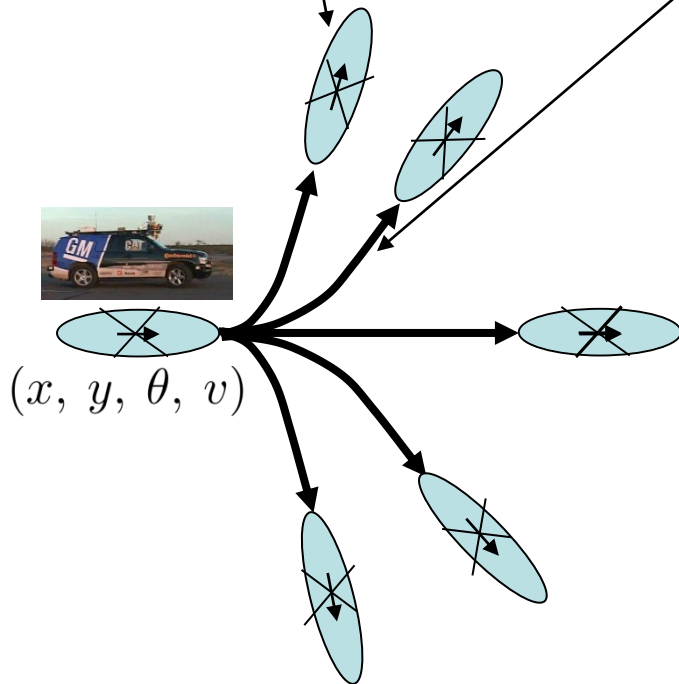
# Building the Graph

- Lattice-based graph [Pivtoraiko & Kelly, '05]:

*outcome state is the center of the corresponding cell*

*each transition is feasible  
(constructed beforehand)*

*action template*



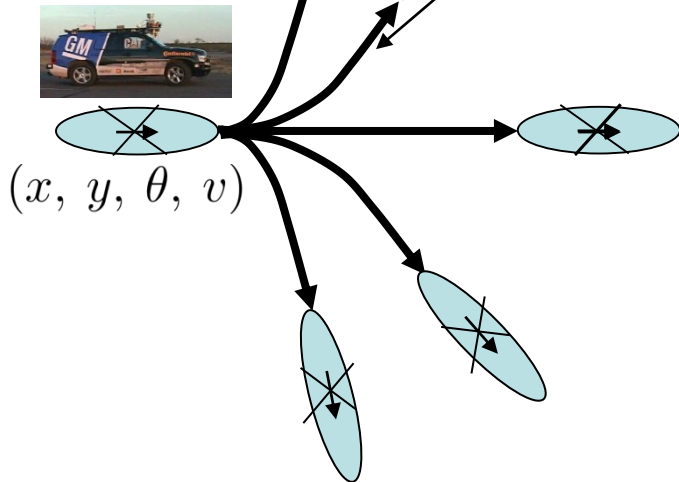
# Building the Graph

- Lattice-based graph [Pivtoraiko & Kelly, '05]:

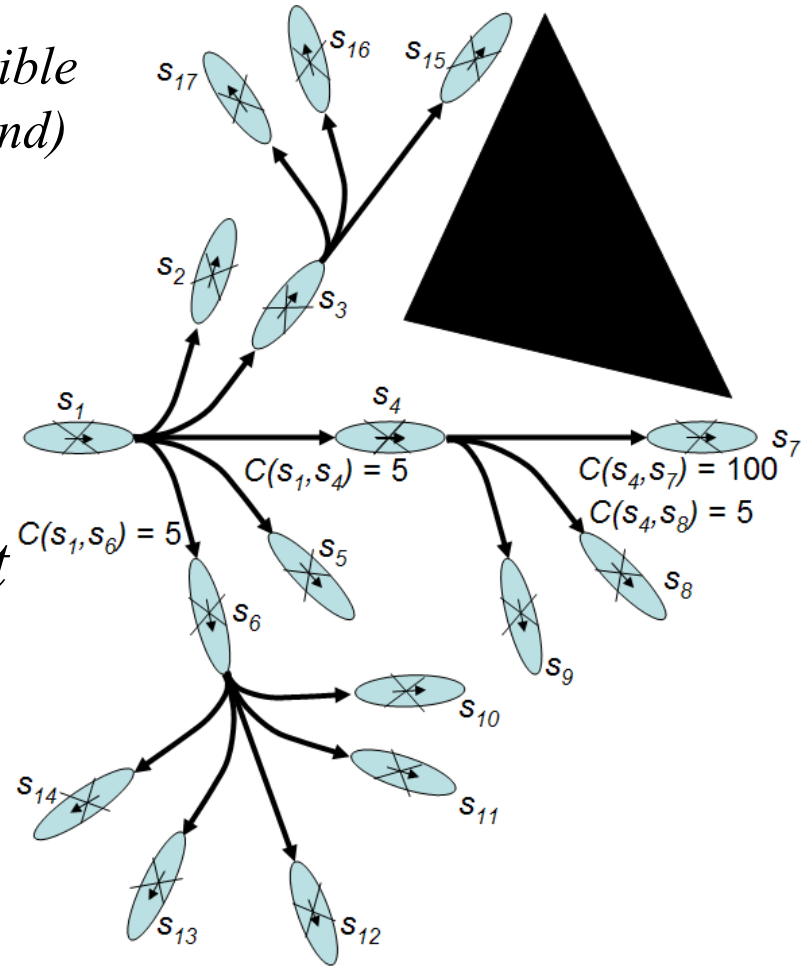
*outcome state is the center of the corresponding cell*

*each transition is feasible  
(constructed beforehand)*

*action template*



*replicate it  
online*



# Building the Graph

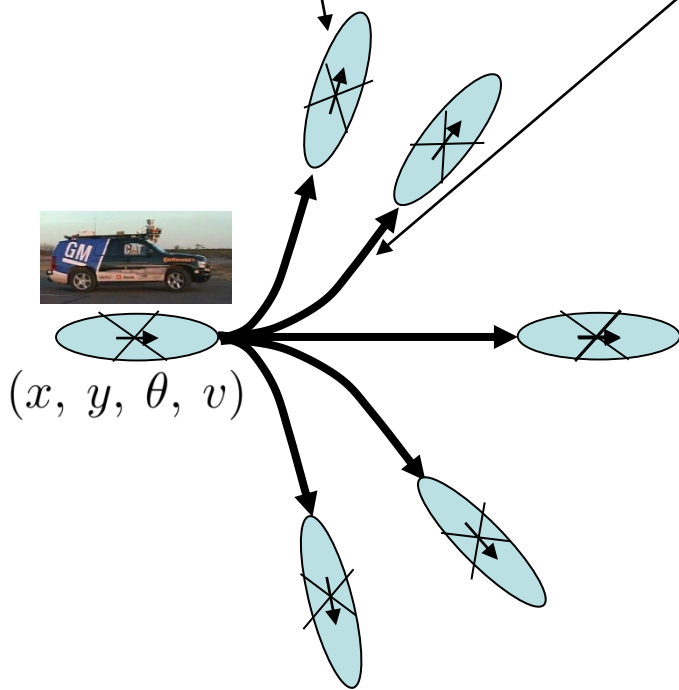
- Lattice-based graph [Pivtoraiko & Kelly, '05]:

