V-CHARGE

Motion Planning for Automated Cars

Ulrich Schwesinger, ETH Zurich

09.07.2014

Planning for Automated Cars

- Great effort in automated driving recently
  - Google, Daimler, Audi, Volvo, Carnegie Mellon, Parma´s VisLab, V-Charge and more
  - Close-to-market solutions for e.g. automated highway driving and automated parking

"Mercedes's autonomous driving on highway"
http://www.youtube.com/watch?v=4jW0fJ80VG8

V-CHARGE

Challenges

- Challenges
  - Coping with high-complexity urban environments
  - Accounting for dynamics
    - Combustion engine effects
    - Kinematic vehicle constraints
    - Other cars, pedestrians, cyclists
  - Dealing with uncertainty
    - Jumping localization
    - Uncertain perception
  - Real-time operation

"A Ride in the Google Self Driving Car"
http://www.youtube.com/watch?v=TsaES--OTzM

V-CHARGE

Challenges

- Challenges
  - How to proof safety of automated vehicles?
    - World contains infinite amount of different situations
    - 100% safety impossible in urban environments (frozen robot problem)
  - Legal Issues
    - Vienna Convention on Road Traffic (1968, article 8, paragraph 5):
      - “Every driver shall at all times be able to control his vehicle or to guide his animals.”

“Autonomous Crash in DARPA Urban Challenge”
http://www.youtube.com/watch?v=bnv5JP8gL_k

V-CHARGE

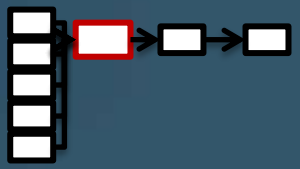
Session Goal

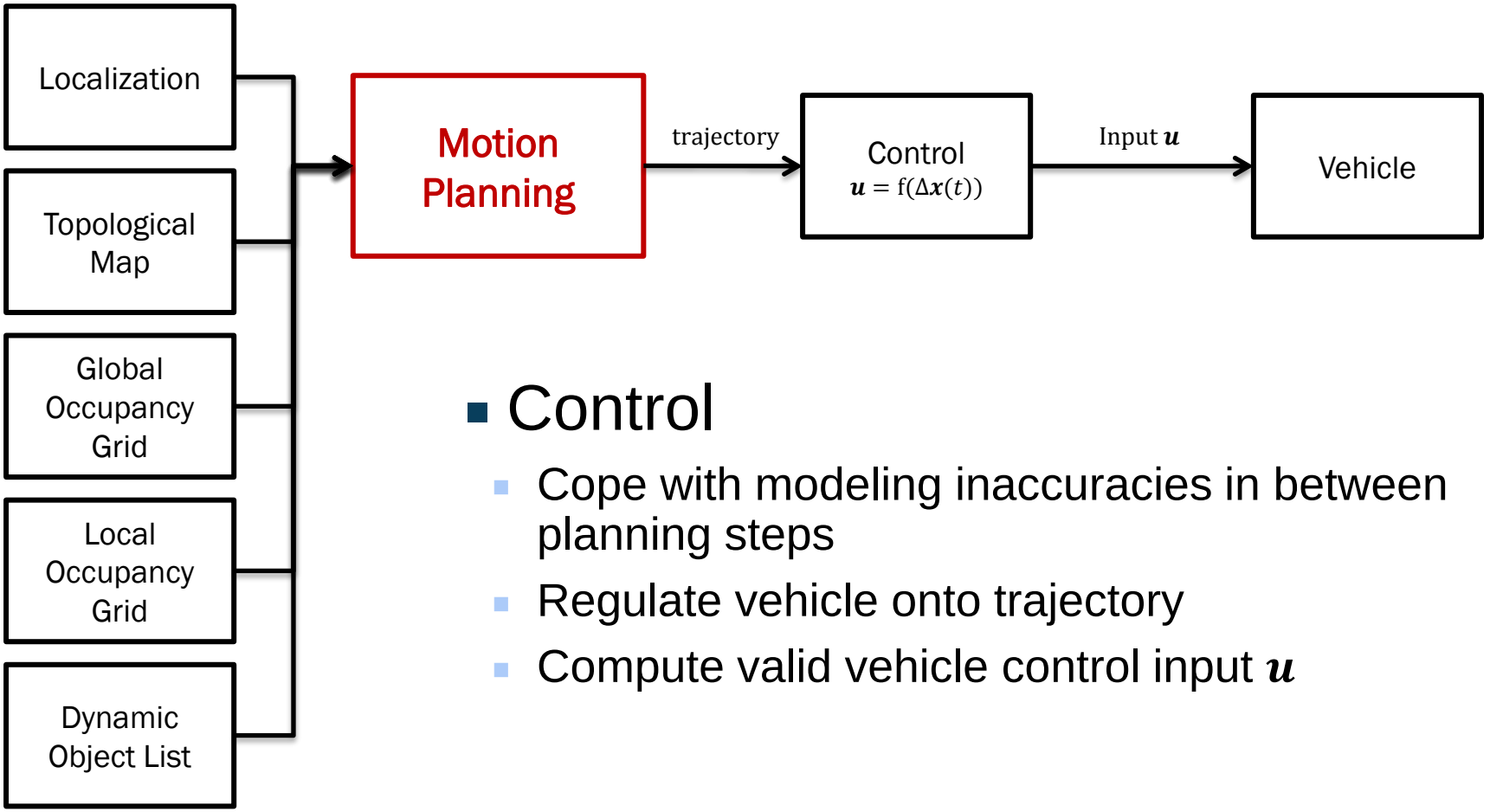
- Give you knowledge and tools at hand to design a motion planning framework for an automated car
- Focus on well-established, real-world approved approaches rather than sophisticated „lab-environment“ planners

Overview

- The Basic Motion Planning Problem
  - I/O
  - Notions and Terms
- Selected algorithms for motion planning
  - The Dynamic Window Approach
  - Graph-Search, State-Lattices
  - Randomized Approaches: RRT, RRT*
- Collision Detection
- Detailed case studies from DARPA Urban Challenge
  - Boss (Carnegie Mellon)
  - AnnieWay (Karlsruhe/Munich)
  - Junior (Stanford)
- The V-Charge Motion Planning Framework
- Getting started for the technical session

Motion Planning I/O

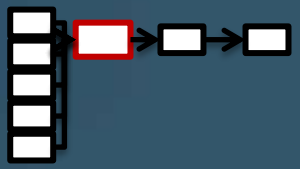


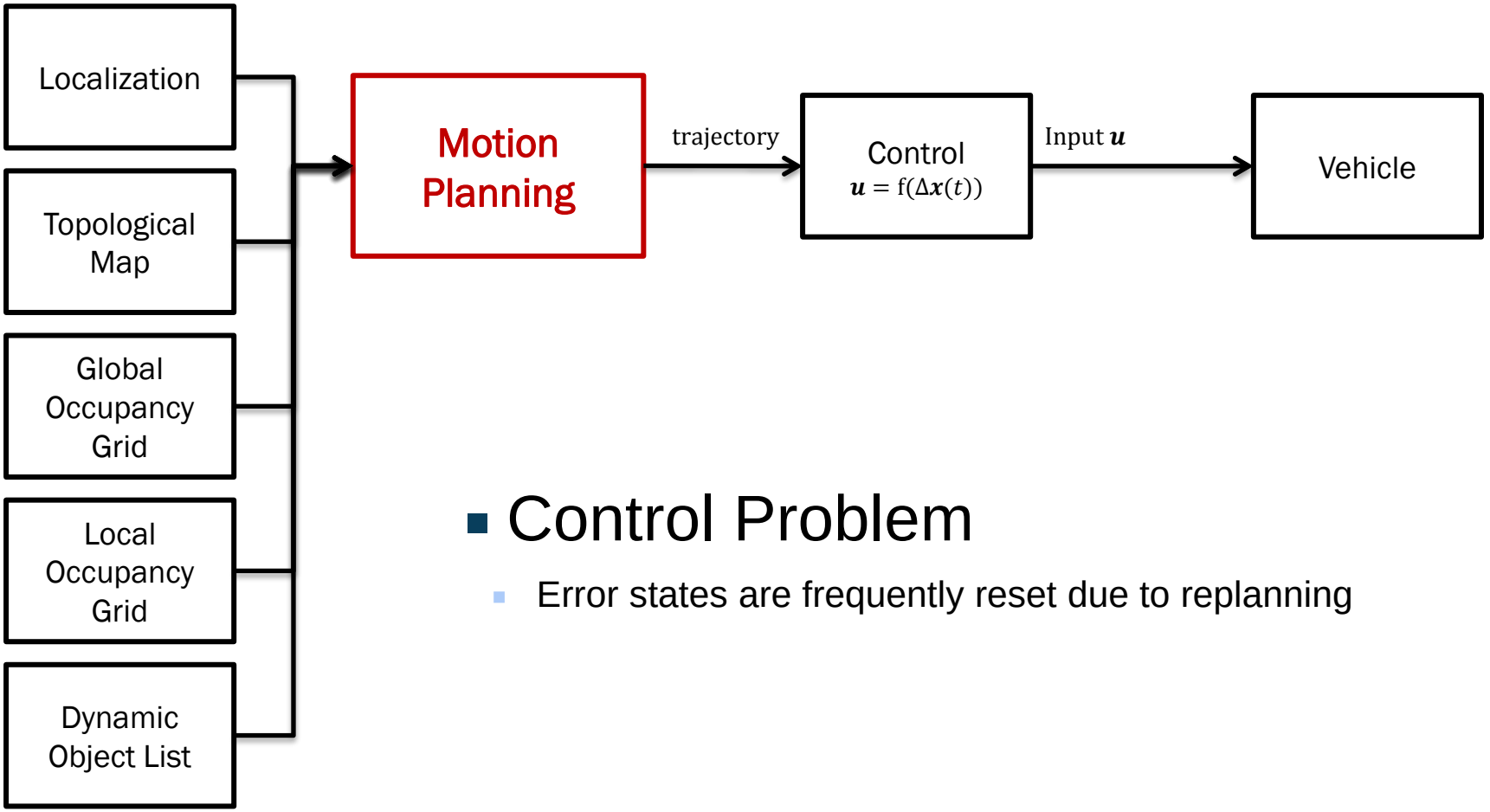


- Control
  - Cope with modeling inaccuracies in between planning steps
  - Regulate vehicle onto trajectory
  - Compute valid vehicle control input $\boldsymbol{u}$

V-CHARGE

Motion Planning I/O

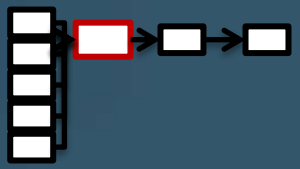


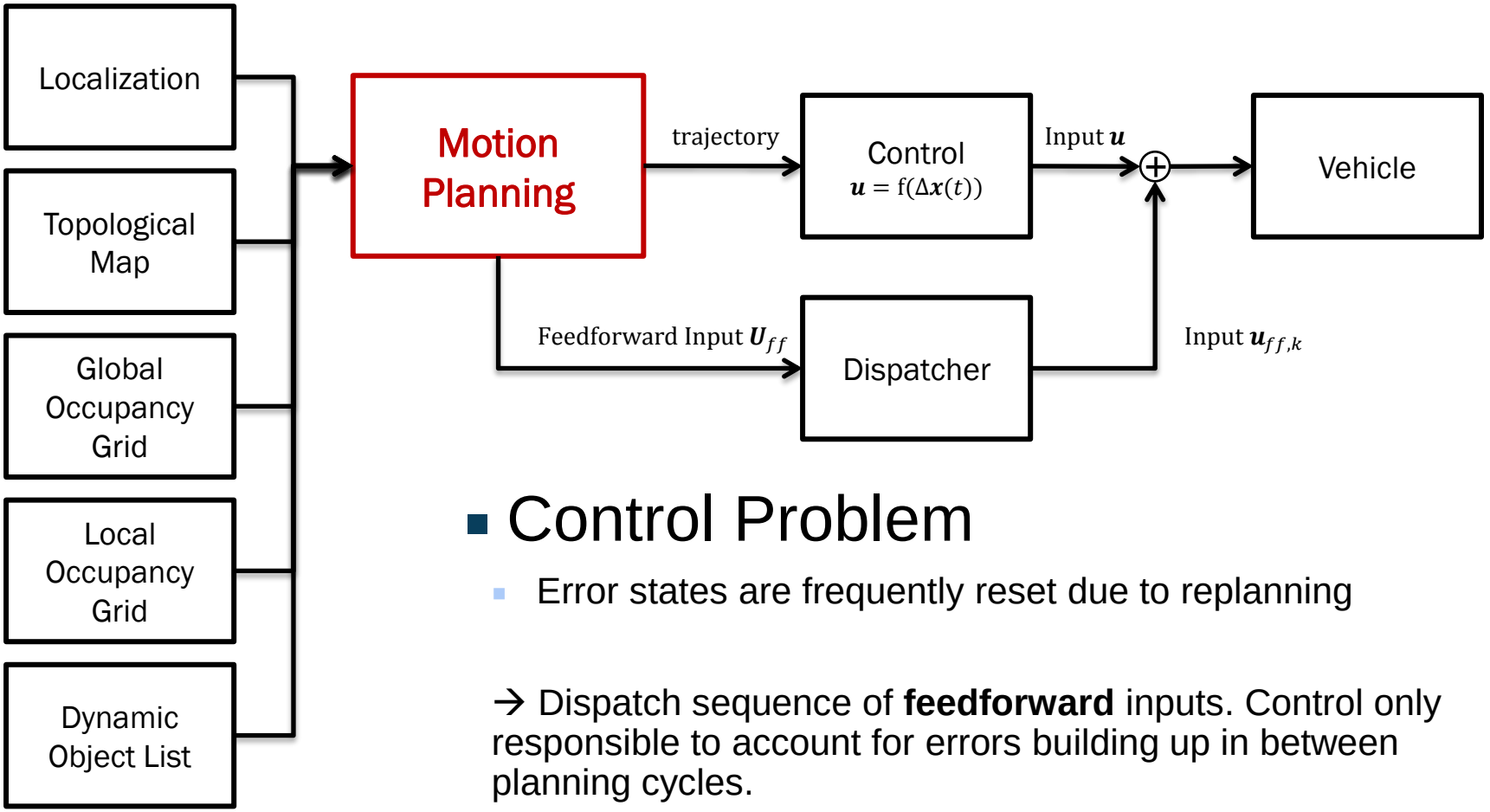


- Control Problem
  - Error states are frequently reset due to replanning

V-CHARGE

Motion Planning I/O





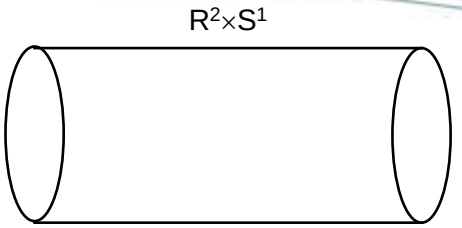
- Control Problem
  - Error states are frequently reset due to replanning

→ Dispatch sequence of **feedforward** inputs. Control only responsible to account for errors building up in between planning cycles.

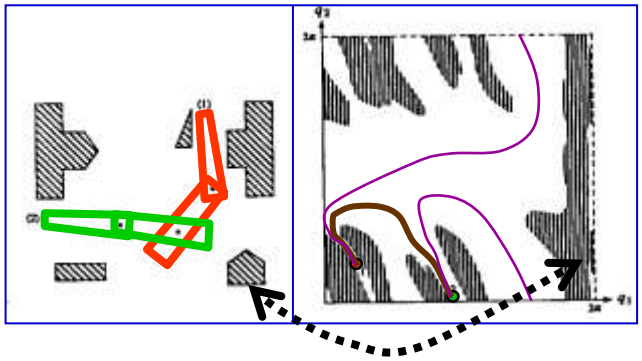
V-CHARGE

Notions and Terms

- Robots operate in the Euclidean workspace $W = R^N$ with $N = 2$ or 3.
  - For automated vehicles usually a $2D$ workspace representation is sufficient
- The configuration-space C: space of all the robot's possible configurations
- A C-obstacle is the set of configurations where the robot collides with a given workspace obstacle
- Some constraints can't be modeled as C-Space obstacles
  - Non-holonomic constraints
  - Global constraints: finite length, power consumption, etc…



Configuration space for a rigid body operating in $W = R^2$



C-obstacles for an articulated robot with 2 joints

Holonomic and Non-Holonomic Systems

- Holonomic Constraints
  - Differential constraints are integrable. They may be expressed as constraints on the robot's pose
  - Restrict the robots configuration space
  - The robot is able to move instantaneously in any direction in the space of its degrees of freedom
  - e.g. revolute joints for an articulated robot
- Non-holonomic Constraints
  - Differential constraints are not integrable. There is no way to express them as constraints on the robot's pose
  - Do not restrict the robot´s configuration space, only the space of differential motions
  - The robot is not able to move instantaneously in every direction in the space of its degrees of freedom

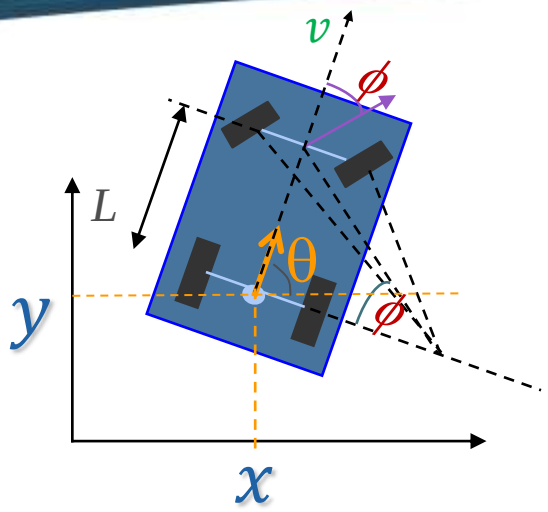
These slides are based on the excellent course material of Prof. Kris Hauser from Indiana University
http://www.cs.indiana.edu/classes/b659-hauserk/schedule.htm

A Holonomic Mobile Robot

Photographer: Frank C. Müller via Wikimedia Commons

- The office chair
  - Castor wheels impose no differential constraints
  - In any configuration, it is possible to choose wheel velocities that move the robot with any velocity within the plane

A Non-Holonomic Mobile Robot



$$\mathrm{d}x/\mathrm{d}t = v\cos\theta$$
$$\mathrm{d}y/\mathrm{d}t = v\sin\theta$$
$$\rightarrow \mathrm{d}x\sin\theta - \mathrm{d}y\cos\theta = 0$$

(rolling-without-slipping)

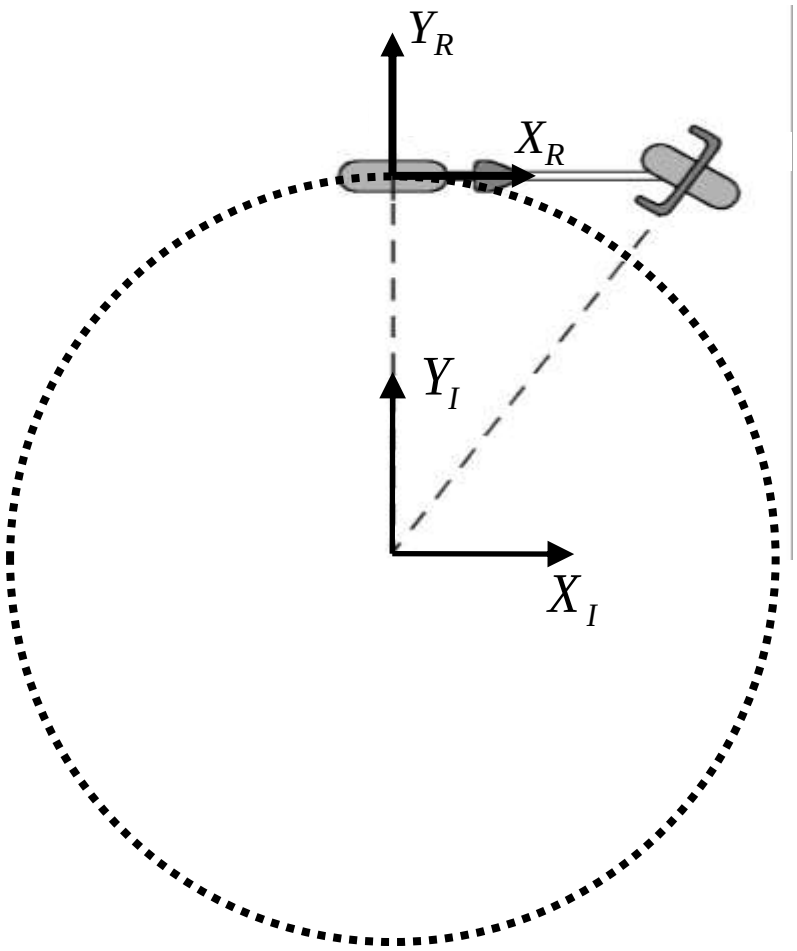
$$\mathrm{d}\theta/\mathrm{d}t = v/L \cdot \tan\phi$$
$$|\phi| < \phi_{\max}$$

- An Ackermann-steered vehicle
  - Workspace: R^2
    - all positions in the 2D plane $\{(x, y)\}$
  - Configuration space: $R^2 \times S^1$
    - all poses in the 2D plane $\{(x, y, \theta)\}$
  - Regardless of the configuration, instantaneous motion in lateral direction is not possible (rolling-without-slipping constraint)

These slides are based on the excellent course material of Prof. Kris Hauser from Indiana University
http://www.cs.indiana.edu/classes/b659-hauserk/schedule.htm

V-CHARGE

A Holonomic Mobile Robot



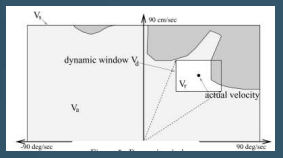
- A bicycle with **fixed steering**
  - Two non-steerable wheels
  - The configuration space of the robot collapses to a single degree of freedom: a circle
  - The differential constraints can be expressed as constraints on position
  - In any configuration, it is possible to choose wheel velocities that move the robot in any direction within its configuration space.

These slides are based on the excellent course material of Prof. Kris Hauser from Indiana University
http://www.cs.indiana.edu/classes/b659-hauserk/schedule.htm

Overview

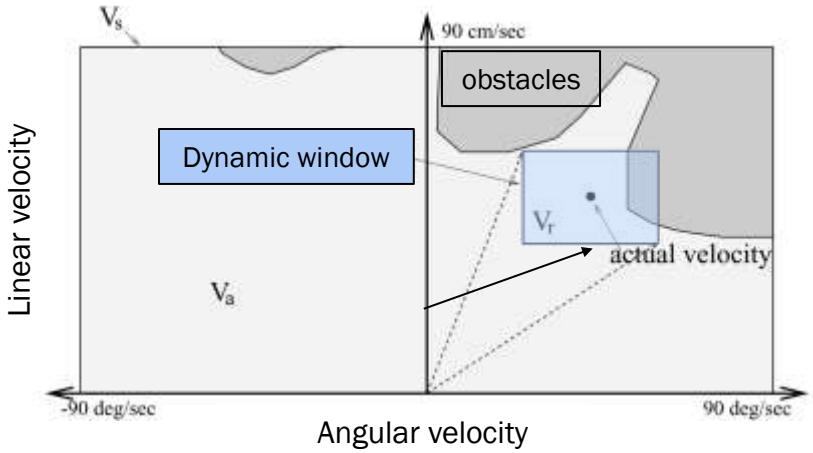
- The Basic Motion Planning Problem
  - I/O
  - Notions and Terms
- Selected algorithms for motion planning
  - The Dynamic Window Approach
  - Graph-Search, State-Lattices
  - Randomized Approaches: RRT, RRT*
- Collision Detection
- Detailed case studies from DARPA Urban Challenge
  - Boss (Carnegie Mellon)
  - AnnieWay (Karlsruhe/Munich)
  - Junior (Stanford)
- The V-Charge Motion Planning Framework
- Getting started for the technical session

Dynamic Window Approach: Search Space



- Maximization of objective function in robot´s velocity space (linear/angular)
  - Assume change of inputs only in next timestep t_{k+1}
  - Constant inputs over $t_{k+1} \dots t_{k+N}$

  → search space $\mathbb{R}^{|U|}$
- Acceleration constraints (linear/angular) limit set of feasible velocities in next time-step t_{k+1}
  → *dynamic window*

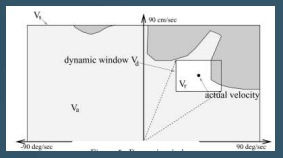


Fox, D., Burgard, W., & Thrun, S. (1997). The dynamic window approach to collision avoidance. *Robotics Automation Magazine, IEEE*, *4*(1), 23–33.

Fox, D., Burgard, W., & Thrun, S. (1997). The dynamic window approach to collision avoidance. *Robotics Automation Magazine, IEEE*, *4*(1), 23–33.

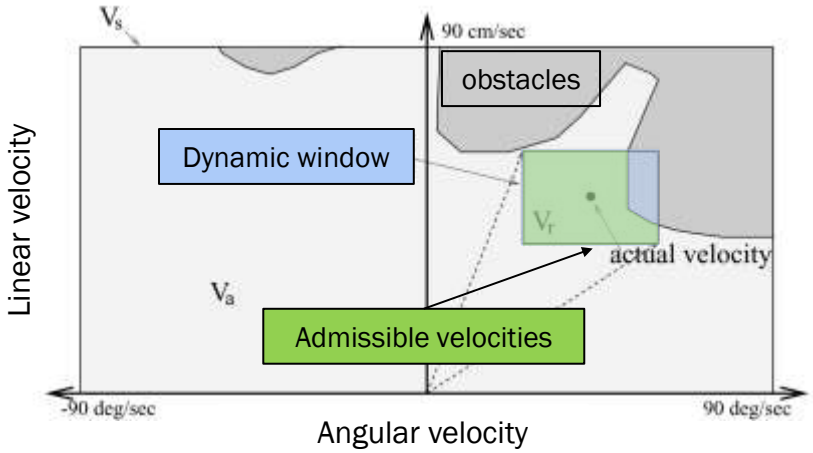
V-CHARGE

Dynamic Window Approach: Search Space



- Maximization of objective function in robot´s velocity space (linear/angular)
  - Assume change of inputs only in next timestep
  - Constant inputs over

  → search space
- Acceleration constraints (linear/angular) limit set of feasible velocities in next time-step
  → *dynamic window*
- Velocities resulting in inevitable collisions shrink search space
  → *admissible velocities*

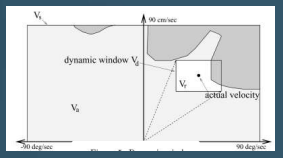


Fox, D., Burgard, W., & Thrun, S. (1997). The dynamic window approach to collision avoidance. *Robotics Automation Magazine, IEEE*, *4*(1), 23–33.

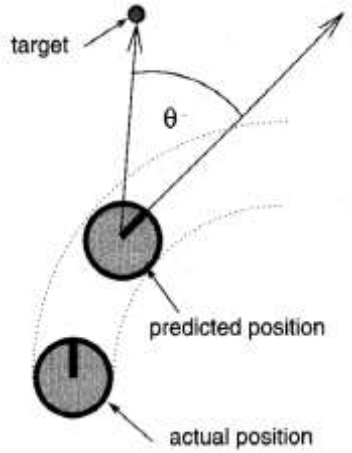
Fox, D., Burgard, W., & Thrun, S. (1997). The dynamic window approach to collision avoidance. *Robotics Automation Magazine, IEEE*, *4*(1), 23–33.

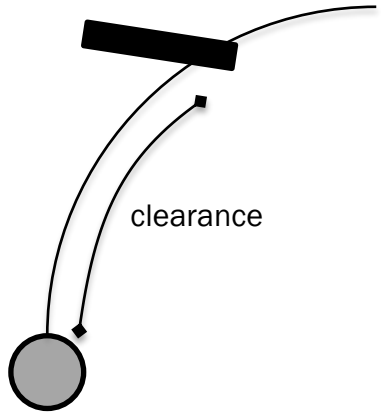
V-CHARGE

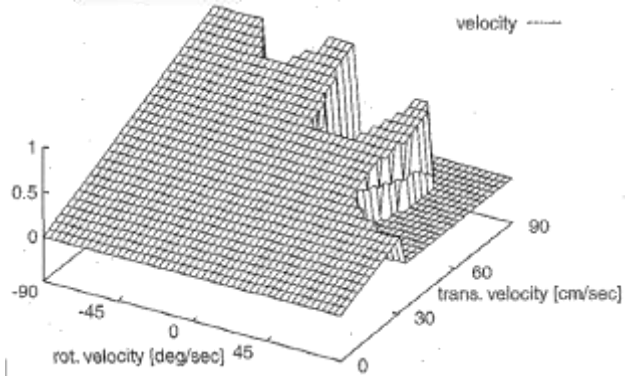
Dynamic Window Approach: Objective Function



$$G(v,\omega) = \text{smoothing\_fcn}(\alpha \cdot heading\_term(v,\omega) + \beta \cdot clearance\_term(v,\omega) + \gamma \cdot speed\_term(v,\omega))$$







$heading\_term(v,\omega)$: measures the **alignment** of the robot with the target direction

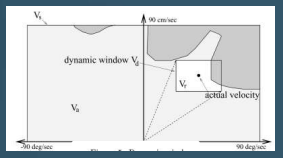
$clearance\_term(v,\omega)$: responsible for **collision avoidance**

$speed\_term(v,\omega)$: reward high translational speeds, respectively **progress towards goal**

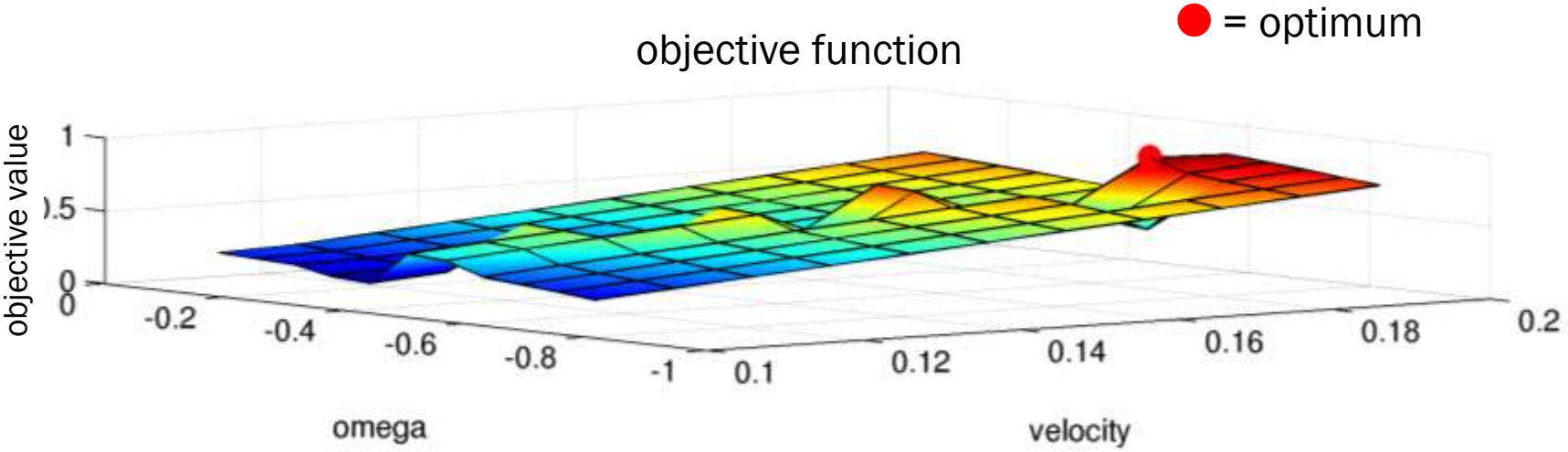
Fox, D., Burgard, W., & Thrun, S. (1997). The dynamic window approach to collision avoidance. *Robotics Automation Magazine, IEEE*, *4*(1), 23–33.

V-CHARGE

Dynamic Window Approach: Optimization



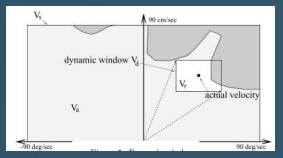
- Maximization of objective function through **discretization of search space** and **exhaustive evaluation**



Fox, D., Burgard, W., & Thrun, S. (1997). The dynamic window approach to collision avoidance. *Robotics Automation Magazine, IEEE*, *4*(1), 23–33.

V-CHARGE

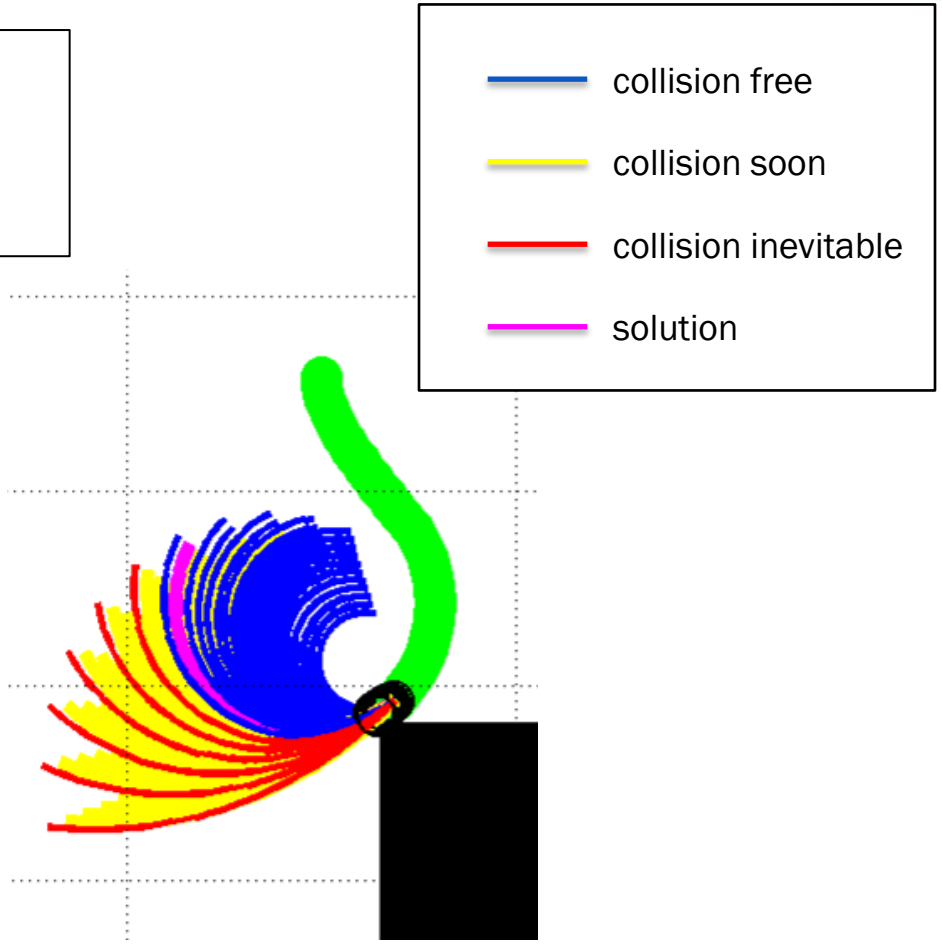
Dynamic Window Approach: Differential Drive Robot



$$G(v,\omega) = \text{smoothing\_fcn}(\alpha \cdot heading\_term(v,\omega) + \beta \cdot clearance\_term(v,\omega) + \gamma \cdot speed\_term(v,\omega))$$

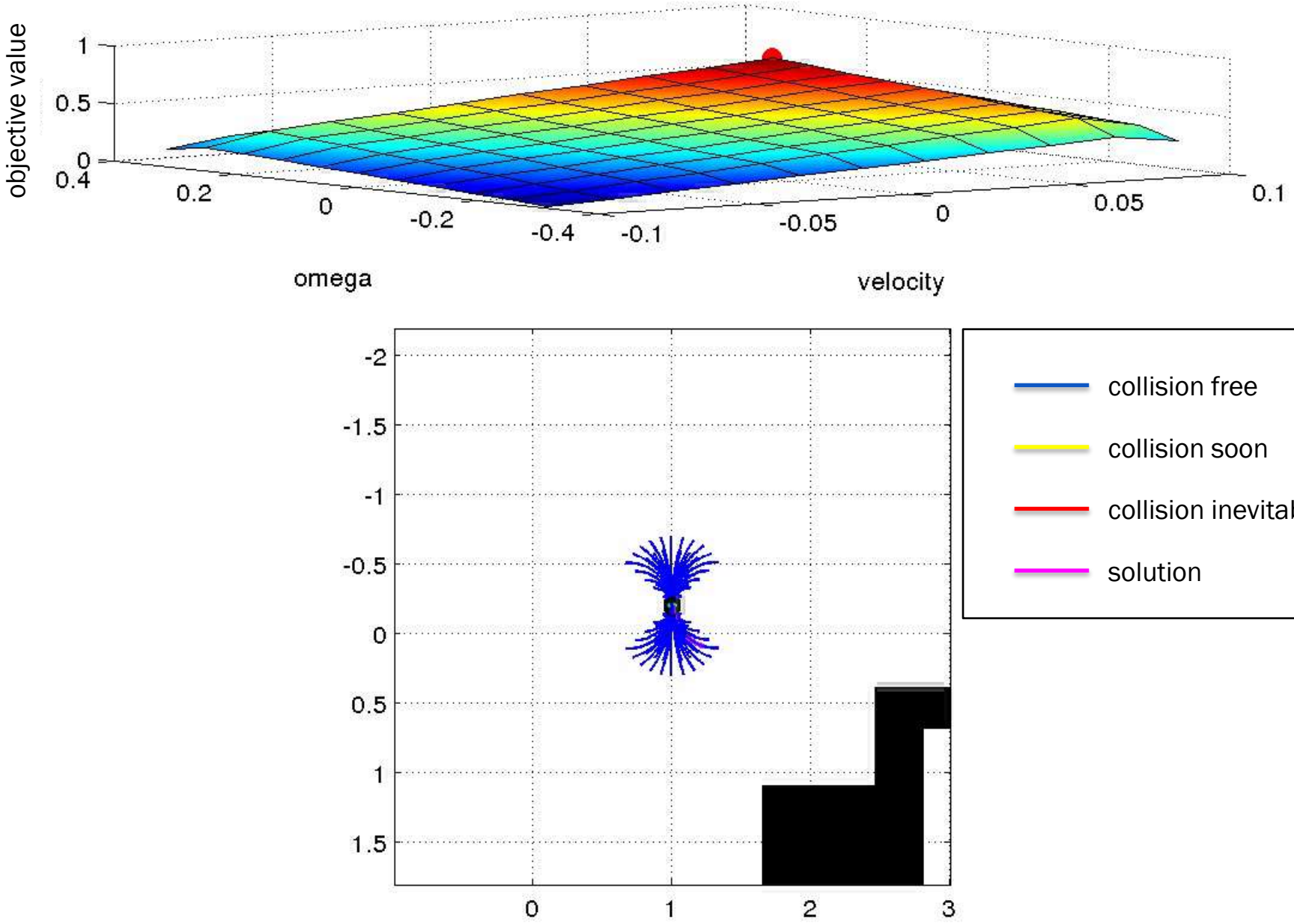
- Differential drive robot
  - inputs $u = [v, \omega]$
  - Assume change of inputs only in next timestep t_{k+1}

    →circular arcs

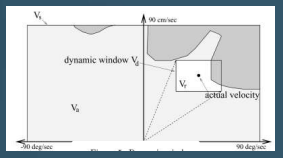


Fox, D., Burgard, W., & Thrun, S. (1997). The dynamic window approach to collision avoidance. *Robotics Automation Magazine, IEEE*, *4*(1), 23–33.