*outcome state is the center of the corresponding cell*

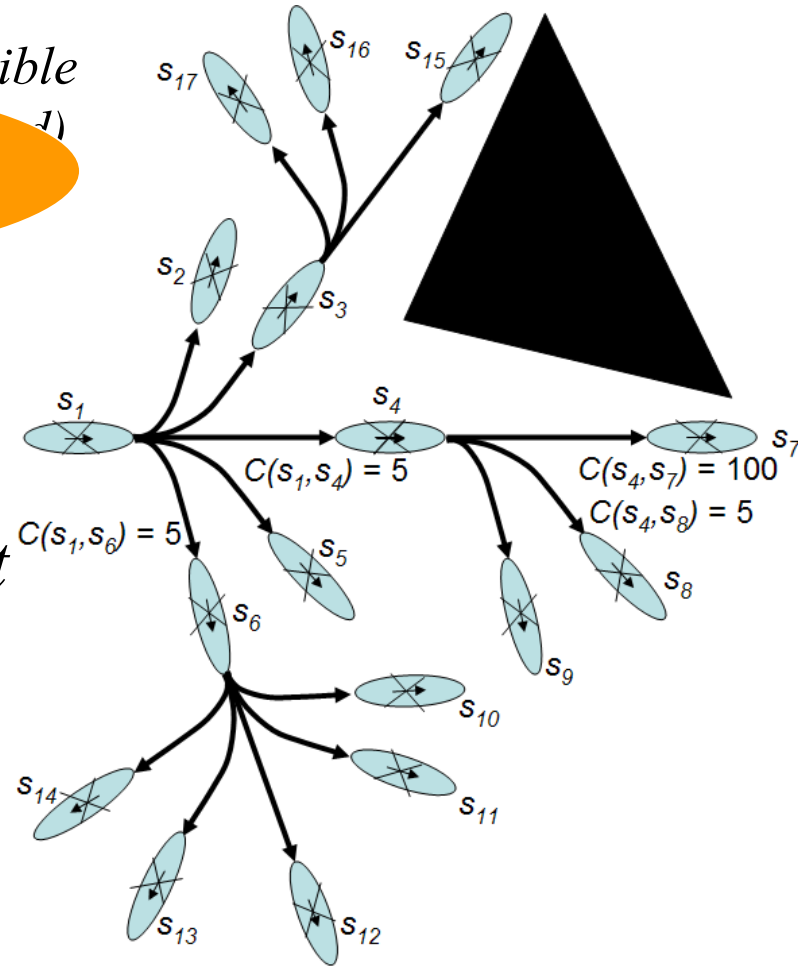
*each transition is feasible*

*we will be searching this graph for  
a least-cost path from  $s_{start}$  to  $s_{goal}$*

*action*

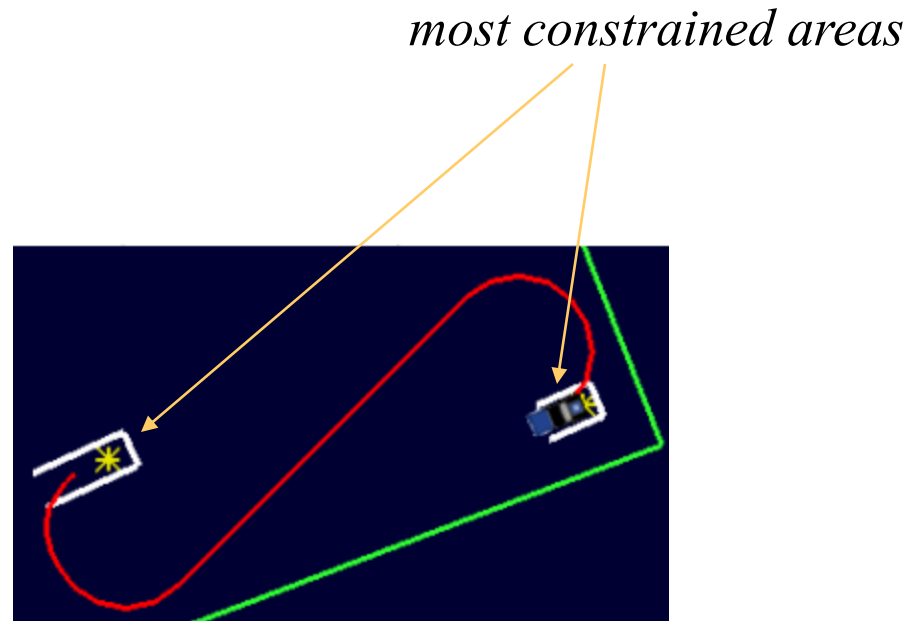


*replicate it  
online*



# 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

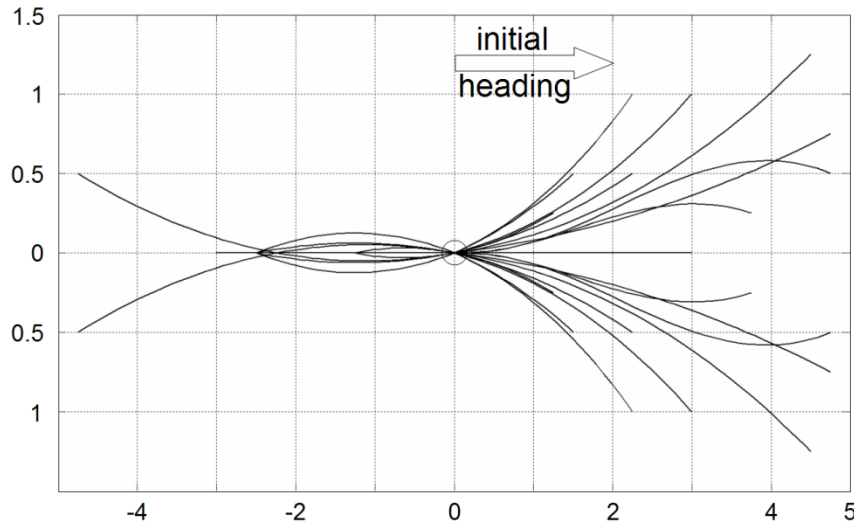
*reduces the size of the state-space for the low-res. lattice*