V-CHARGE


objective function
objective value
1
0.5
0
0.4
0.2
0
-0.2
-0.4
omega
-0.1
-0.05
0
0.05
0.1
velocity
-2
-1.5
-1
-0.5
0
0.5
1
1.5
0
1
2
3
collision free
collision soon
collision inevitable
solution

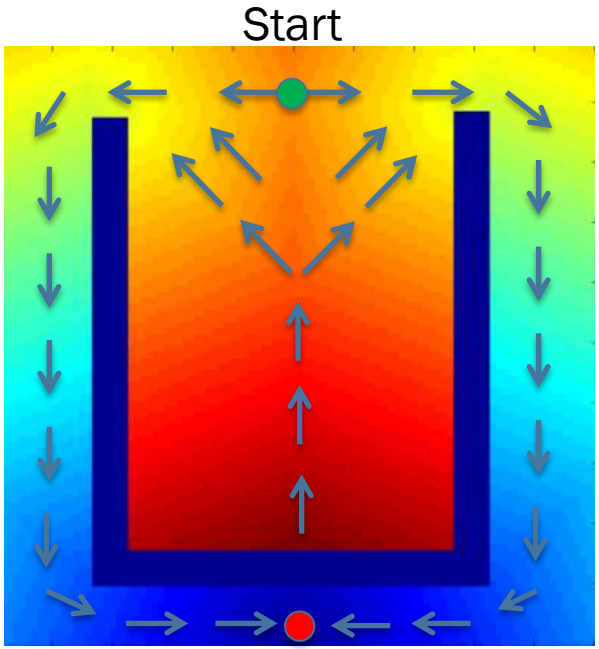
Global Dynamic Window Approach



- Original dynamic window approach is **prone to local minima**

→ replace heading term with alignment with **minima-free navigation function** (next chapter)

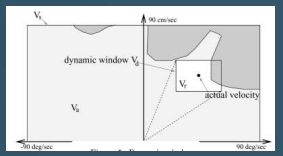
$$G(v,\omega) = \alpha \cdot heading\_term(v,\omega) + \beta \cdot clearance\_term(v,\omega) + \gamma \cdot speed\_term(v,\omega)$$



Brock, O., & Khatib, O. (1999). High-speed navigation using the global dynamic window approach. In *IEEE International Conference on Robotics and Automation*

V-CHARGE

Dynamic Window Approach: Pros & Cons



Pro	Con
• **Fast** to compute (<10ms) • Incorporates **vehicle model** • Intuitive **parameter tuning** • Global Dynamic Window approach deals well with **local minima**	• Local Dynamic Window approach can get stuck in **local minima** • Constant input assumption often **too simplistic**

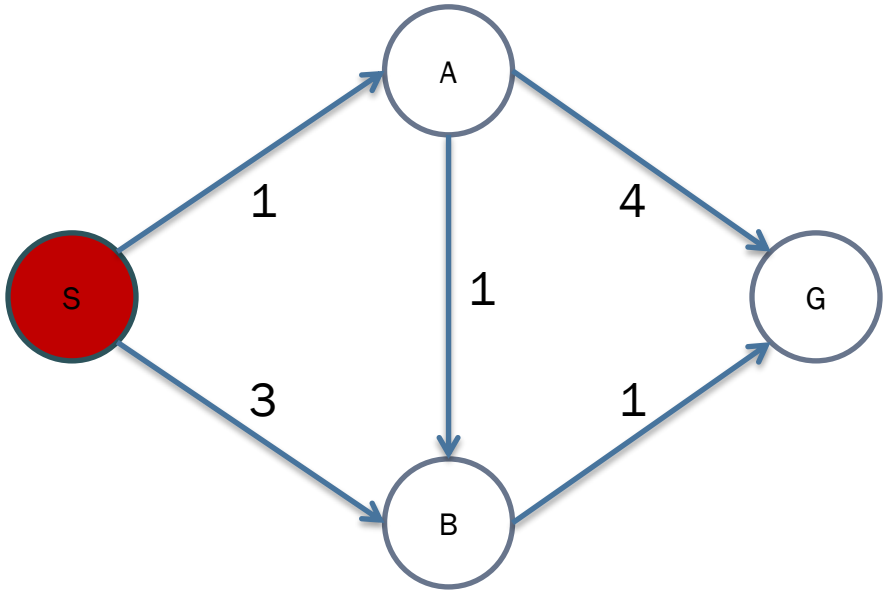
V-CHARGE

Overview

- The Basic Motion Planning Problem
  - I/O
  - Notions and Terms
- The dominant algorithms for motion planning
  - The Dynamic Window Approach
  - Graph-Search, State-Lattices
  - Randomized Approaches: RRT, RRT*
- Collision Detection
- Detailed case studies from DARPA Urban Challenge
  - Boss (Carnegie Mellon)
  - AnnieWay (Karlsruhe/Munich)
  - Junior (Stanford)
- The V-Charge Motion Planning Framework
- Getting started for the technical session

Graph Search: Shortest Path Problem

- What is the shortest path from S to G?
  - S→A→B→G, path length = 3



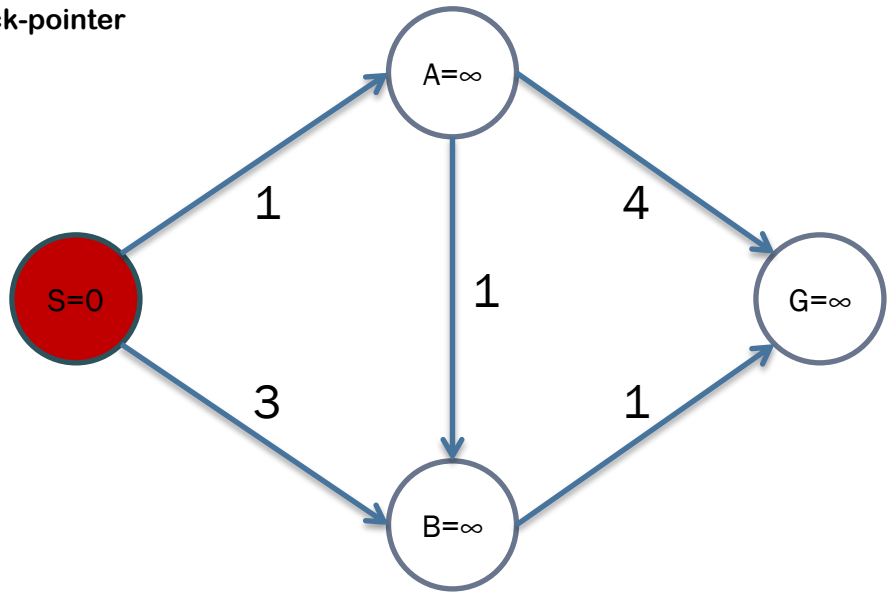
V-CHARGE

Dijkstra‘s Algorithm

- Dijstra´s algorithm
  - Pseudo-Code:
    1. **Assign cost ∞ to all vertices N but start node “s“, “s” gets cost 0**
    2. **Create sorted heap Q = {s}**
    3. while not empty Q:
       v = Q.pop_min()
       for n in neighbors(v):
       tentative_cost = cost(v) + weight(v,n)
       if tentative_cost < cost(n):
       Q.insert_or_update(n, tentative_cost)

E. Dijkstra, “A note on two problems in connexion with graphs”, *Numerische Mathematik*, 1959.

→ **Back-pointer**



Q: S=0

V-CHARGE

Dijkstra‘s Algorithm

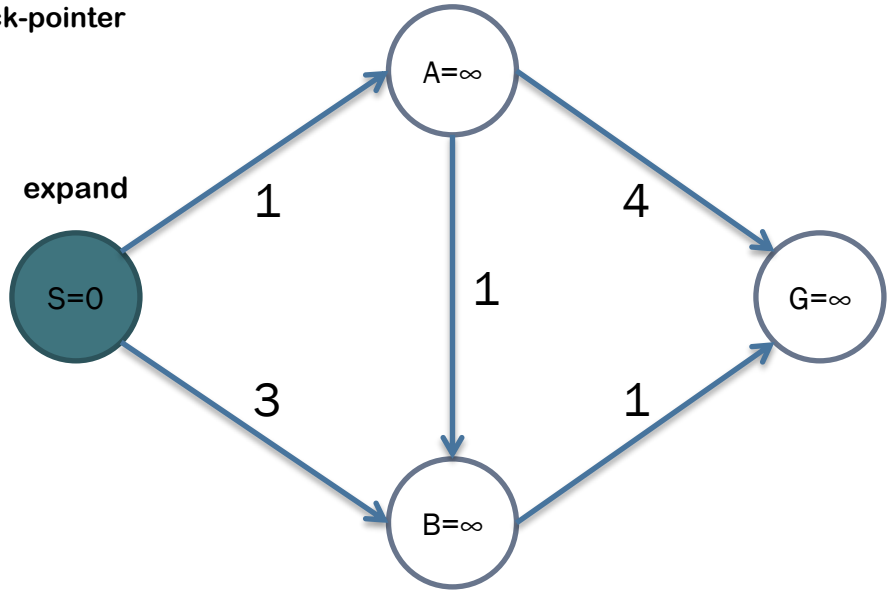
- Dijstra´s algorithm
  - Pseudo-Code:

1. **Assign cost ∞ to all vertices N but start node “s“, “s” gets cost 0**
2. **Create sorted heap Q = {s}**
3. **while not empty Q:**

```
    v = Q.pop_min()
    for n in neighbors(v):
        tentative_cost = cost(v) + weight(v,n)
        if tentative_cost < cost(n):
            Q.insert_or_update(n, tentative_cost)
```

E. Dijkstra, “A note on two problems in connexion with graphs”, *Numerische Mathematik*, 1959.

→ **Back-pointer**



Q:

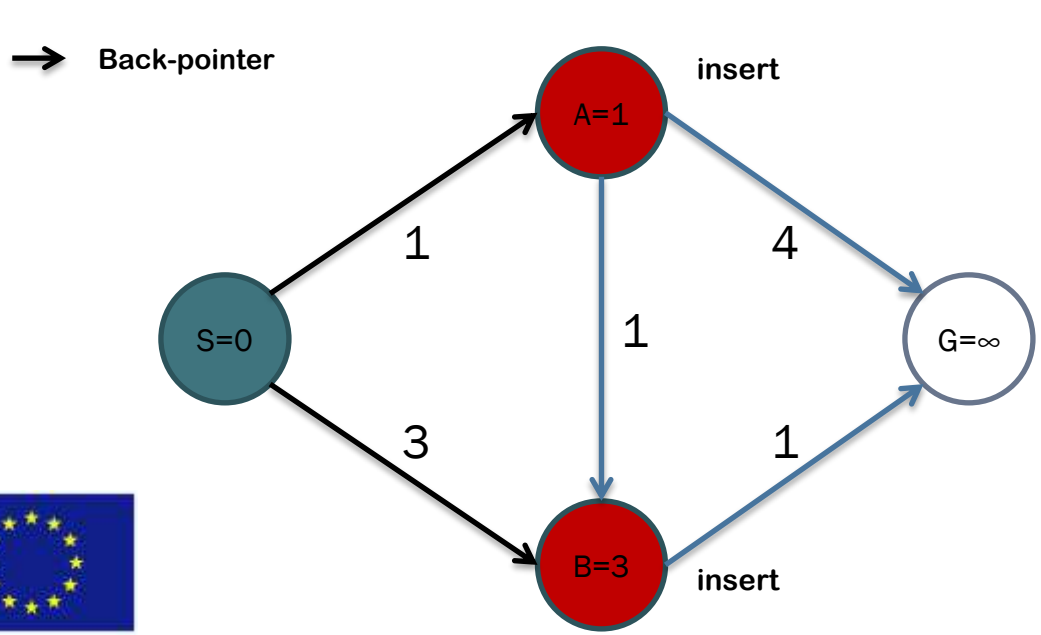
V-CHARGE

Dijkstra‘s Algorithm

- Dijstra´s algorithm
  - Pseudo-Code:

1. **Assign cost ∞ to all vertices N but start node “s“, “s” gets cost 0**
2. **Create sorted heap Q = {s}**
3. **while not empty Q:**
   v = Q.pop_min()
   for n in neighbors(v):
   tentative_cost = cost(v) + weight(v,n)
   if tentative_cost < cost(n):
   Q.insert_or_update(n, tentative_cost)

E. Dijkstra, “A note on two problems in connexion with graphs”, *Numerische Mathematik*, 1959.



Q: A=1 B=3

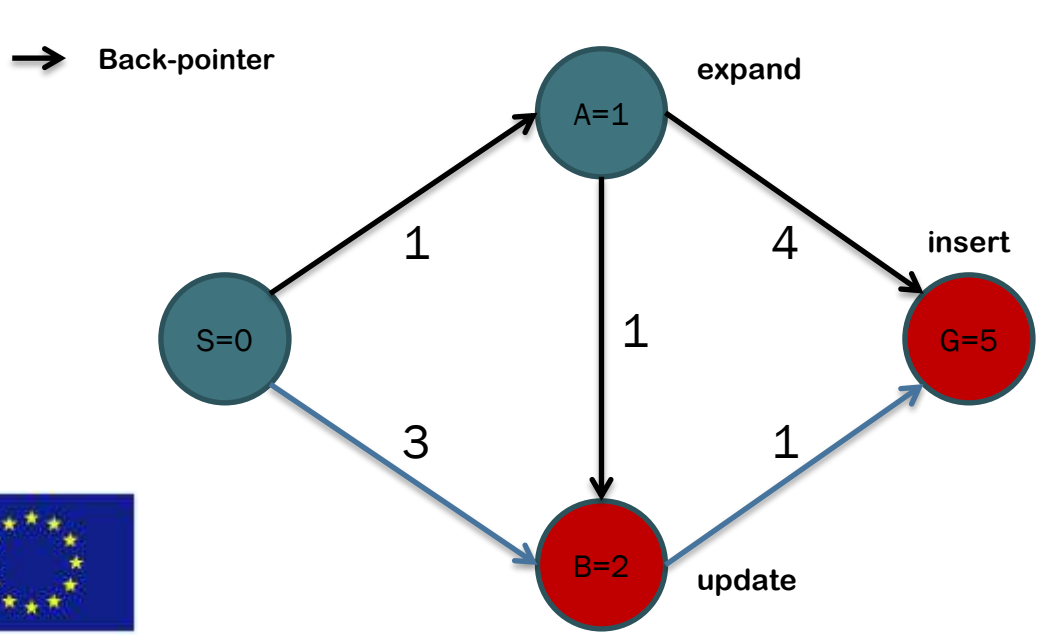
V-CHARGE

Dijkstra‘s Algorithm

- Dijstra´s algorithm
  - Pseudo-Code:

1. **Assign cost ∞ to all vertices N but start node “s“, “s” gets cost 0**
2. **Create sorted heap Q = {s}**
3. **while not empty Q:**
   v = Q.pop_min()
   for n in neighbors(v):
   tentative_cost = cost(v) + weight(v,n)
   if tentative_cost < cost(n):
   Q.insert_or_update(n, tentative_cost)

E. Dijkstra, “A note on two problems in connexion with graphs”, *Numerische Mathematik*, 1959.



Q: B=2 G=5

V-CHARGE

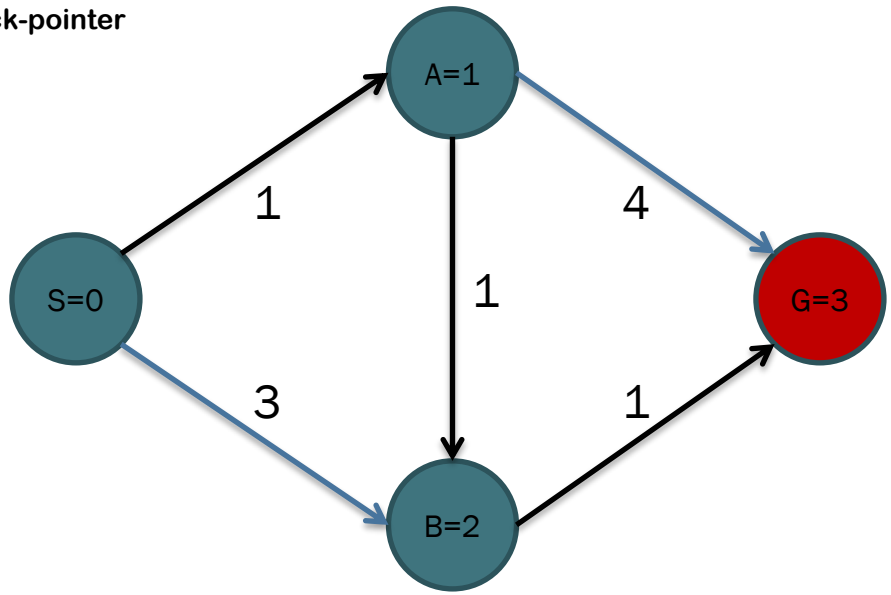
Dijkstra‘s Algorithm

- Dijstra´s algorithm
  - Pseudo-Code:

1. **Assign cost ∞ to all vertices N but start node “s“, “s” gets cost 0**
2. **Create sorted heap Q = {s}**
3. **while not empty Q:**
   v = Q.pop_min()
   for n in neighbors(v):
   tentative_cost = cost(v) + weight(v,n)
   if tentative_cost < cost(n):
   Q.insert_or_update(n, tentative_cost)

E. Dijkstra, “A note on two problems in connexion with graphs”, *Numerische Mathematik*, 1959.

Back-pointer



Q: G=3

V-CHARGE

Dijkstra‘s Algorithm

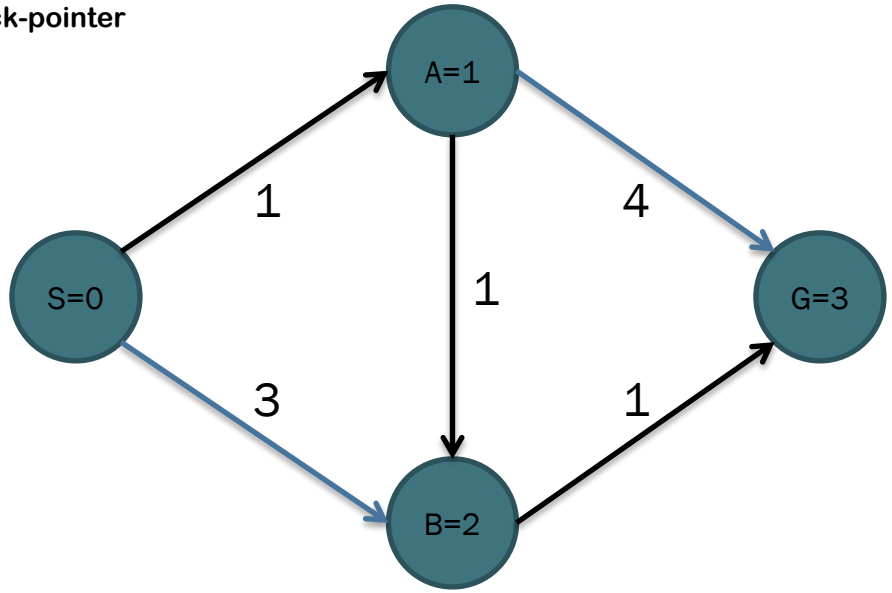
- Dijstra´s algorithm
  - Pseudo-Code:

1. **Assign cost ∞ to all vertices N but start node “s“, “s” gets cost 0**
2. **Create sorted heap Q = {s}**
3. **while not empty Q:**

```
v = Q.pop_min()
for n in neighbors(v):
        tentative_cost = cost(v) + weight(v,n)
        if tentative_cost < cost(n):
                Q.insert_or_update(n, tentative_cost)
```

E. Dijkstra, “A note on two problems in connexion with graphs”, *Numerische Mathematik*, 1959.

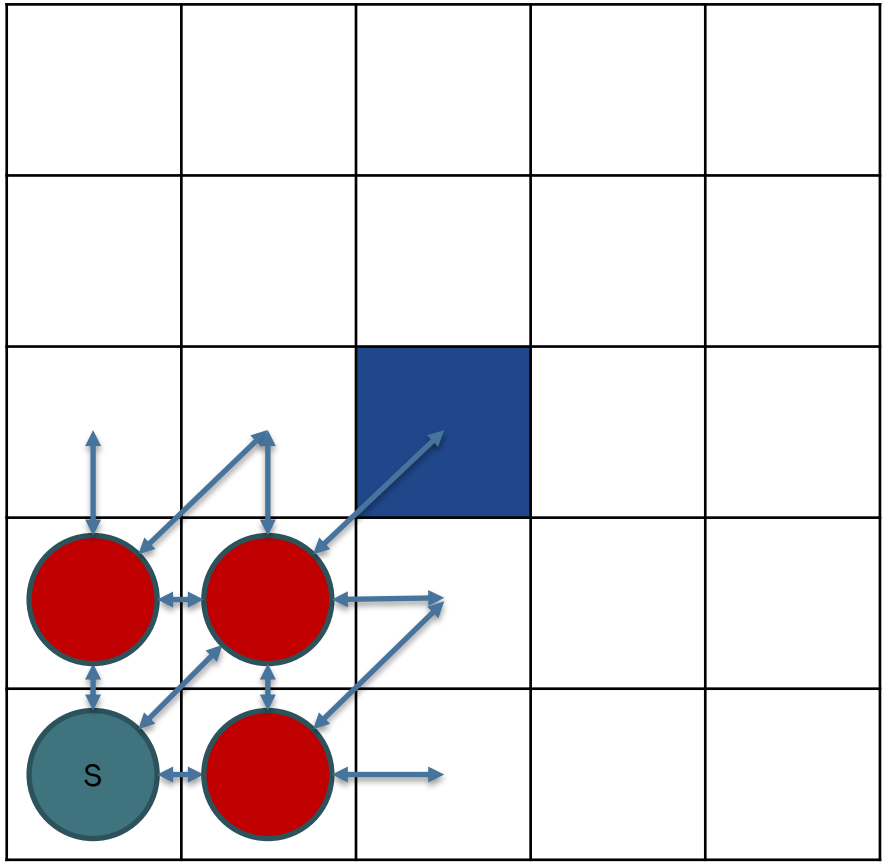
Back-pointer

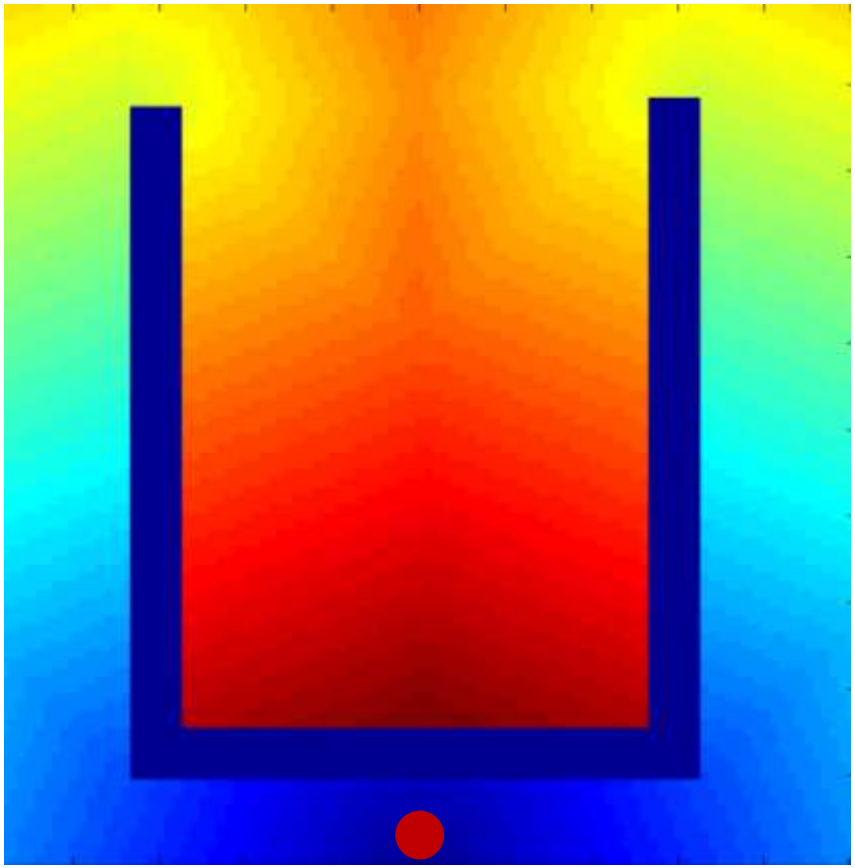


Q:

V-CHARGE

Computing Navigation Functions on 2D Grids

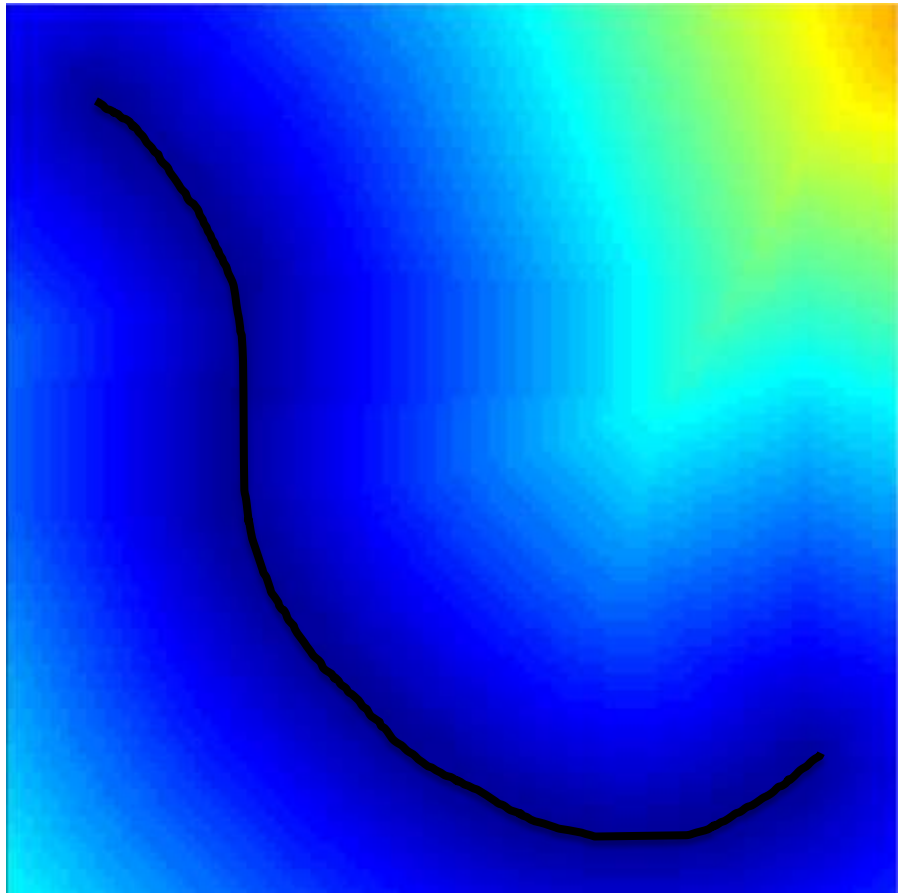




Goal

V-CHARGE

Distance Transform of Curves



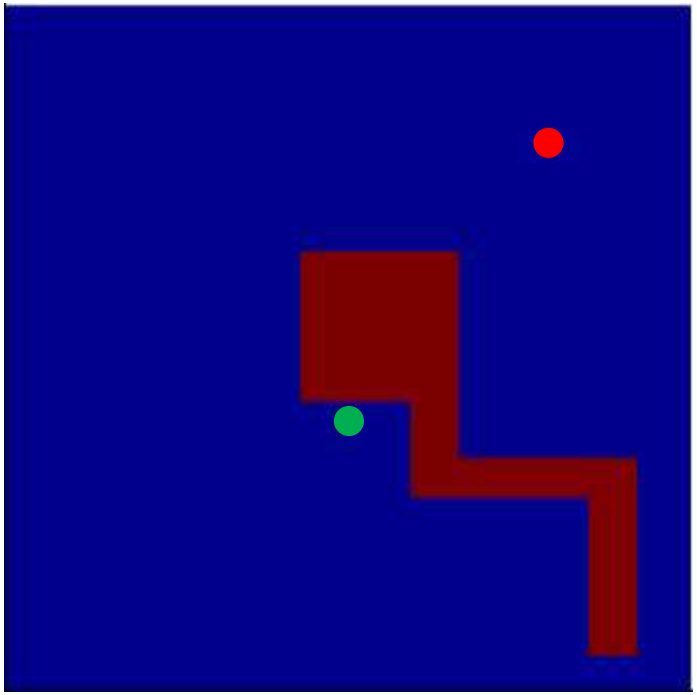
Curve distance transform

Heuristic Graph Search: A*

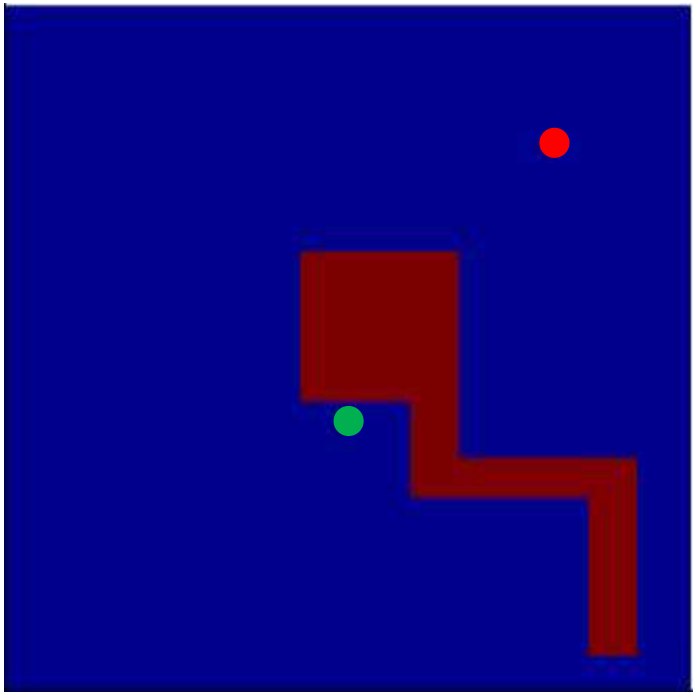
- A*: Prioritize expansion queue by expected total cost h: $f(v) = c(S, v) + h(v, G)$
  - Expand vertices with lowest expected total cost first
- Requirements on heuristic
  - Admissible (not over-estimating): $h(v, G) \leq c(v, G) \; \forall \; v$
    - e.g. do not use Manhattan distance as heuristic for an 8-connected graph on a 2D grid
  - Monotone (triangle inequality): $h(v, G) \leq c(v, n) + h(n, G) \; \forall \; v, n \in children(v)$
    - „Estimated cost of reaching goal from current vertex has to be smaller or equal to the cost of moving to its child plus the expected cost from there.“
- Guarantees optimality iff requirements hold

Dijkstra vs. A*

Dijkstra



A*



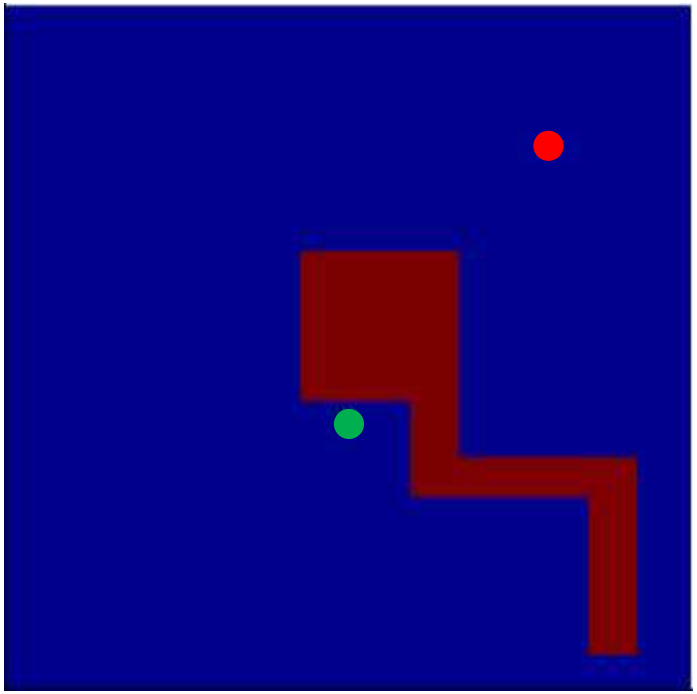
Suboptimal Heuristics in Graph Search

- Introducing sub-optimality factor $\epsilon > 1.0$:
  $\rightarrow f(v) = c(S, v) + \epsilon \cdot h(v, G)$
- Heuristic is not admissible anymore, but …
- Solution is guaranteed not to be worse than ϵ times the optimal solution!

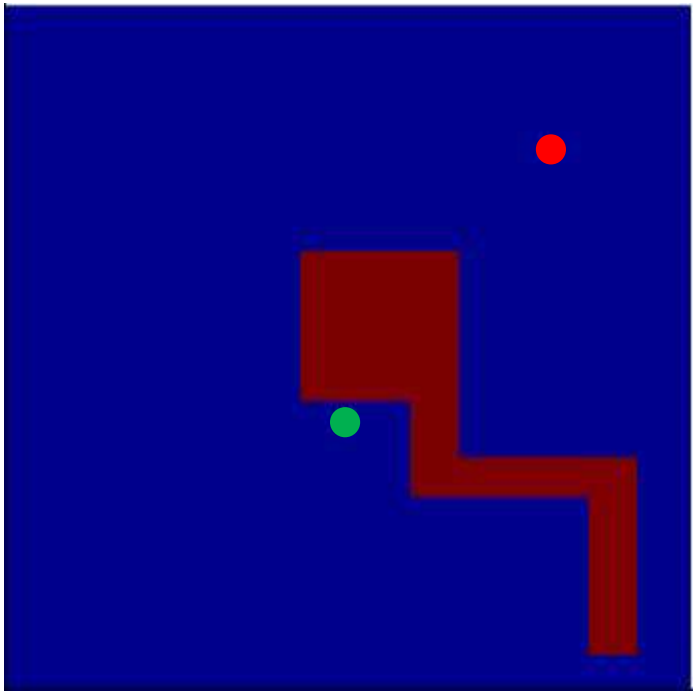
J. Pearl., "Heuristics: Intelligent Search Strategies for Computer Problem Solving", *Addison-Wesley*, 1984

Suboptimal Heuristics in Graph Search

$\epsilon = 1$ (A*)



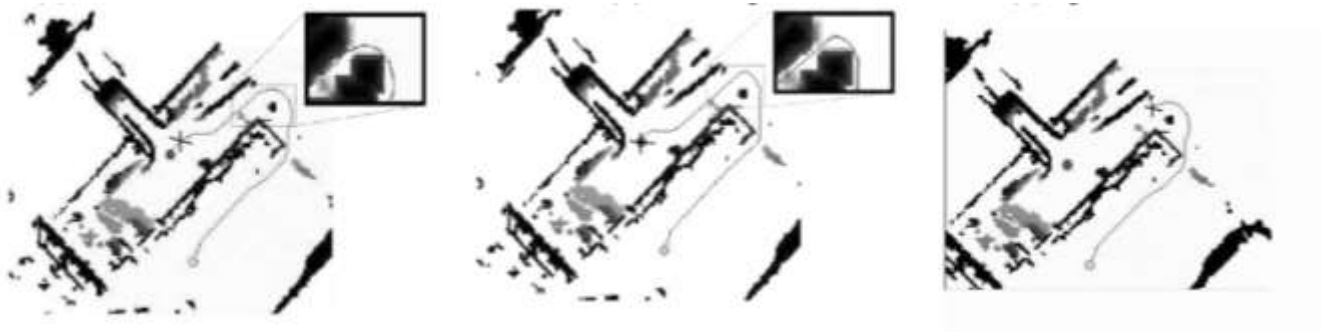
$\epsilon = 10$



Anytime behavior of graph-search

- ARA* (Anytime Repairing A*)
  - Subsequent queries with decreasing suboptimality factor ϵ
    - Fast initial (suboptimal) solution
    - Refinement over time

Example: 4D state-lattice search:



A*
25 sec

ARA* ($\epsilon = 2.5$)
0.6 sec

ARA* ($\epsilon = 1.0$)
25 sec

Likhachev, M. (2003). "ARA*: Anytime A* with provable bounds on sub-optimality", *Advances in Neural Information Processing Systems*.

Other graph-search variants

- D*/D*-Light

  Stentz, Anthony, “The Focussed D* Algorithm for Real-time Replanning”, *Proceedings of the 14th International Joint Conference on Artificial Intelligence*

  - Re-use parts of the previous query and only **repair solution** locally where changes occured

- Anytime D*

  Likhachev, M., Ferguson, D., Gordon, G., Stentz, A., Thrun, S. (2005), “Anytime dynamic A*: An anytime, replanning algorithm”, *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*

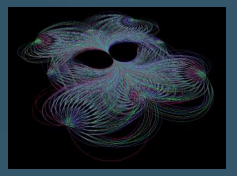
  - D* + ARA*: Anytime graph-search re-using previous query

- Field D*/E*

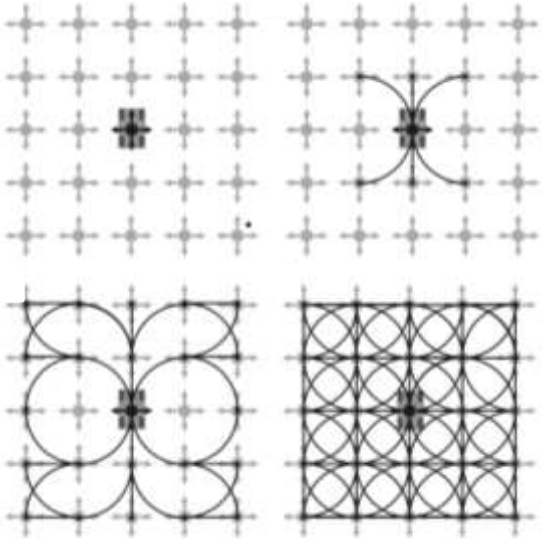
  Ferguson, D., & Stentz, A. (2005). “Field D*: An Interpolation-based Path Planner and Replanner”, *Proceedings of the International Symposium on Robotics Research (ISRR)*
  Philippsen, R. (2006), “A light formulation of the E* interpolated path replanner”, *Autonomous Systems Lab, Ecole Polytechnique Federale de Lausanne, Tech. Rep*.

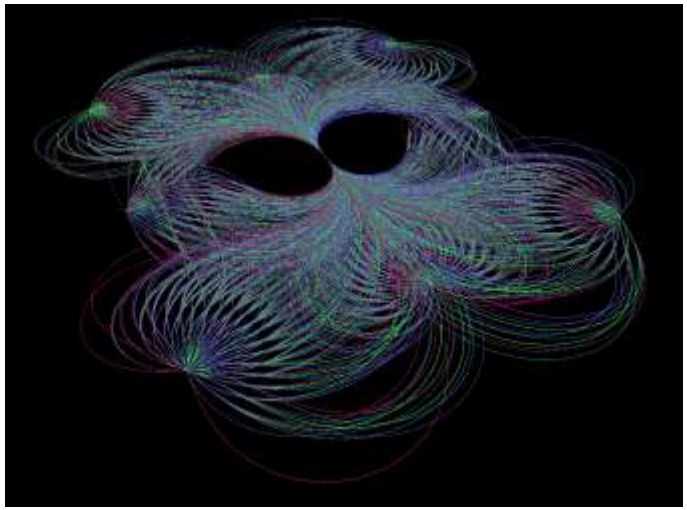
  - Interpolates the cost, smoother navigation functions

State-Lattice Search



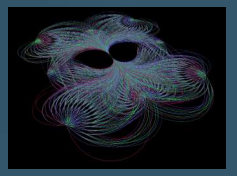
- Motion planning for contrained platforms as a graph search in state-space
  - Discretize state-space into *hypergrid (e.g.* (x, y, θ, κ)*)*
  - Compute neighborhood *set* by attempting to connect each tuple of states with feasible motions
  - Define cost-function/edge-weights
  - Run any graph-search algorithm to find lowest-cost path



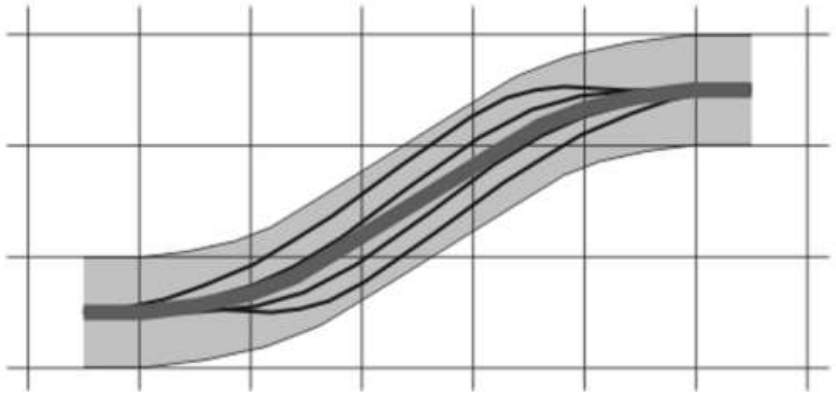


Pivtoraiko, M., & Kelly, A. (2005). Constrained Motion Planning in Discrete State Spaces. In *Field and Service Robotics* (pp. 269–280).