*both allow for seamless transitions*

# Building the Graph

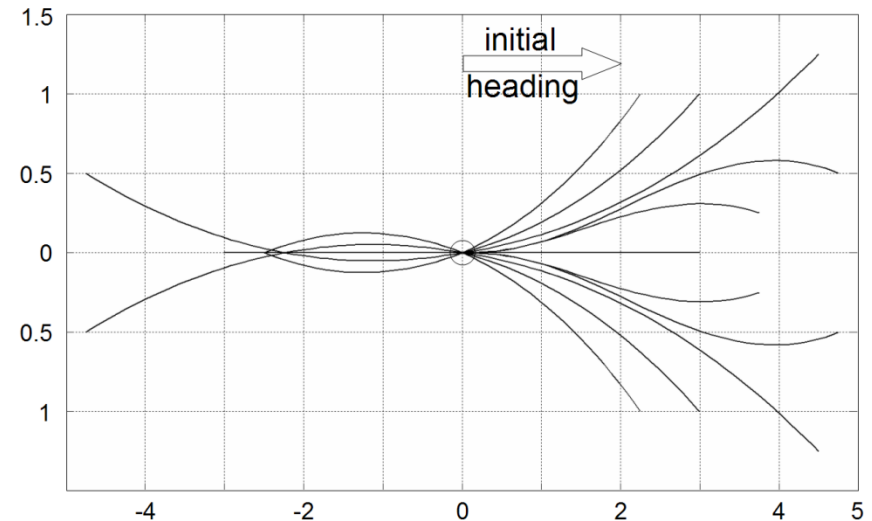
- Multi-resolution lattice used for Urban Challenge:

*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

*can be multiple levels*

*can also be non-uniform in x,y & v*

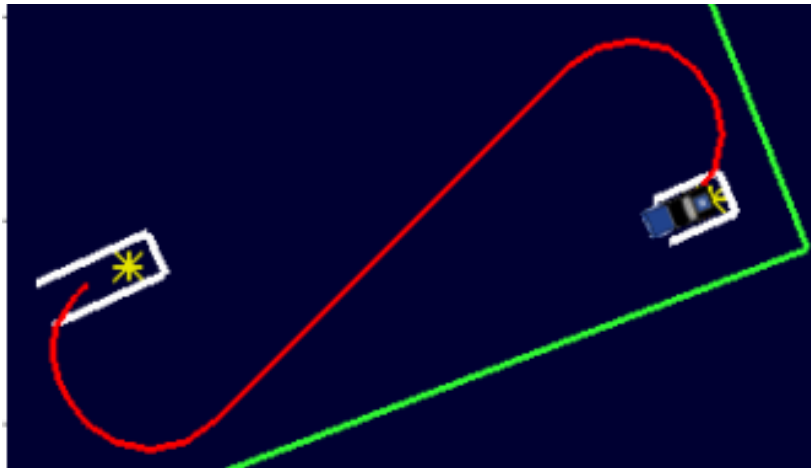


# Building the Graph

- 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\* [Likhachev et al. '05]:
  - 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\* [Likhachev et al.]

*desired bound on the*

*computes a path reusing all  
of the previous search efforts*

set  $\epsilon$  to large value;

until goal is reached

    ComputePathwithReuse();

    publish  $\epsilon$ -suboptimal path for execution;

    update the map based on new sensory information;

    update current state of the agent;

    if significant changes were observed

        increase  $\epsilon$  or replan from scratch;

    else

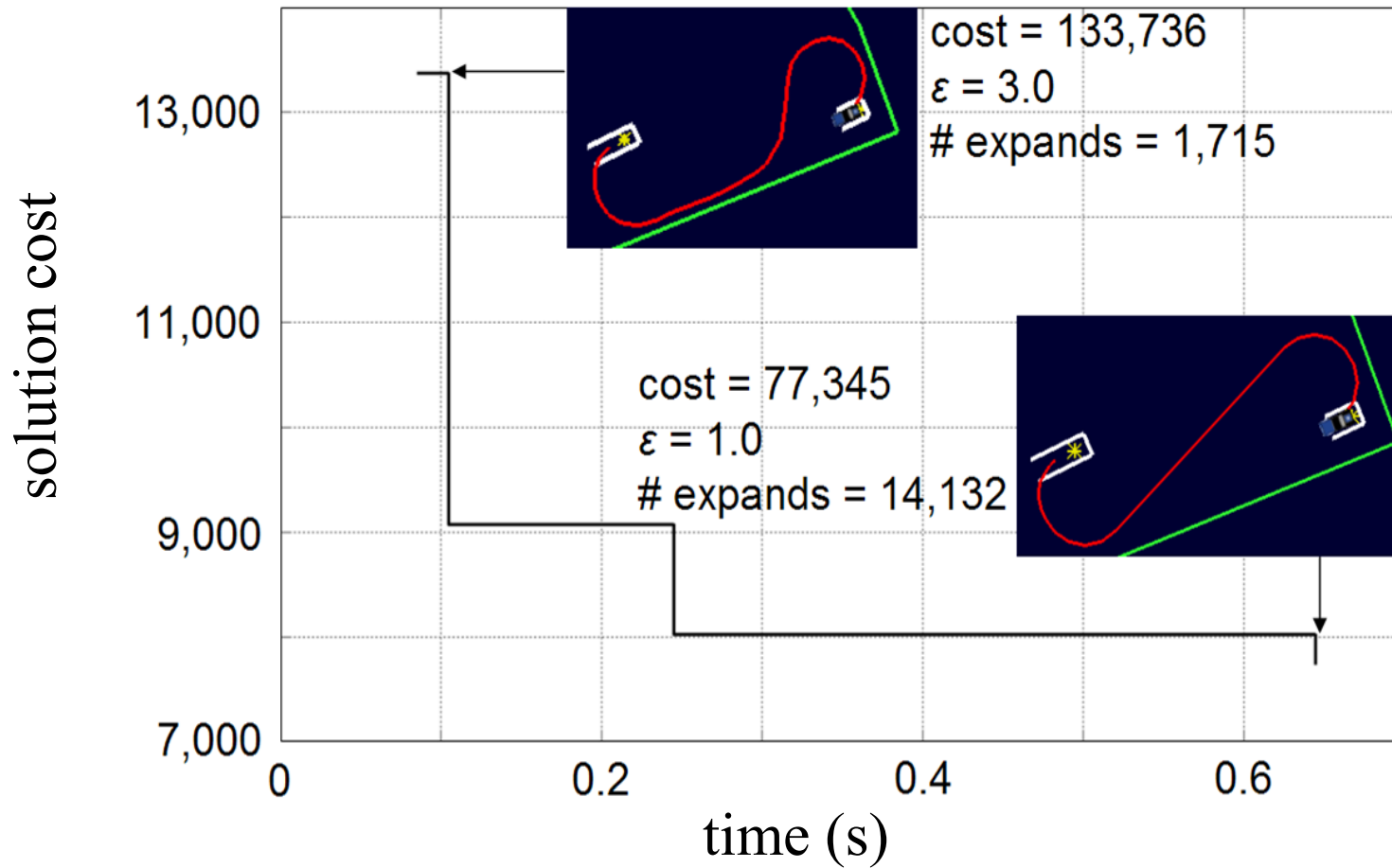
        decrease  $\epsilon$ ;