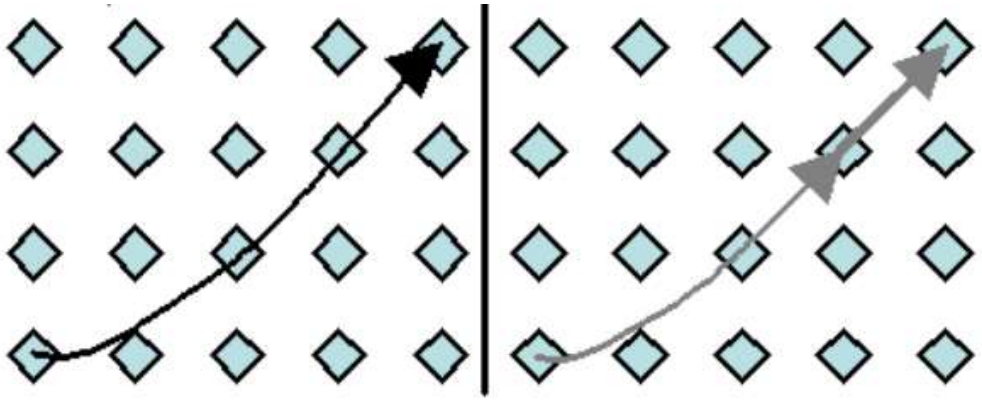
State-Lattice Search



- Obtaining minimal neighborhood sets
  - Avoid insertion of edges that can be decomposed with the existing control set
  - Decomposition has to be „close“ in cost-space



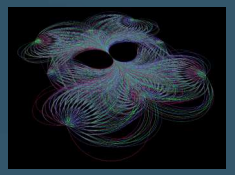
Pivtoraiko, M., & Kelly, A. (2005). Constrained Motion Planning in Discrete State Spaces. In *Field and Service Robotics* (pp. 269–280).



Pivtoraiko, M., & Kelly, A. (2005). Constrained Motion Planning in Discrete State Spaces. In *Field and Service Robotics* (pp. 269–280).

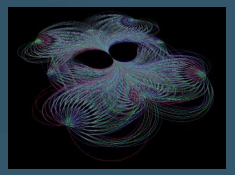
V-CHARGE

State-Lattices: Pros & Cons



Pro	Con
• Resolution **complete** • **Optimal** • **Offline computations** due to regular structure possible	• "Curse of dimensionality". Number of states grows exponentially with dimensionality of state-space → **computational complexity** in large state-spaces • State-lattice construction requires **solving nontrivial two-state boundary value problem** • Regular **discretization** might cause problems in narrow passages, not aligned with the hypergrid • Discretization causes **discontinuities** in state variables not considered in the hypergrid → motion plans not inherently executable

State-Space Discretization: Example



Example: Discretization of the state-space for an Ackermann vehicle

x = [-50.0, 50.0], 0.2 m resolution → 501 states
y = [-50.0, 50.0], 0.2 m resolution → 501 states
heading θ = [$-\pi, +\pi$], 0.1 rad resolution → ~64 states
steering angle ϕ = [-0.6, +0.6], 0.1 rad resolution → 13 states

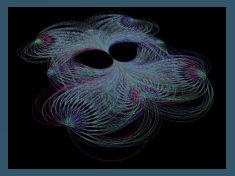
→ $|S| = |x| \times |y| \times |\theta| \times |\phi| =$ 208′832′832 states

→ Impossible to search exhaustively (during online operation)

Opportunities for speed-up:

- Well-informed heuristics
- Multi-resolution state-lattices
  - sparsify outgoing edge set

Well-Informed Heuristics for State-Lattice Search



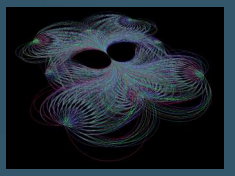
The closer the heuristics estimates the true cost-to-go, the faster the search

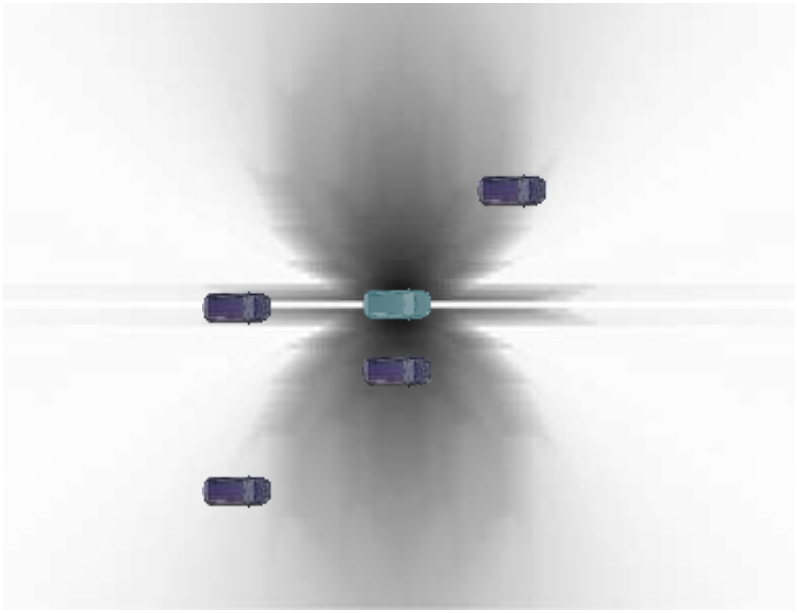
Maximum of two admissible heuristics is also admissible!

- h_1: *non-holonomic-without-obstacles* (expensive, but can be computed offline)
- h_2: *holonomic-with-obstacles* (cheap to compute/online)

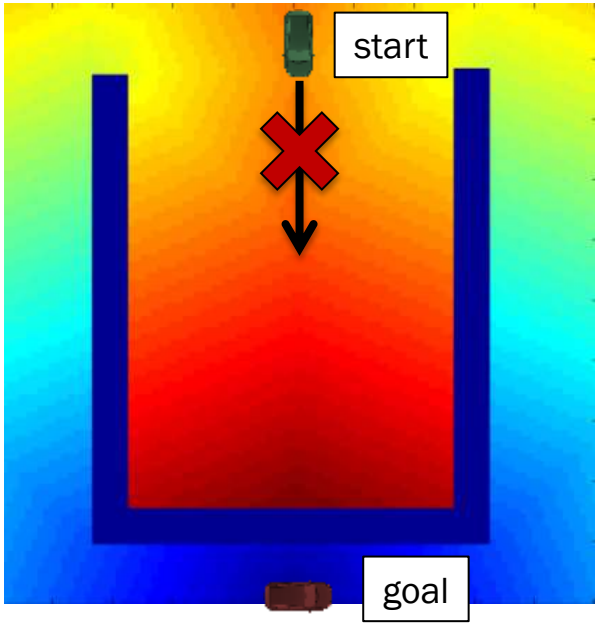
→ $h = \max(h_1, h_2)$

Well-Informed Heuristics for State-Lattice Search





Knepper, R., & Kelly, A. (2006). High Performance State Lattice Planning Using Heuristic Look-Up Tables. *Intelligent Robots and Systems (IROS)*.

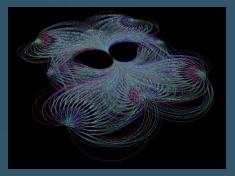


- Nonholonomic-without obstacles
  - Better estimate of true cost in *close vicinity* to goal
- Holonomic-with-obstacles
  - Better estimate of true cost *far* from goal

Dolgov, D., et al., "Path Planning for Autonomous Vehicles in Unknown Semi-structured Environments", *The International Journal of Robotics Research*, 2010

V-CHARGE

State-lattices: Two-State boundary value problem



- Control set generation for state-lattice search requires to obtain motion primitives connect two states in state space **exactly**
  - Define as optimization problem

$$u^* = \underset{\boldsymbol{u}}{\operatorname{argmin}} \left(\Phi(\boldsymbol{u}, t_f) + \int_{t_o}^{t_f} L(\boldsymbol{x}, \boldsymbol{u}, t) dt \right)$$

$$\text{subject to:}$$
$$\boldsymbol{x}(t_0) = \boldsymbol{x}_0$$
$$\boldsymbol{x}(t_f) = \boldsymbol{x}_f$$

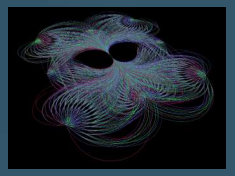
  - Convert functional optimization problem into parametric one by restriction to a parametrized function of the inputs

$$p^* = \underset{\boldsymbol{p}}{\operatorname{argmin}} \left(\Phi(\boldsymbol{u}(\boldsymbol{p}), t_f) + \int_{t_o}^{t_f} L(\boldsymbol{x}, \boldsymbol{u}(\boldsymbol{p}), t) dt \right)$$

$$\text{subject to:}$$
$$\boldsymbol{x}(t_0) = \boldsymbol{x}_0$$
$$\boldsymbol{x}(t_f) = \boldsymbol{x}_f$$

Kelly, A., Nagy, B., "Reactive Nonholonomic Trajectory Generation via Parametric Optimal Control", *The International Journal of Robotics Research*, 2003

State-lattices: Two-State boundary value problem



- Control set generation for state-lattice search requires to obtain motion primitives connect two states in state space **exactly**
  - Often convenient to parametrize problem over path length instead of time

$$p^* = \underset{\boldsymbol{p}}{\operatorname{argmin}} \left(\Phi\big(\boldsymbol{u}(\boldsymbol{p}, s_f), s_f\big) + \int_0^{s_f} L(\boldsymbol{x}, \boldsymbol{u}(\boldsymbol{p}), s) ds \right)$$

$$\text{subject to:}$$

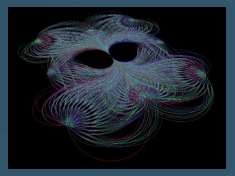
$$\boldsymbol{x}(s_0) = \boldsymbol{x}_0$$

$$\boldsymbol{x}\big(s_f\big) = \boldsymbol{x}_f$$

  - e.g. $\boldsymbol{u}(\boldsymbol{p}, s) = \kappa(\boldsymbol{p}, s) = \boldsymbol{p}_0 + \boldsymbol{p}_1 s + \boldsymbol{p}_2 s^2 + \cdots$
    - Polynomial curvature input
  - Design variables for optimization problem $\boldsymbol{q} = \big[\boldsymbol{p}, s_f\big]^T$

Kelly, A., Nagy, B., "Reactive Nonholonomic Trajectory Generation via Parametric Optimal Control", *The International Journal of Robotics Research*, 2003

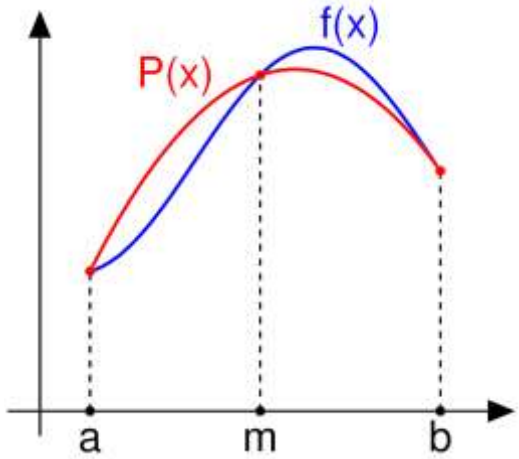
State-lattices: Two-State boundary value problem



- Control set generation for state-lattice search requires to obtain motion primitives connect two states in state space **exactly**
  - Problem: cartesian coordinates $\mathrm{x}(s)$, $\mathrm{y}(s)$ cannot be computed in closed-form for an Ackermann system model even for simple inputs

$$x(s) = \int_0^s \cos\big(\theta(\tau)\big)\, d\tau$$
$$y(s) = \int_0^s \sin\big(\theta(\tau)\big)\, d\tau$$
$$\theta(s) = \int_0^s \kappa(\tau)\, d\tau$$
$$\kappa(s) = \boldsymbol{p}_0 + \boldsymbol{p}_1 s + \boldsymbol{p}_2 s^2 + \cdots$$

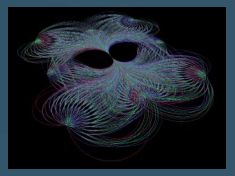
  - → Numerical approximation of integrals (e.g. Simpson‘s rule)



http://en.wikipedia.org/wiki/Simpson%27s_rule

Kelly, A., Nagy, B., “Reactive Nonholonomic Trajectory Generation via Parametric Optimal Control”, *The International Journal of Robotics Research*, 2003

State-lattices: Two-State boundary value problem



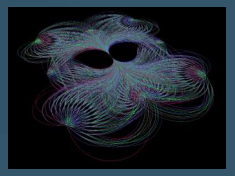
- Control set generation for state-lattice search requires to obtain motion primitives connect two states in state space **exactly**
  - For 3^{rd} order curvature polynomials, the problem has a unique solution
    - $\boldsymbol{q} = [\boldsymbol{p}_0 = \kappa_0, \boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3, s_f]^T$
    - Constraints on Δx, Δy, $\Delta\theta$, κ_0, κ_f
    - → two design variables remain
  - Unconstrained optimization (constraint satisfaction as optimization criterion)

$$\boldsymbol{p}^* = \underset{\boldsymbol{p}}{argmin} \begin{bmatrix} x_f - x(s_f) \\ y_f - y(s_f) \\ \theta_f - \theta(s_f) \\ \kappa_f - \kappa(s_f) \end{bmatrix}$$

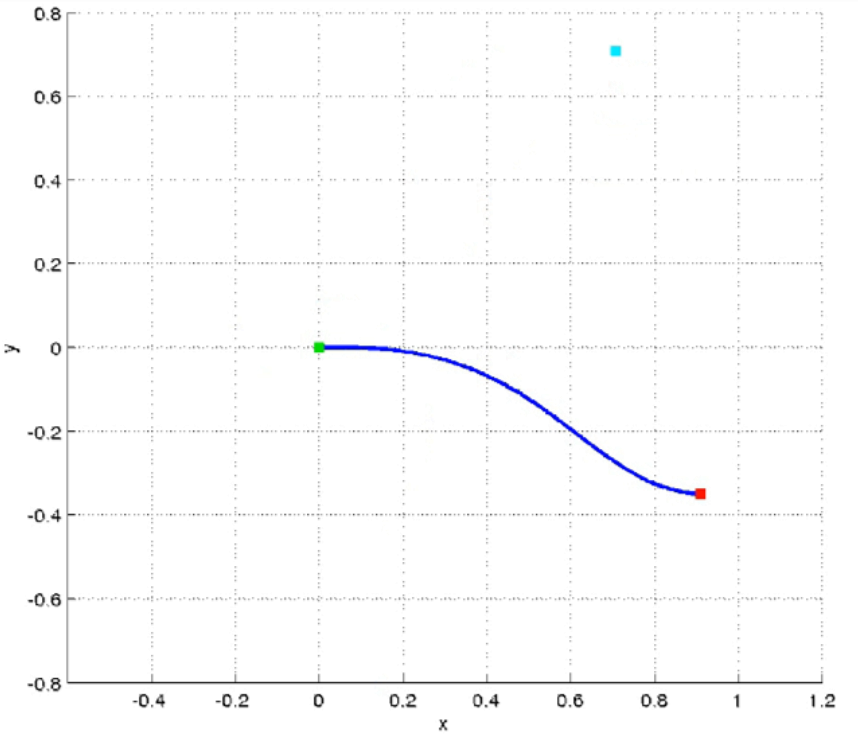
  - → Gradient-based nonlinear optimization to obtain remaining design variables

Kelly, A., Nagy, B., "Reactive Nonholonomic Trajectory Generation via Parametric Optimal Control", *The International Journal of Robotics Research*, 2003

State-lattices: Two-State boundary value problem



- Control set generation for state-lattice search requires to obtain motion primitives connect two states in state space **exactly**



initial state $[x = 0, y = 0, \theta = 0, \kappa = 0]^T$

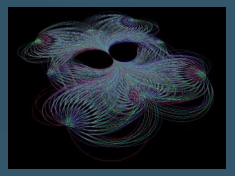
desired state $[x = 4, y = 4, \theta = 0, \kappa = 0.78]^T$

terminal state

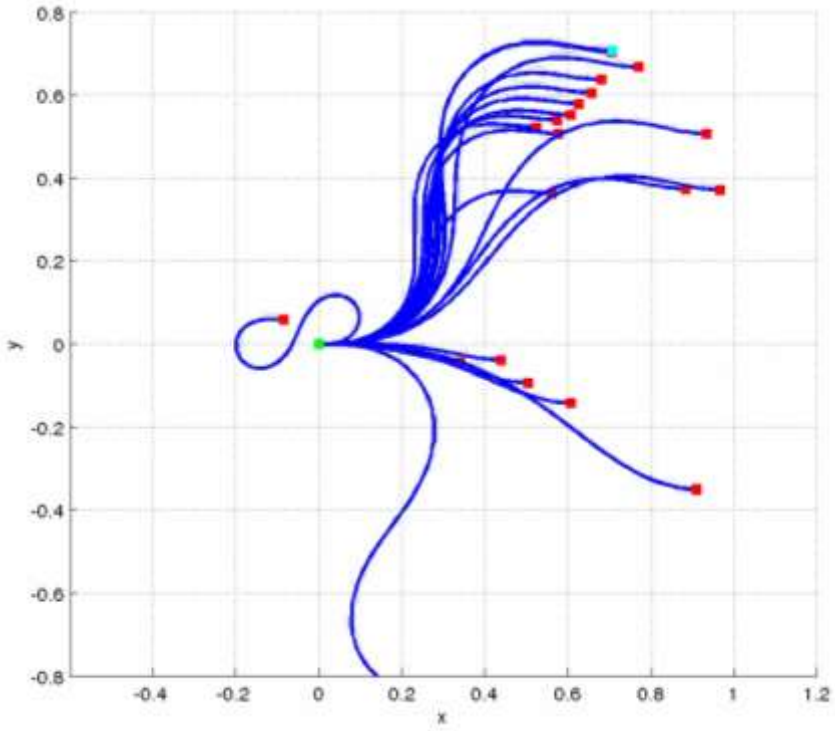
Kelly, A., Nagy, B., "Reactive Nonholonomic Trajectory Generation via Parametric Optimal Control", *The International Journal of Robotics Research*, 2003

V-CHARGE

State-lattices: Two-State boundary value problem



- Control set generation for state-lattice search requires to obtain motion primitives connect two states in state space **exactly**



initial state $[x = 0, y = 0, \theta = 0, \kappa = 0]^T$

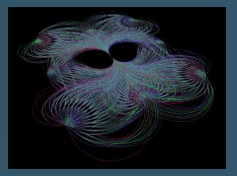
desired state $[x = 4, y = 4, \theta = 0, \kappa = 0.78]^T$

terminal state

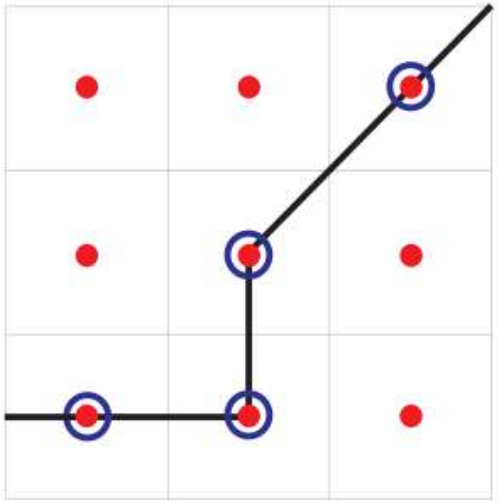
Discouraged? Try Hybrid A* …

Kelly, A., Nagy, B., "Reactive Nonholonomic Trajectory Generation via Parametric Optimal Control", *The International Journal of Robotics Research*, 2003

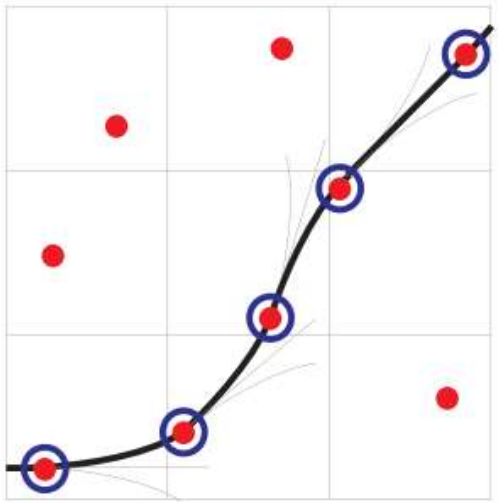
Hybrid A*



- Generate motion primitives by **sampling control space**
  - No need to solve boundary value problem
- Resulting continuous states are associated with a discrete state in the hypergrid
  - Each grid-cell stores a continuous state



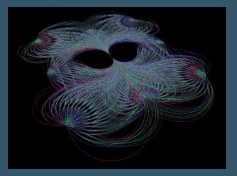
Conventional State-Lattice



Hybrid A*

Dolgov, D., et al., "Path Planning for Autonomous Vehicles in Unknown Semi-structured Environments", *The International Journal of Robotics Research*, 2010

Hybrid A*



- Generate motion primitives by **sampling control space**
  - No need to solve boundary value problem
- Resulting continuous states are associated with a discrete state in the hypergrid
  - Each grid-cell stores a continuous state

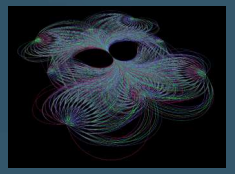
→ No completeness guarantees any more (changing reachable state-space)

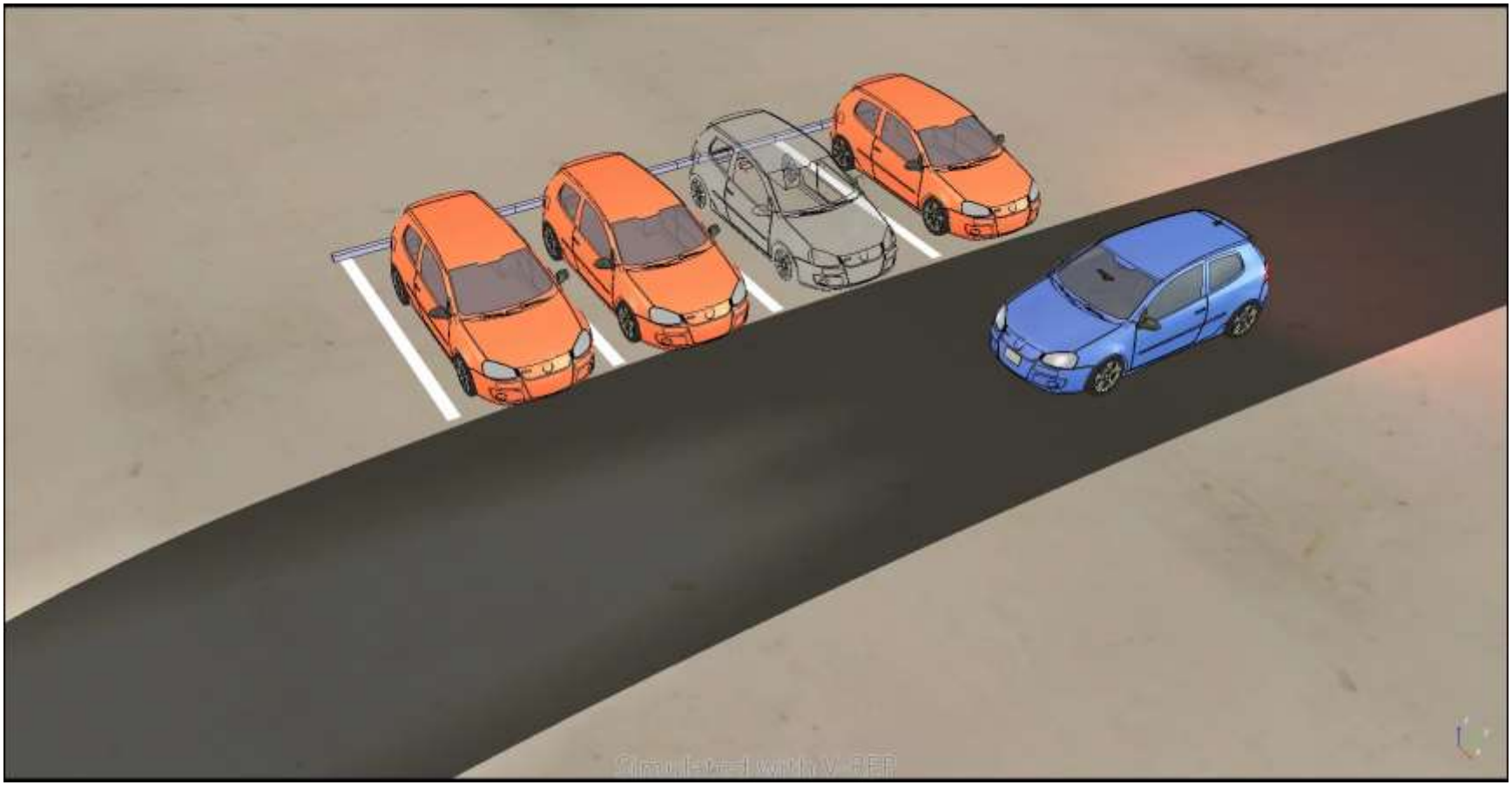
→ No optimality guarantees any more (pruning of continuous-state branches)

But: Produces inherently driveable paths and above mentioned shortcomings almost never happen in practice

Dolgov, D., et al., "Path Planning for Autonomous Vehicles in Unknown Semi-structured Environments", *The International Journal of Robotics Research*, 2010

State-Lattice Search for Parking Maneuvers



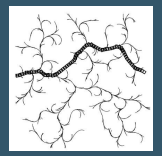


V-CHARGE

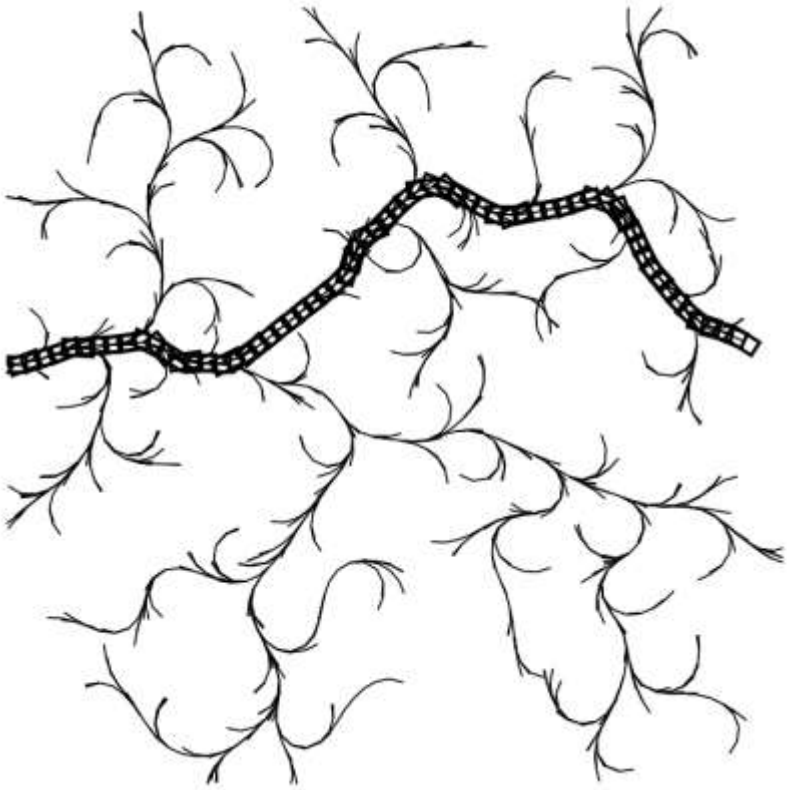
Overview

- The Basic Motion Planning Problem
  - I/O
  - Notions and Terms
- Selected algorithms for motion planning
  - The Dynamic Window Approach
  - Graph-Search, State-Lattices
  - Randomized Approaches: RRT, RRT*
- Collision Detection
- Detailed case studies from DARPA Urban Challenge
  - Boss (Carnegie Mellon)
  - AnnieWay (Karlsruhe/Munich)
  - Junior (Stanford)
- The V-Charge Motion Planning Framework
- Getting started for the technical session

Randomized Approaches



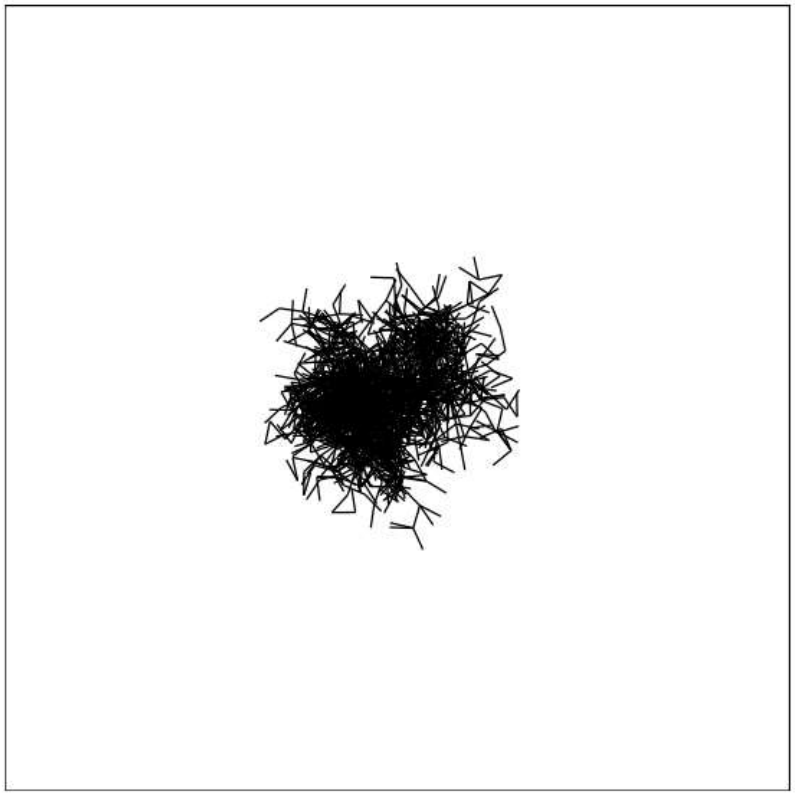
If the problem gets too hard, use an RRT



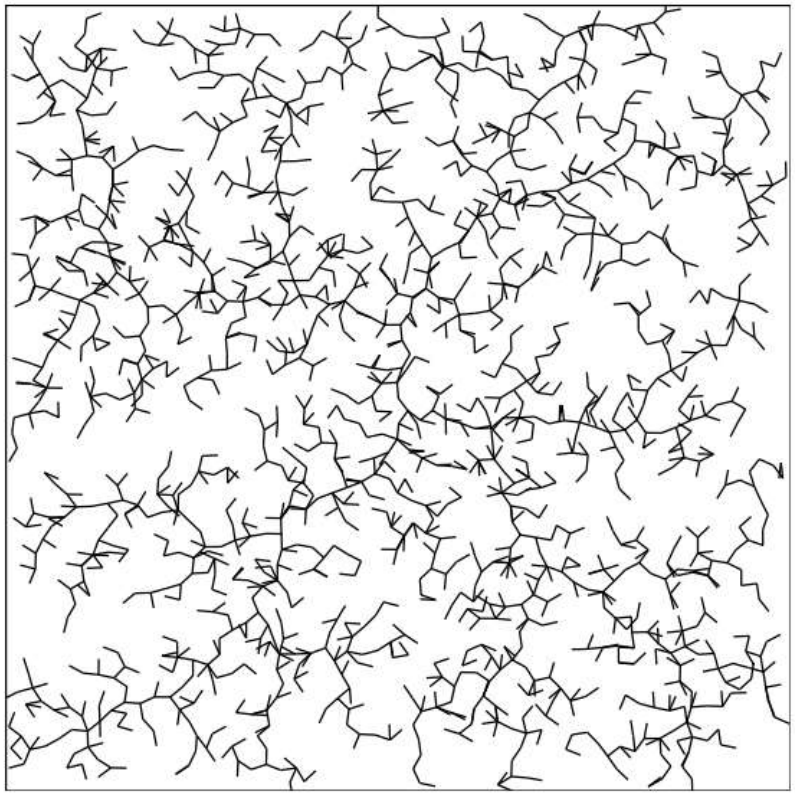
LaValle, S. (1998). Rapidly-Exploring Random Trees A New Tool for Path Planning.

V-CHARGE

Randomized Approaches: The Rapidly-Exploring Random Tree



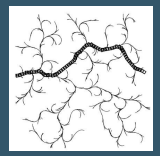
Naïve tree

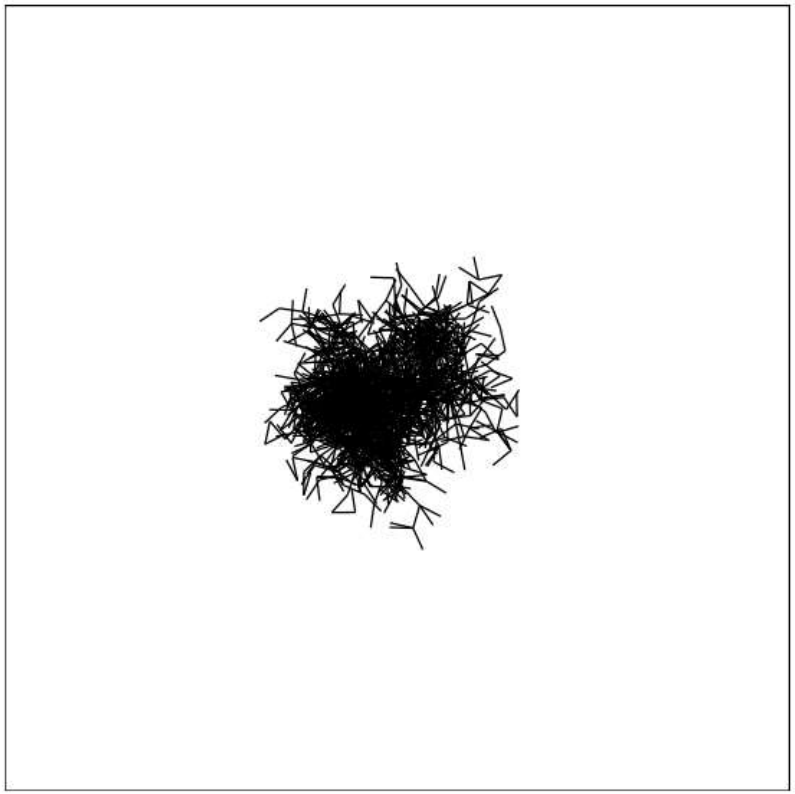


Rapidly-Exploring Random Tree

LaValle, S. M., Kuffner, J. J. J., “Randomized Kinodynamic Planning”, *IEEE International Conference on Robotics and Automatio*, 1999

Randomized Approaches: Naive Algorithm



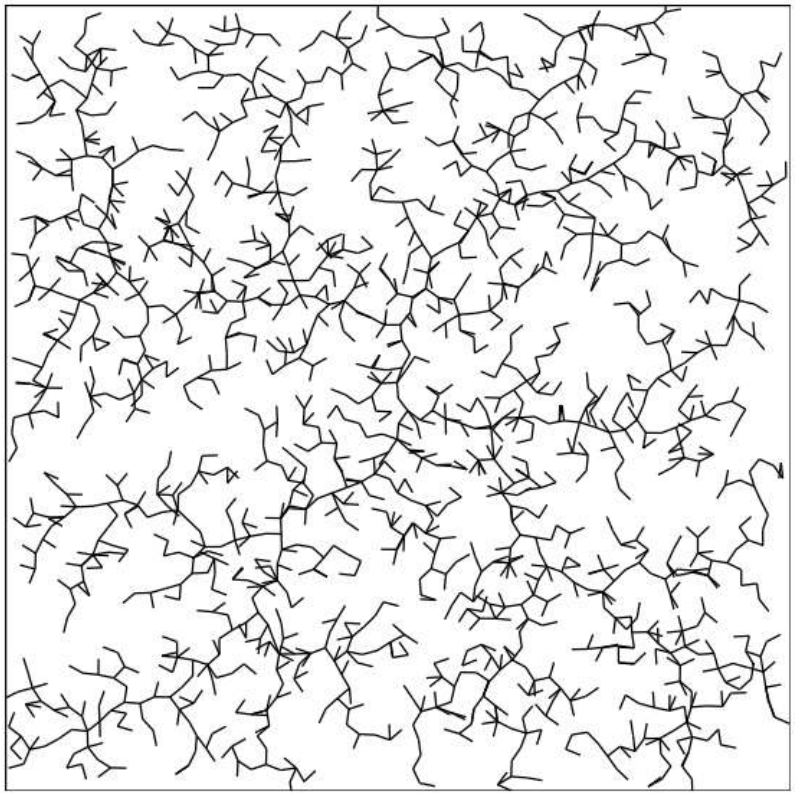


Naïve tree

- Grow tree for system $\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u})$
- Rapidly-Exploring Random Tree
  - Randomly **sample** existing state in tree $\boldsymbol{x}_{sample}$
  - Randomly sample **control input** $\boldsymbol{u}_{sample}$
  - Generate new state via $\boldsymbol{x}_{new} = \int_{t_0}^{t_f} f(\boldsymbol{x}_{sample}, \boldsymbol{u}_{sample})\, dt$

LaValle, S. M., Kuffner, J. J. J., "Randomized Kinodynamic Planning", *IEEE International Conference on Robotics and Automatio*, 1999

Rapidly-Exploring Random Tree



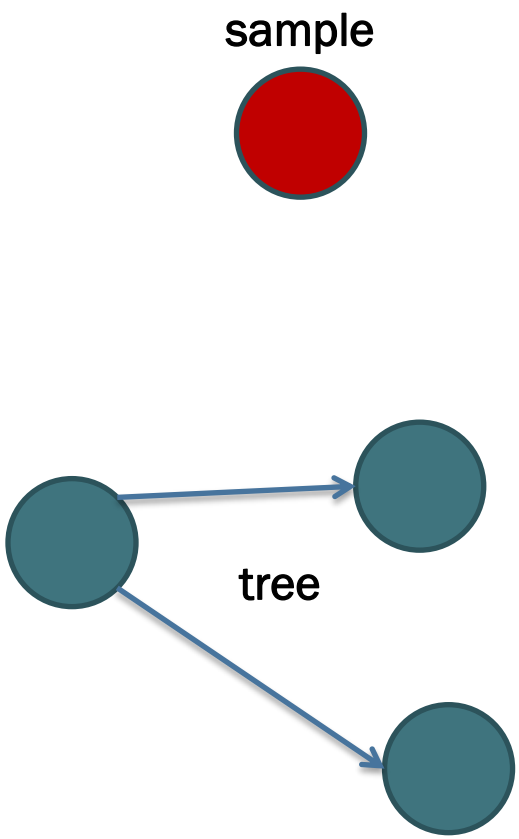
Rapidly-Exploring Random Tree

- Grow tree for system $\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u})$
- Rapidly-Exploring Random Tree
  - Randomly **sample** state $\boldsymbol{x}_{sample}$ in state space
  - Obtain **nearest** state in tree $\boldsymbol{x}_{nearest}$
  - **Steer** system from $\boldsymbol{x}_{nearest}$ towards state $\boldsymbol{x}_{sample}$, generates $\boldsymbol{x}_{new}$

→ Fast exploration of state-space

LaValle, S. M., Kuffner, J. J. J., “Randomized Kinodynamic Planning”, *IEEE International Conference on Robotics and Automatio*, 1999

Rapidly-Exploring Random Tree

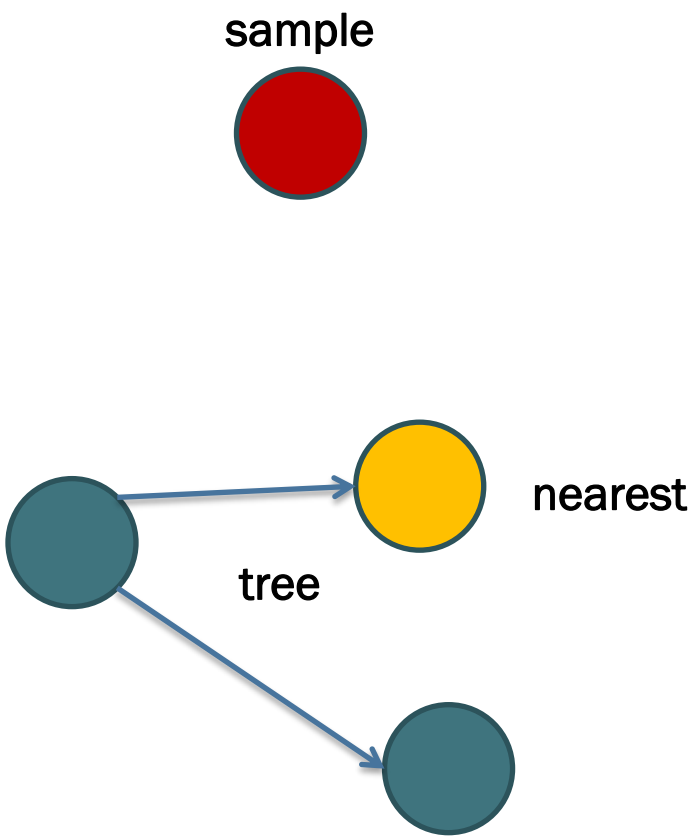


- Grow tree for system $\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u})$
- Rapidly-Exploring Random Tree
  - Randomly **sample** state $\boldsymbol{x}_{sample}$ in state-space
  - Obtain **nearest** state in tree $\boldsymbol{x}_{nearest}$
  - **Steer** system from $\boldsymbol{x}_{nearest}$ towards state $\boldsymbol{x}_{sample}$, generates $\boldsymbol{x}_{new}$