*guarantees that  
 $\text{cost}(\text{path}) \leq \epsilon \text{ cost}(\text{optimal path})$*

*makes it improve the solution*

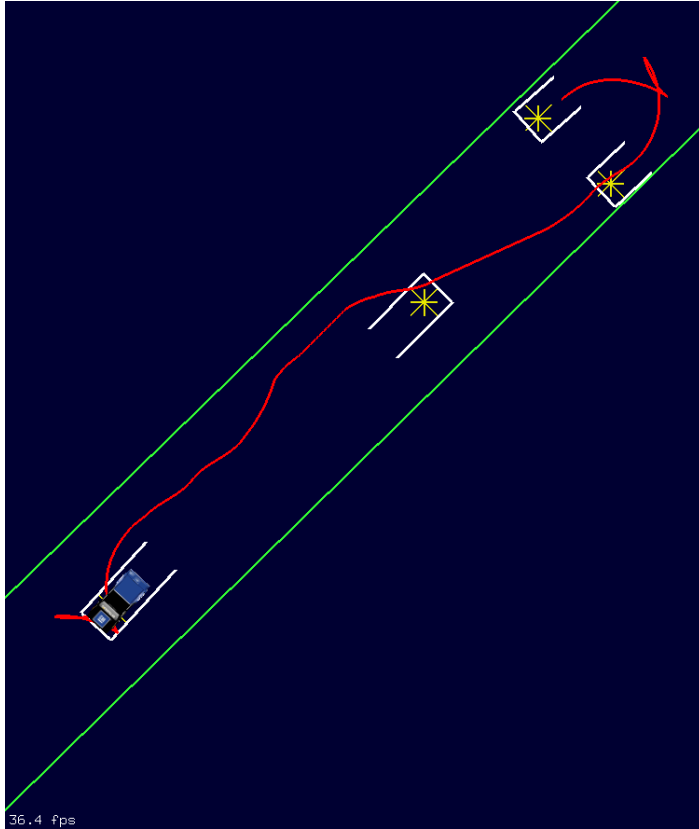
# Searching the Graph

- Anytime behavior of Anytime D\*:

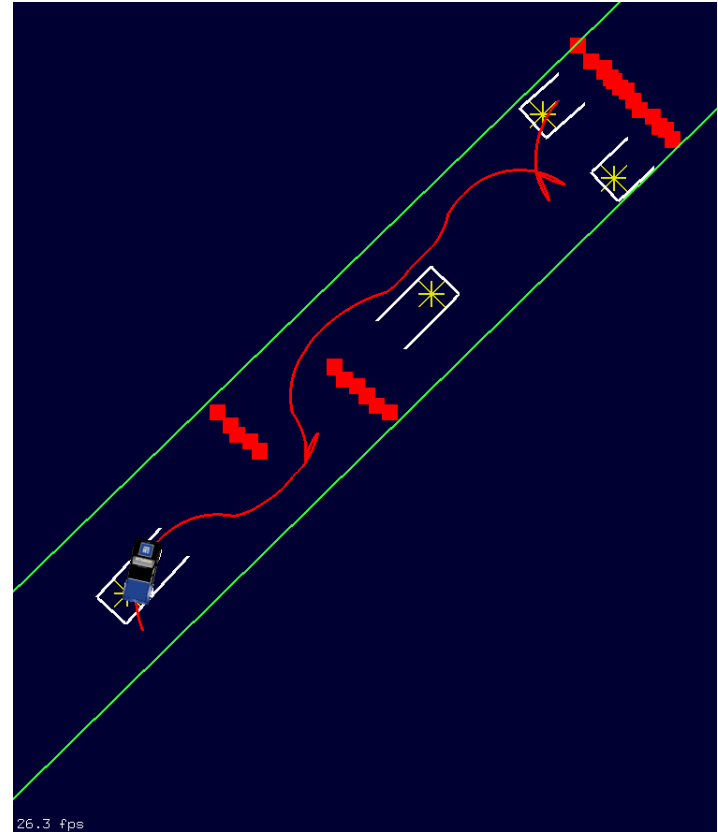


# Searching the Graph

- Incremental behavior of Anytime D\*:



*initial path*

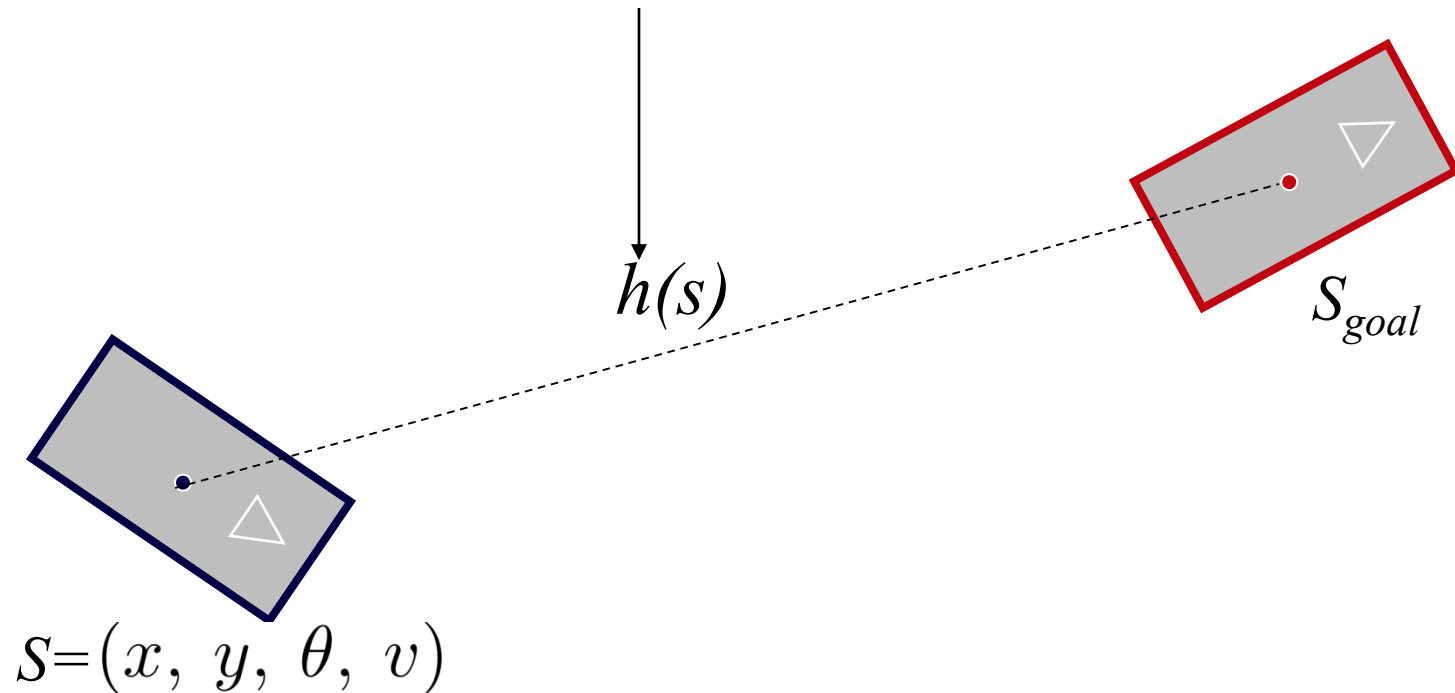


*a path after re-planning*

# Searching the Graph

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

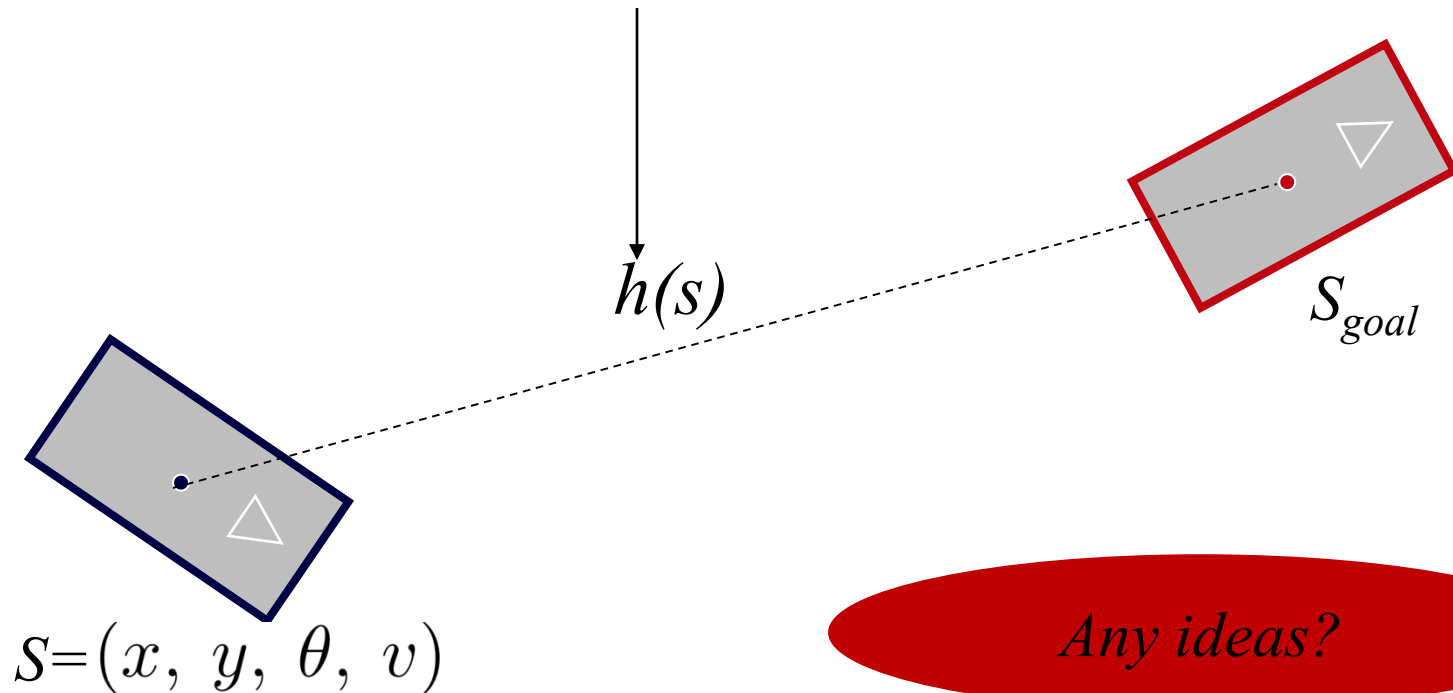
*should be consistent and admissible (never overestimate cost-to-goal)*