→ Fast exploration of state-space

LaValle, S. M., Kuffner, J. J. J., “Randomized Kinodynamic Planning”, *IEEE International Conference on Robotics and Automatio*, 1999

Rapidly-Exploring Random Tree

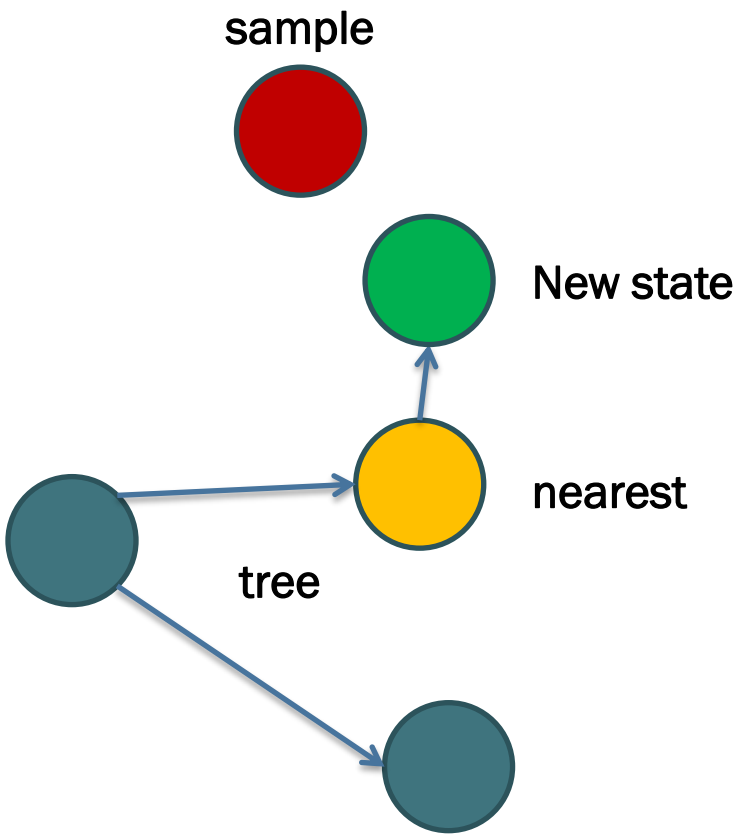


- Grow tree for system $\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u})$
- Rapidly-Exploring Random Tree
  - Randomly **sample** state $\boldsymbol{x}_{sample}$ in state-space
  - Obtain **nearest** state in tree $\boldsymbol{x}_{nearest}$
  - **Steer** system from $\boldsymbol{x}_{nearest}$ towards state $\boldsymbol{x}_{sample}$, generates $\boldsymbol{x}_{new}$

→ Fast exploration of state-space

LaValle, S. M., Kuffner, J. J. J., "Randomized Kinodynamic Planning", *IEEE International Conference on Robotics and Automatio*, 1999

Rapidly-Exploring Random Tree

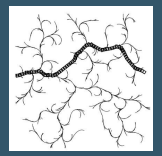


- Grow tree for system $\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u})$
- Rapidly-Exploring Random Tree
  - Randomly **sample** state $\boldsymbol{x}_{sample}$ in state-space
  - Obtain **nearest** state in tree $\boldsymbol{x}_{nearest}$
  - **Steer** system from $\boldsymbol{x}_{nearest}$ towards state $\boldsymbol{x}_{sample}$, generates $\boldsymbol{x}_{new}$

→ Fast exploration of state-space

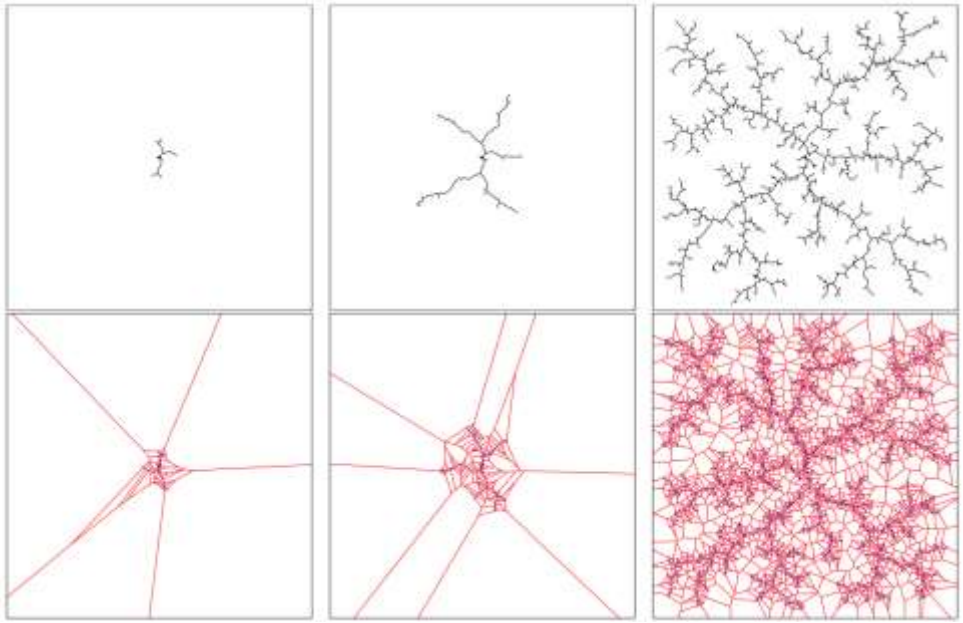
LaValle, S. M., Kuffner, J. J. J., "Randomized Kinodynamic Planning", *IEEE International Conference on Robotics and Automatio*, 1999

Rapidly-Exploring Random Tree: Voronoi Bias



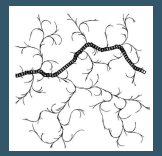
- Random sampling introduces voronoi bias
  - larger voronoi regions have larger probability of being sampled

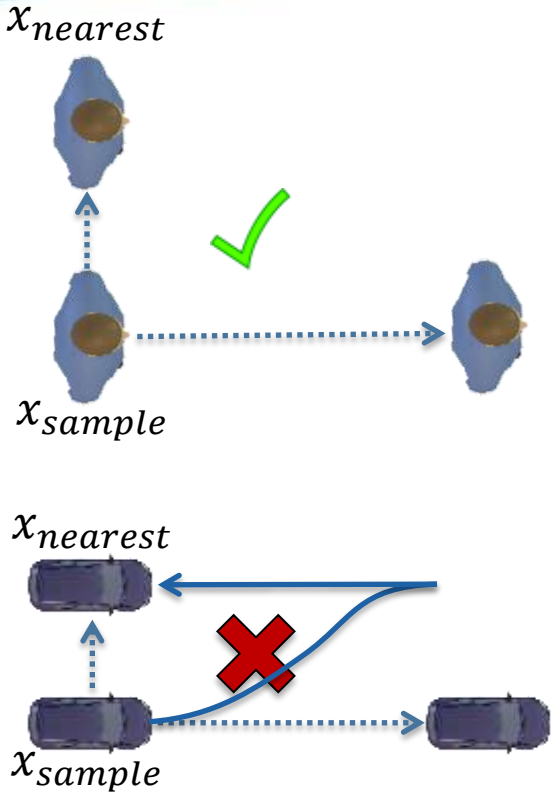
→ exploration bias towards unexplored space



LaValle, S., “From dynamic programming to RRTs: Algorithmic design of feasible trajectories”, *Control Problems in Robotics*, 2003

Rapidly-Exploring Random Tree: Distance Metric





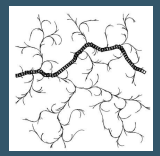
- Grow tree for system $\dot{x} = f(x, u)$
- Rapidly-Exploring Random Tree
  - Randomly **sample** state x_{sample}
  - Obtain **nearest** state in tree $x_{nearest}$
  - **Steer** system from $x_{nearest}$ towards state x_{sample}, generates x_{new}

→ Fast exploration of state-space

- RRT exploration quality is sensitive to distance metric
- Obtaining distance metrics for *non-holonomic* systems is non-trivial

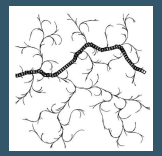
V-CHARGE

Rapidly-Exploring Random Tree: Extensions



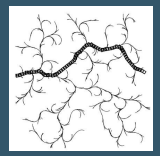
- Bidirectional RRT
  - Grows two trees from start and goal and frequently tries to merge them
- Goal-biased RRT
  - Sample goal state every n-th sample. Tradeoff between exploration and exploitation

RRT: Pros & Cons



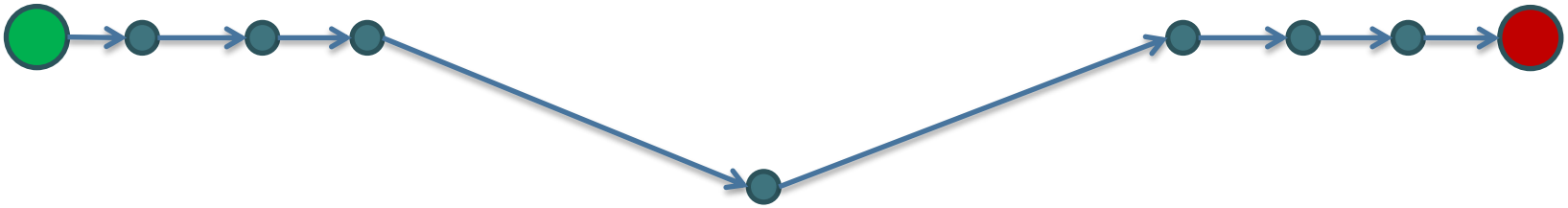
Pro	Con
• Asymptotically **complete** • Works reasonably well in **high-dimensional state-spaces** • **No two-state boundary value solver** required • **Easy implementation** • Easy to deal with **constrained platforms**	• No **optimality guarantee** (!) • Produces **“jerky” paths** in finite time • Hardly any **offline computations** possible

RRT: Why No Asymptotic Optimality Guarantee?

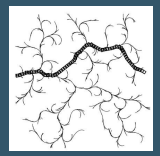


- Mathematical proof exists
  - Karaman, S., & Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, *30*(7), 846–894.
- Illustrative example
  - Search for shortest path

Tree at k=8:

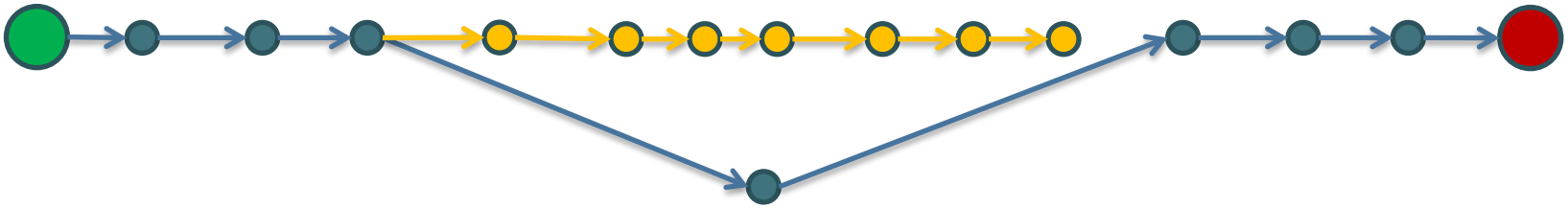


RRT: Why No Asymptotic Optimality Guarantee?

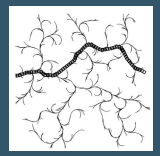


- Mathematical proof exists
  - Karaman, S., & Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, *30*(7), 846–894.
- Illustrative example
  - Search for shortest path

Tree at k=15:

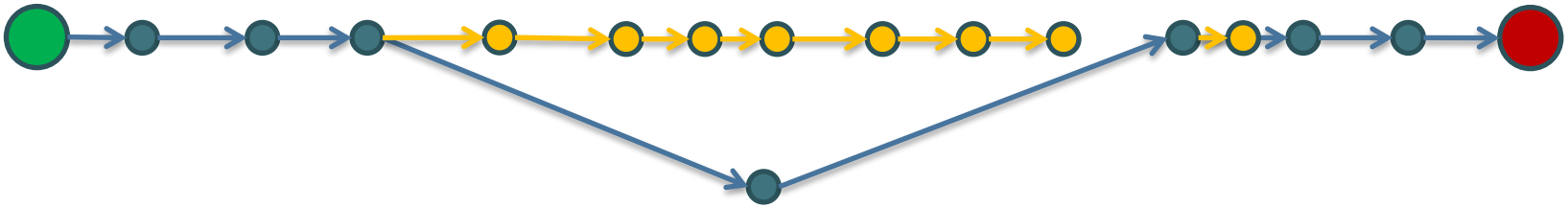


RRT: Why No Asymptotic Optimality Guarantee?

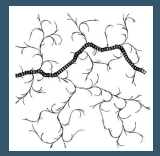


- Mathematical proof exists
  - Karaman, S., & Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, *30*(7), 846–894.
- Illustrative example
  - Search for shortest path

Tree at k=16:



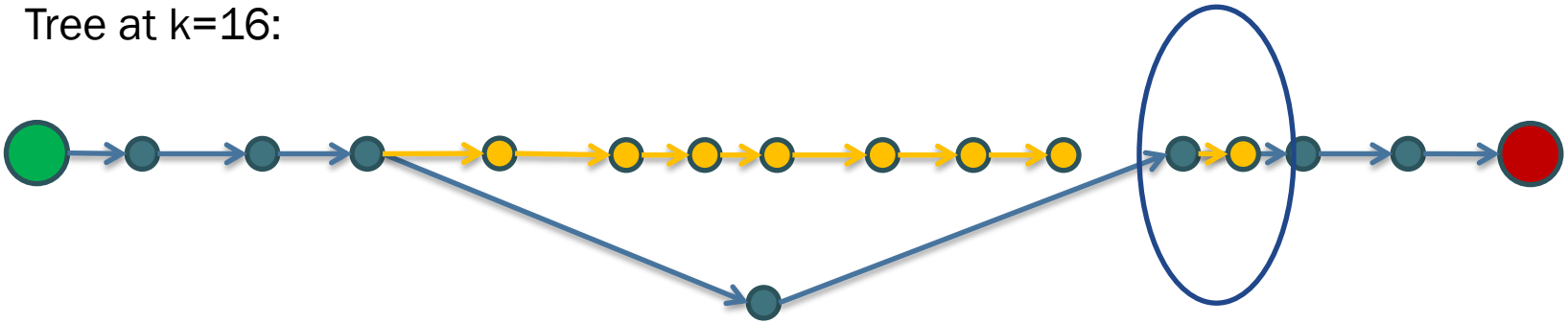
RRT: Why No Asymptotic Optimality Guarantee?



- Mathematical proof exists
  - Karaman, S., & Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, *30*(7), 846–894.
- Illustrative example
  - Search for shortest path

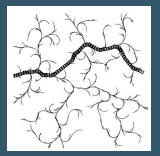
Lack of ability to "rewire" (change parent) existing vertices in the tree!

Tree at k=16:

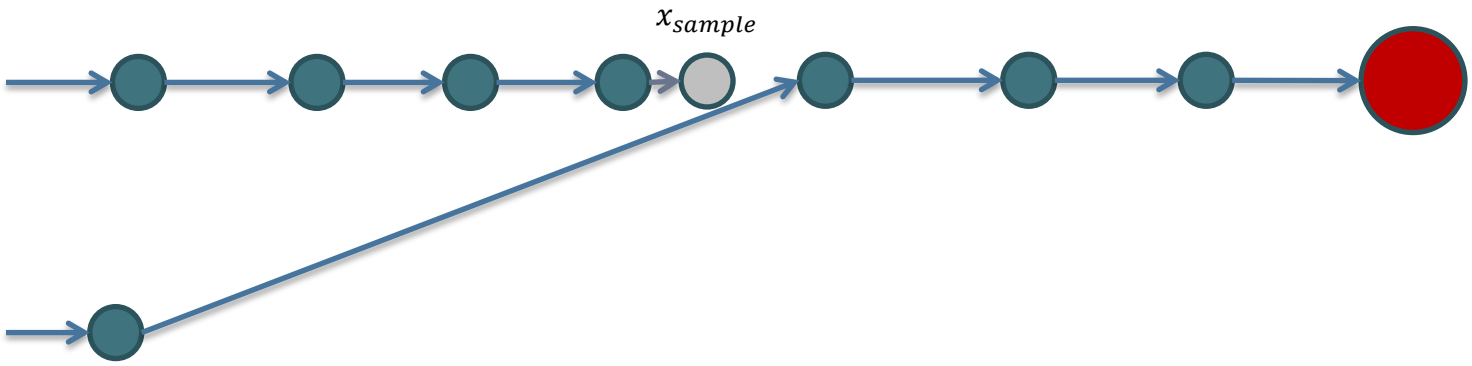


V-CHARGE

RRT*: Asymptotically Optimal RRT

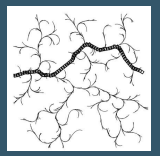


- RRT*
  - Introduces local rewiring step to obtain asymptotic optimality

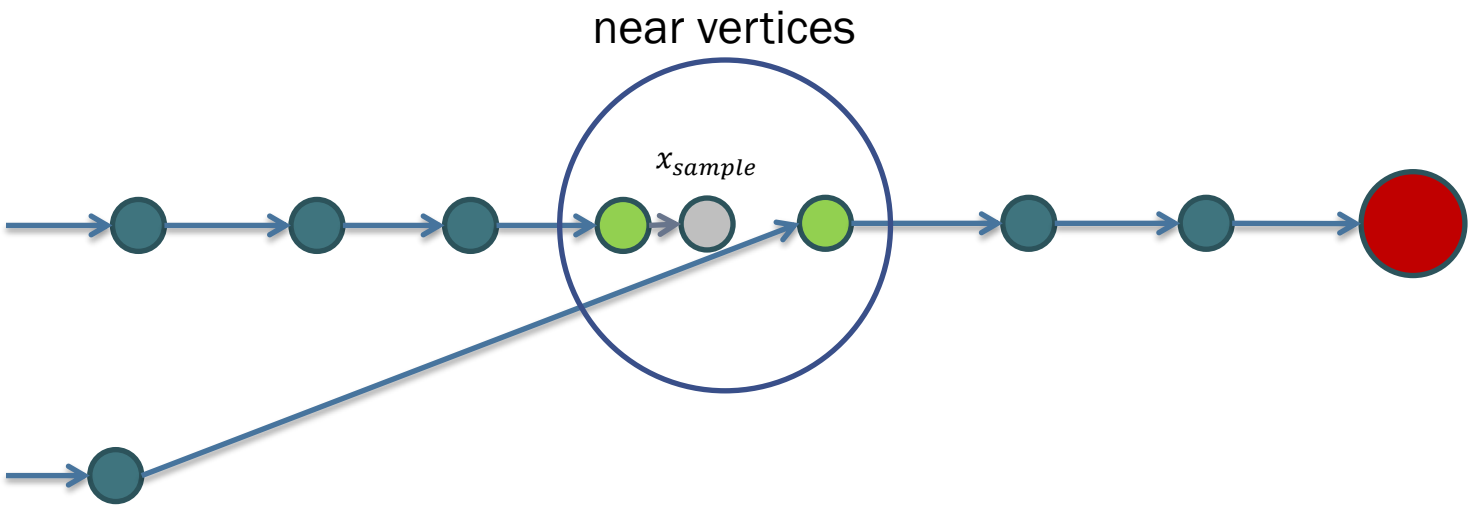


S. Karaman and E. Frazzoli, “Incremental sampling-based algorithms for optimal motion planning”, Robotics: Science and Systems, 2010