# 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)*



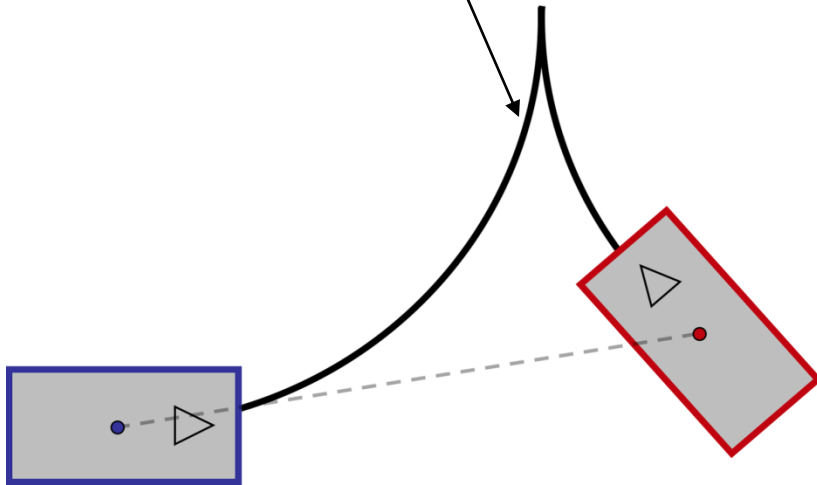
*Any ideas?*



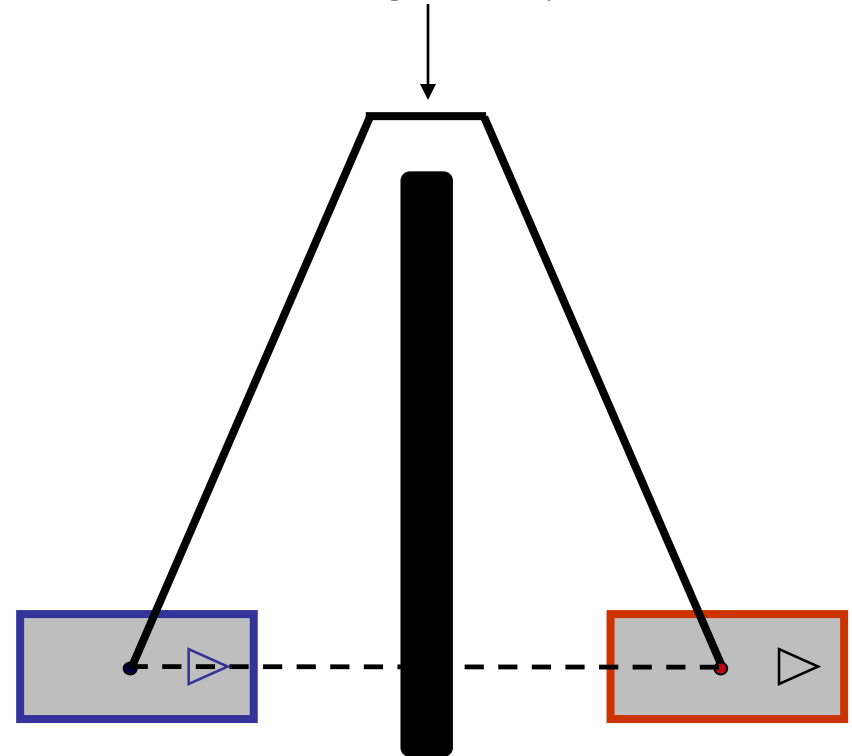
# Searching the Graph

- 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

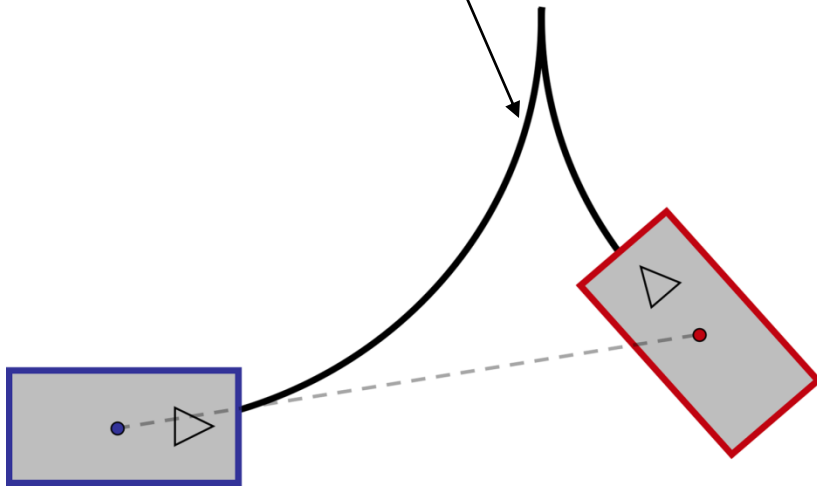


# Searching the Graph

- 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

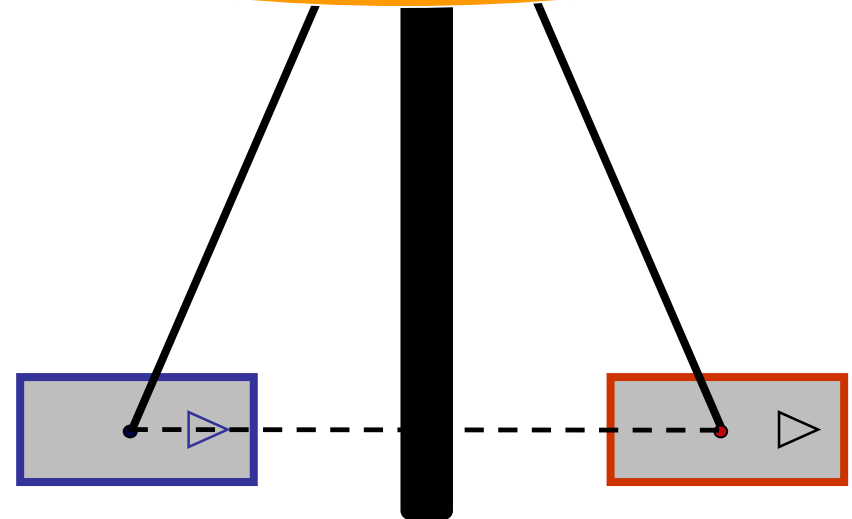
$h_{mech}(s)$  – considers only dynamics constraints and ignores environment

*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:  $h(s) = \max(h_{mech}(s), h_{env}(s))$ , 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*

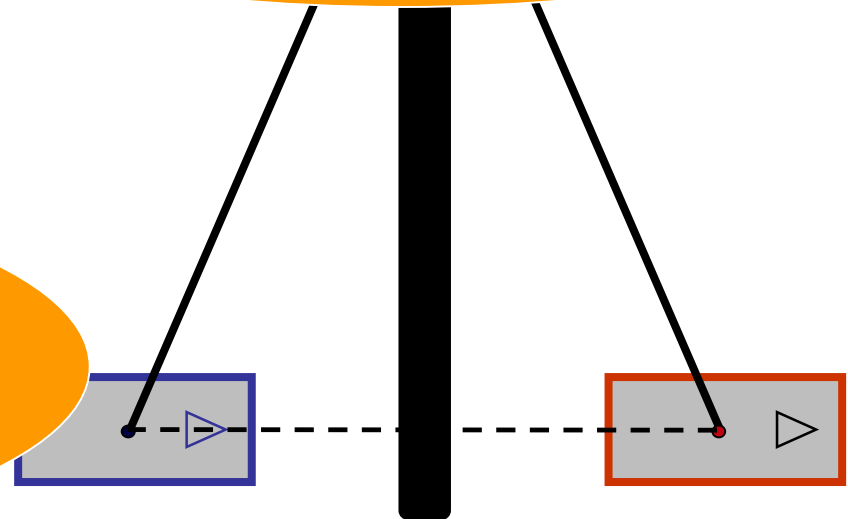
*Any other options?*

*Closed-form analytical solutions  
(Dubins paths [Dubins, '57],  
Reeds-Shepp paths [Reeds & Shepp, '90])*

*Any challenges using it?*

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

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



# Searching the Graph

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

*for efficiency, valid paths, suboptimality bounds, optimality in the limit*

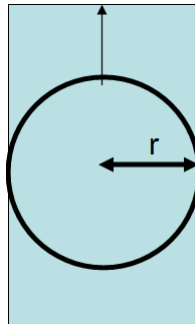
# Searching the Graph

- In our planner:  $h(s) = \max(h_{mech}(s), h_{env}(s))$
- $h(s)$  needs to be admissible and consistent
- if  $h_{mech}(s)$  and  $h_{env}(s)$  are admissible and consistent, then  $h(s)$  is admissible and consistent [Pearl, 84]
- $h_{mech}(s)$  – cost of a path in high-res. lattice with no obstacles and no boundaries

*$h_{mech}(s)$  – admissible and consistent*

# Searching the Graph

- In our planner:  $h(s) = \max(h_{mech}(s), h_{env}(s))$
- $h(s)$  needs to be admissible and consistent
- if  $h_{mech}(s)$  and  $h_{env}(s)$  are admissible and consistent, then  $h(s)$  is admissible and consistent [Pearl, 84]
- $h_{env}(s)$  – cost of a 2D path of the inner circle of the vehicle into the center of the goal location



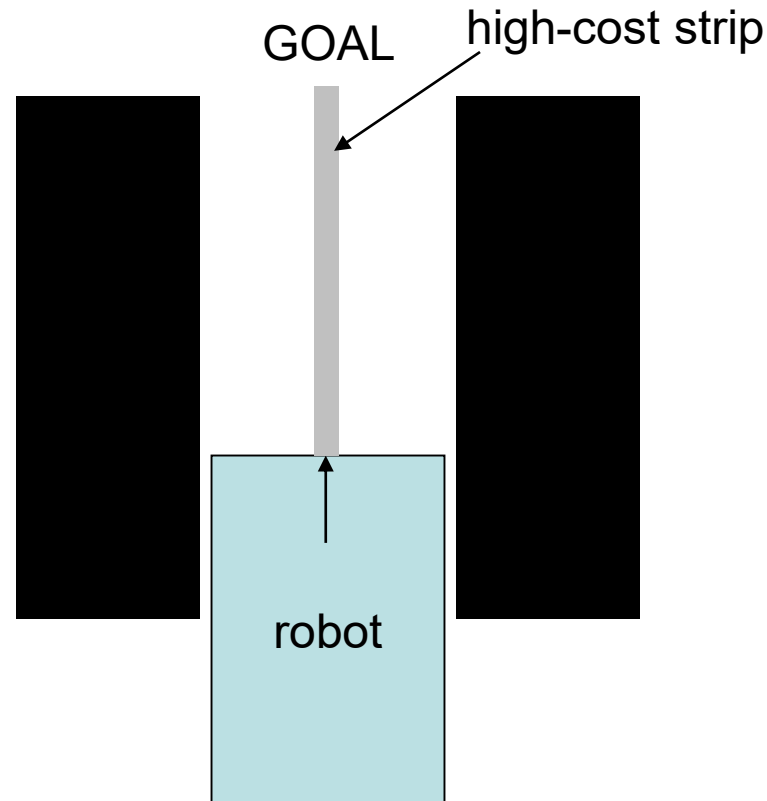
$h_{env}(s)$  – NOT admissible

# Searching the Graph

- In our planner:  $h(s) = \max(h_{mech}(s), h_{env}(s))$

*$h_{env}(s)$  – NOT admissible*

- $h_{env}(s)$  – cost of a 2D path of the inner circle of the vehicle into the center of the goal location



# Searching the Graph

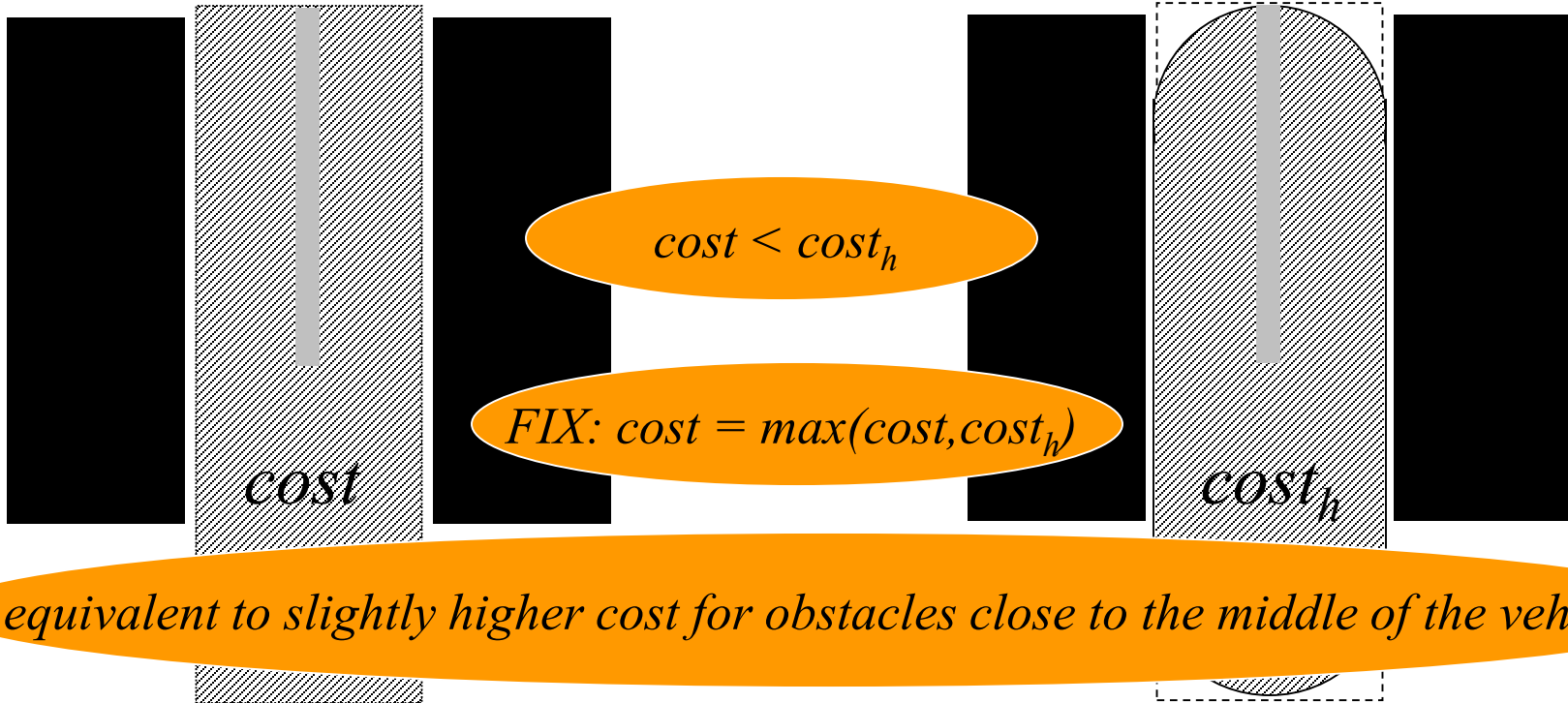
- In our planner:  $h(s) = \max(h_{mech}(s), h_{env}(s))$

$h_{env}(s)$  – NOT admissible

- $h_{env}(s)$  – cost of a 2D path of the inner circle of the vehicle into the center of the goal location

*cost = average over this box (convolution)*

*according to  $h_{env}(s)$ :  
cost = average over the trace of inner circle*





# 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

# 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  $\varepsilon$  times the cost of a least-cost path from the vehicle configuration to the goal configuration using actions in the multi-resolution lattice, where  $\varepsilon$  is the current value by which Anytime  $D^*$  inflates heuristics.*

# Searching the Graph

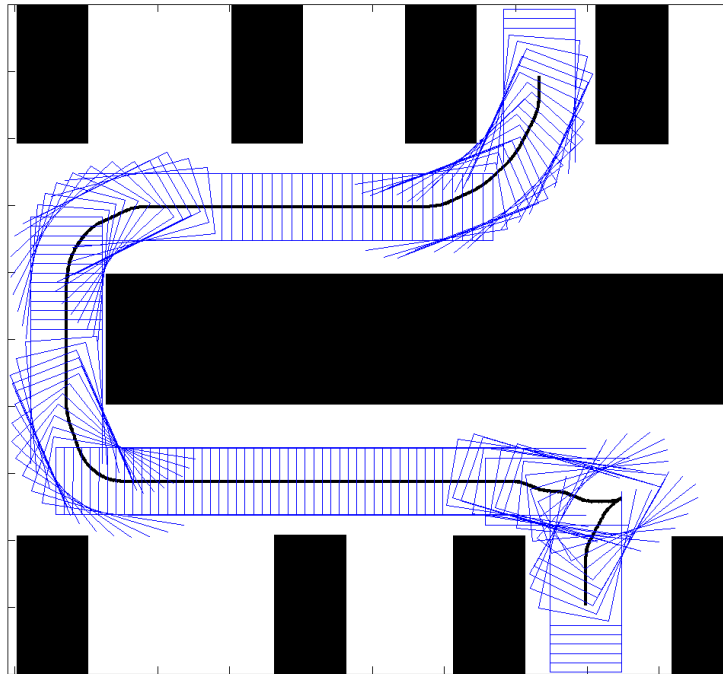
- Benefit of the combined heuristics:



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

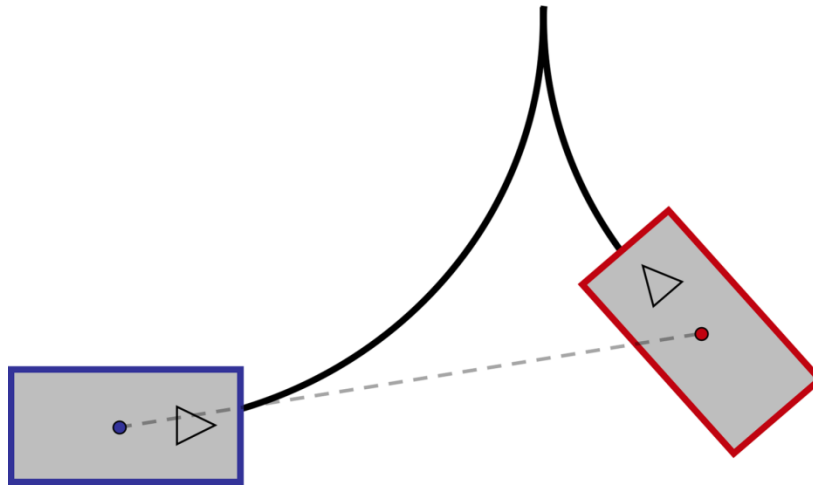
# Optimizations

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



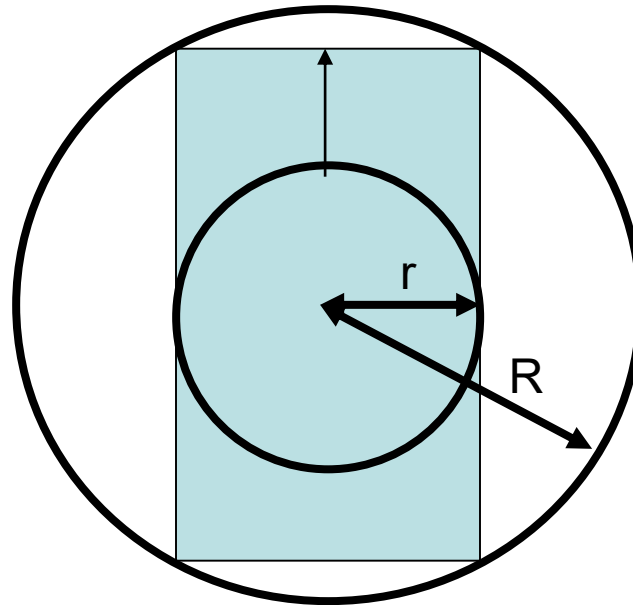
# Optimizations

- Pre-compute as much as possible
  - mechanish-constrained heuristics



# Optimizations

- avoid convolutions based on collision checking with inner and outer circles



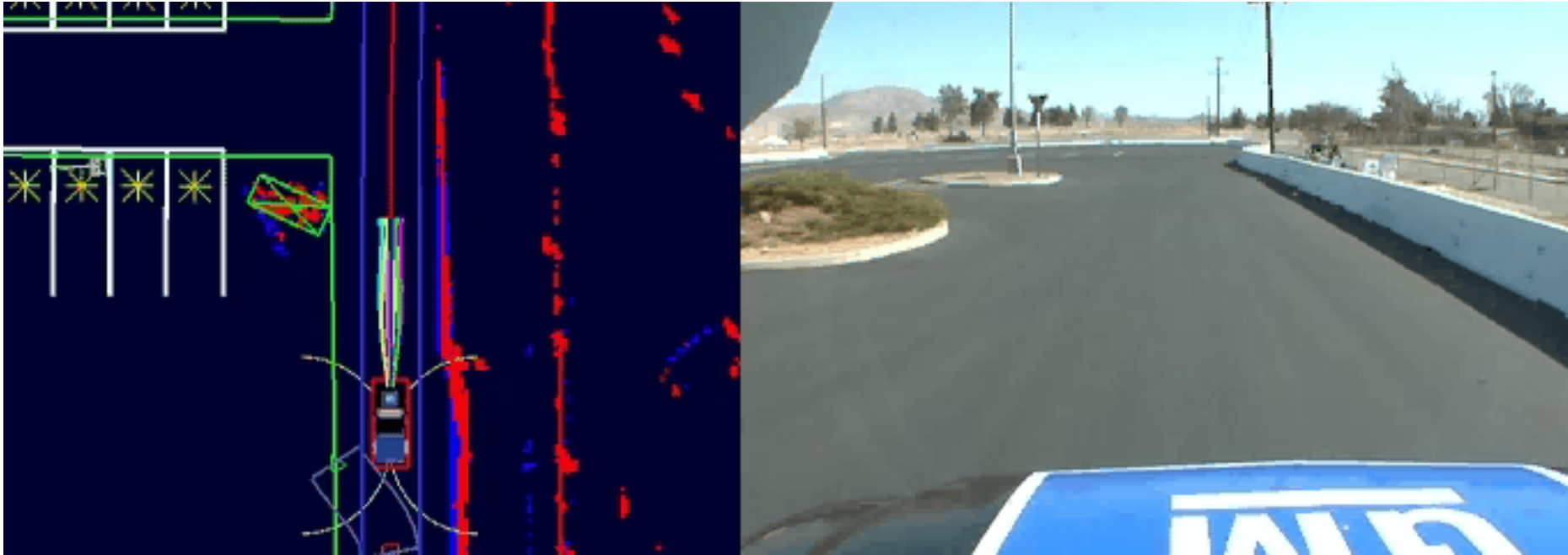
# Optimizations

---

- Efficient re-planning by maintaining low-resolution boolean map of states expanded
  - each map update may affect thousands of states
  - need to iterate over those states to see if they are effected
  - **optimization:** iterate and update edge costs only when map update is in the area that have states expanded

# Results

- Plan improvement



*Tartanracing, CMU*



# Results

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



*Tartanracing, CMU*

# What You Should Know...

---

- Different types of planning for autonomous driving and how they interact
- What is multi-resolution lattice
- Different heuristic functions used in Motion Planning