RRT*: Asymptotically Optimal RRT

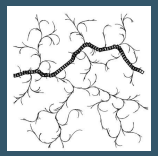


- RRT*
  - Introduces local rewiring step to obtain asymptotic optimality

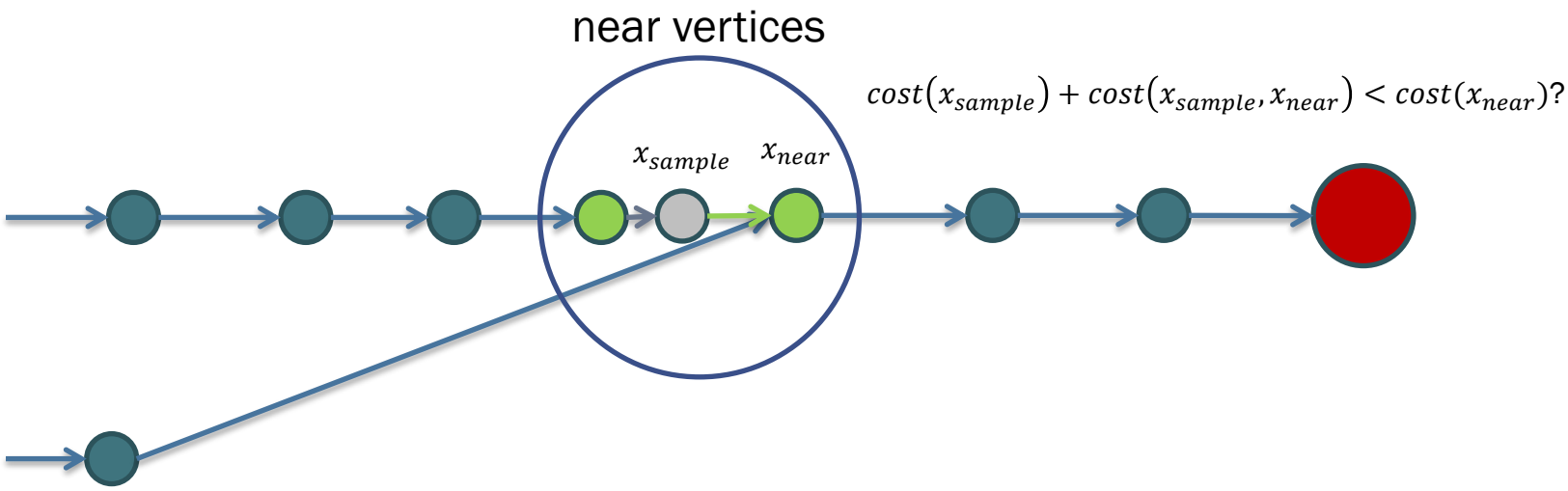


S. Karaman and E. Frazzoli, “Incremental sampling-based algorithms for optimal motion planning”, Robotics: Science and Systems, 2010

RRT*: Asymptotically Optimal RRT



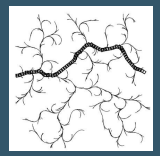
- RRT*
  - Introduces local rewiring step to obtain asymptotic optimality



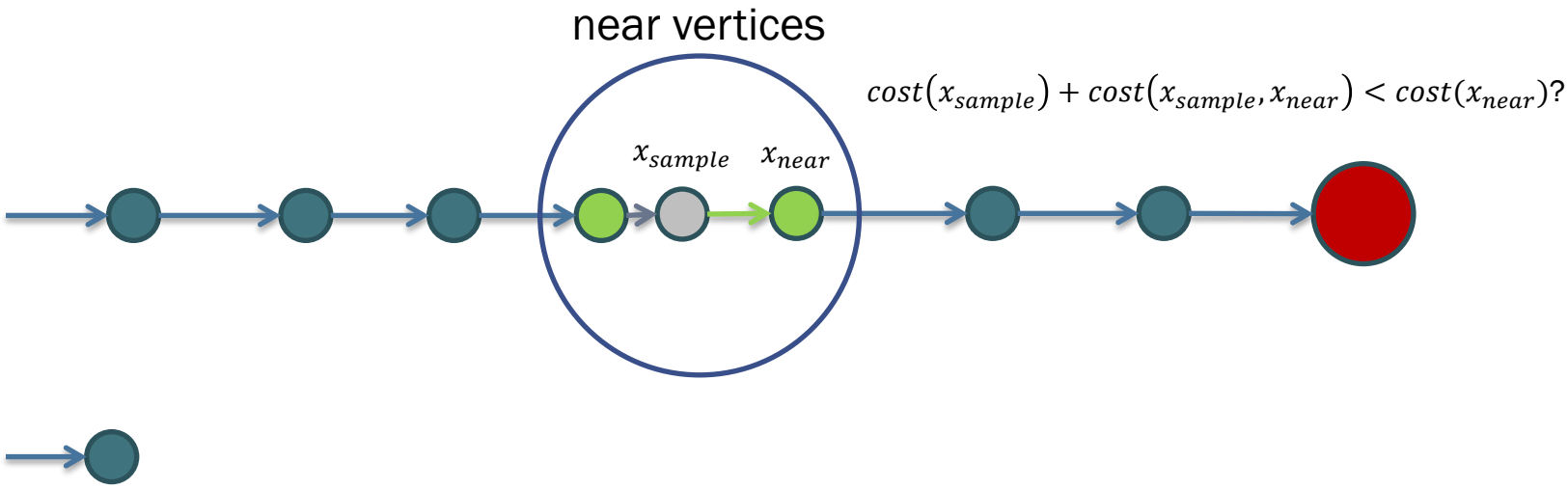
S. Karaman and E. Frazzoli, "Incremental sampling-based algorithms for optimal motion planning", Robotics: Science and Systems, 2010

V-CHARGE

RRT*: Asymptotically Optimal RRT

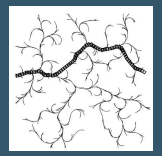


- RRT*
  - Introduces local rewiring step to obtain asymptotic optimality



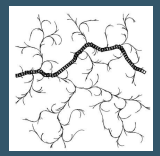
S. Karaman and E. Frazzoli, "Incremental sampling-based algorithms for optimal motion planning", Robotics: Science and Systems, 2010

RRT*: Pros & Cons



Pro	Con
• Asymptotically **complete** • Asymptotical **optimality guarantee** • Works reasonably well in **high-dimensional state-spaces** • **No two-state boundary value solver** required • **Easy implementation** • Easy to deal with **constrained platforms**	• **Two-state boundary value solver** required (rewiring phase) • Produces **"jerky" paths** in finite time • Hardly any **offline computations** possible

Randomized Approaches: Food for Thoughts



- Is *random* key?
  - Work on derandomizing RRTs while preserving their strength in exploration (voronoi bias)
    - Lindemann, S. LaValle, S., “Steps toward derandomizing RRTs”, *Proceedings of the Fourth International Workshop on Robot Motion and Control*, 2004
    - Count nearest neighbor occurrences to deterministic samples for vertices in the tree
      - Expand state with highest number of nearest neighbors towards average of its nearest neighbors → voronoi bias without randomization

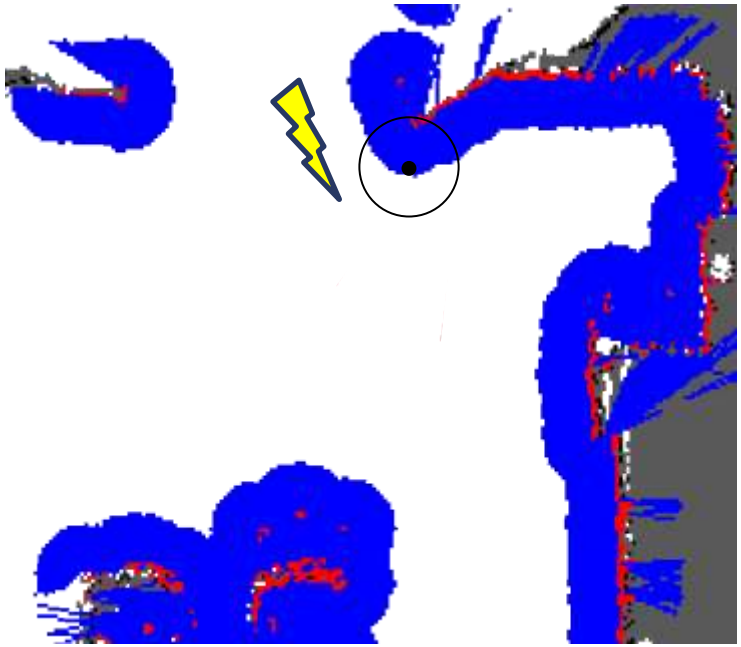
      → Multi-sample RRT

Overview

- The Basic Motion Planning Problem
  - I/O
  - Notions and Terms
- Selected algorithms for motion planning
  - The Dynamic Window Approach
  - Graph-Search, State-Lattices
  - Randomized Approaches: RRT, RRT*
- Collision Detection
- Detailed case studies from DARPA Urban Challenge
  - Boss (Carnegie Mellon)
  - AnnieWay (Karlsruhe/Munich)
  - Junior (Stanford)
- The V-Charge Motion Planning Framework
- Getting started for the technical session

Collision Detection with Static Obstacles

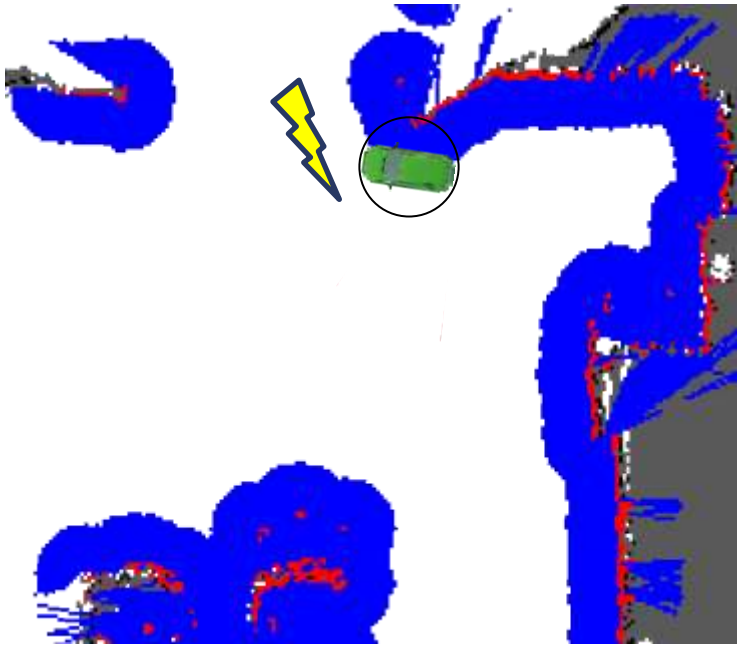
- Occupancy grids
  - Map inflation/dilation
    - Increase the size of occupied cells in the map by the robot's radius
    - Minkowski sum of disk and obstacles in the map
    - Very fast algorithms exist. Less than *5ms* on *500x500* grid.
      - Thresholded distance transform
        - "Felzenszwalb, P., & Huttenlocher, D. (2004). *Distance transforms of sampled functions*
        - Implementation in OpenCV available
    - Collision query reduces to single lookup in inflated grid



http://wiki.ros.org/costmap_2d

Collision Detection with Static Obstacles

- Occupancy grids
  - Map inflation/dilation
    - Increase the size of occupied cells in the map by the robot's radius
    - Minkowski sum of disk and obstacles in the map
    - Very fast algorithms exist. Less than *5ms* on *500x500* grid.
      - Thresholded distance transform
        - "Felzenszwalb, P., & Huttenlocher, D. (2004). *Distance transforms of sampled functions*
        - Implementation in OpenCV available
    - Collision query reduces to single lookup in inflated grid

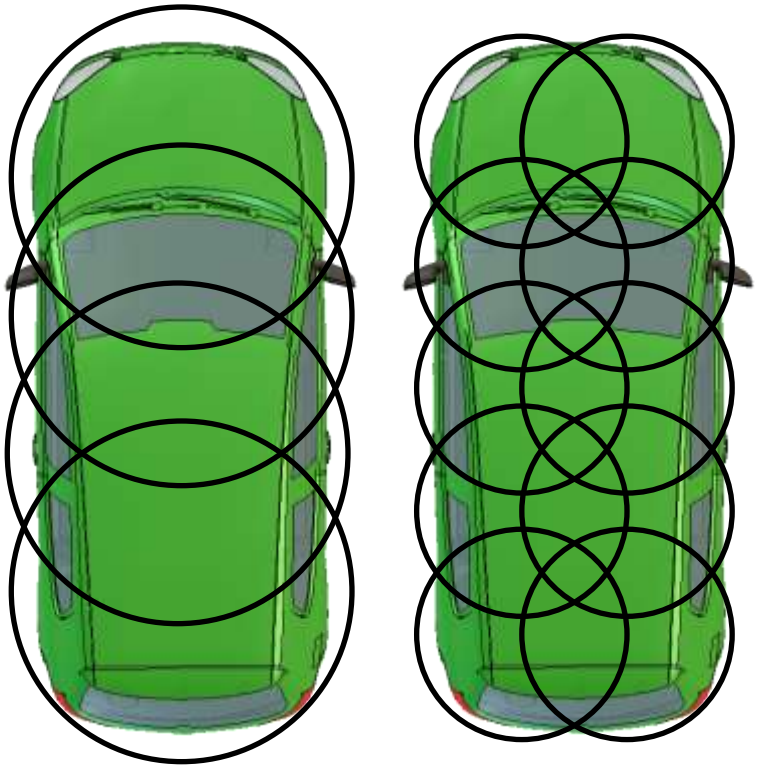


http://wiki.ros.org/costmap_2d

But: A car is not a disc nor can it be approximated sufficiently with one!

Collision Detection with Static Obstacles

- Approximation of rectangular vehicle shape with N equal sized disks
  - N lookups on inflated map
- Overlap decreases with increasing amount of disks

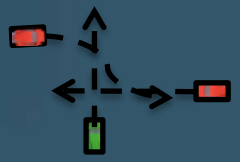


Example:

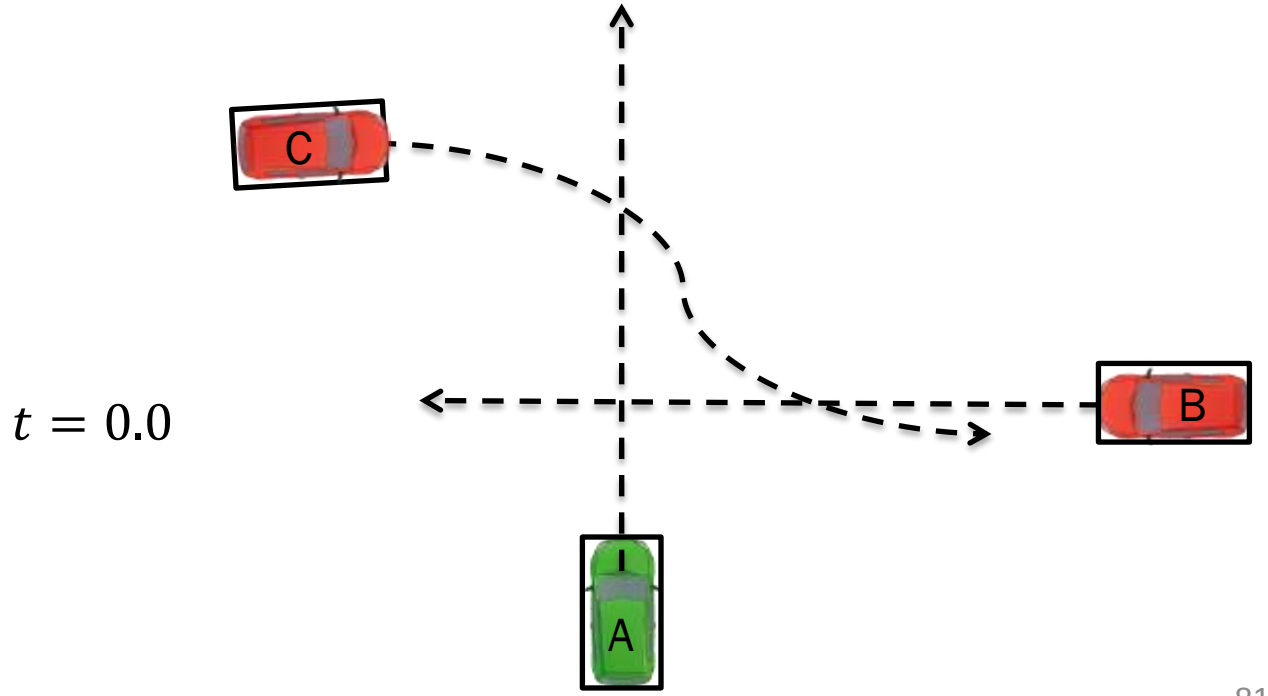
- *15‘000‘000 configurations* (3 disks) in *1.4 seconds*
- *~10’000 trajectories* (100 discrete configurations per trajectory) in *100 milliseconds*

V-CHARGE

Collision Detection with Dynamic Obstacles: Multiple Interference Test

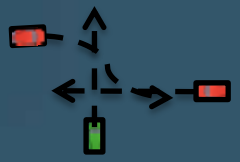


- Naive/Exhaustive checks (multiple interference test)
  - Test each pair of configurations at each timestamp for intersection

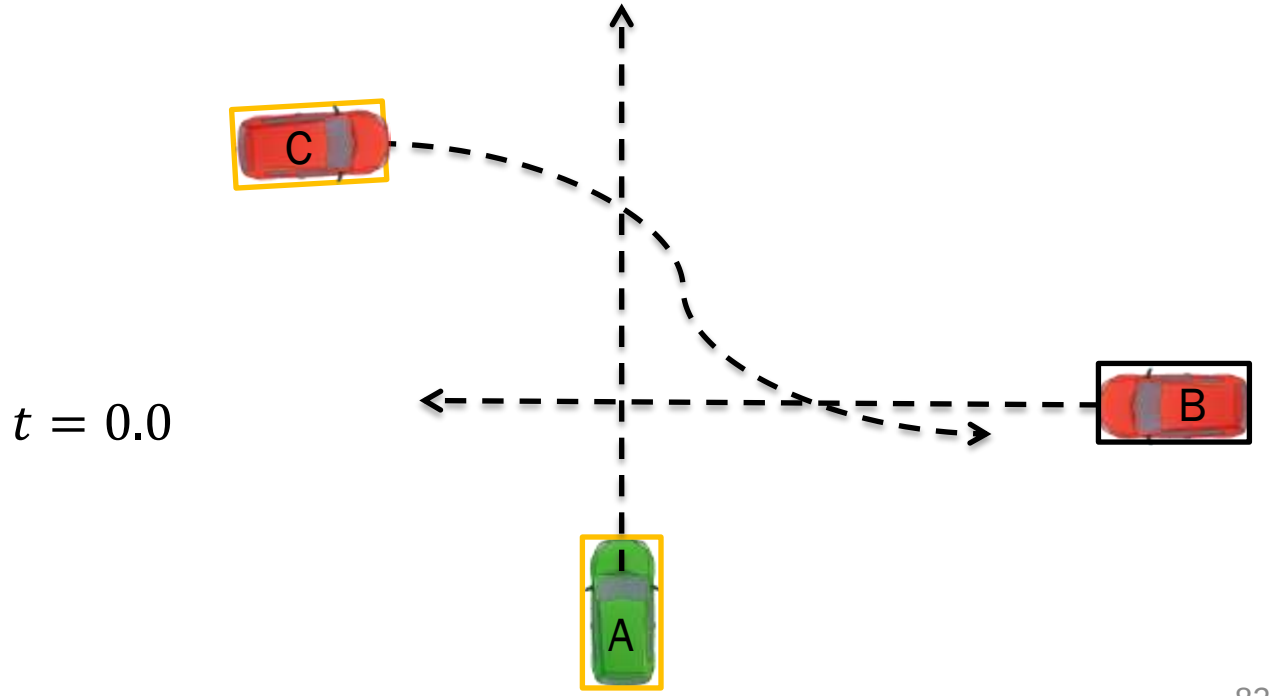


S. Cameron, “Modelling Solids in Motion”, PhD thesis, University of Edinburgh, 1984

Collision Detection with Dynamic Obstacles: Multiple Interference Test



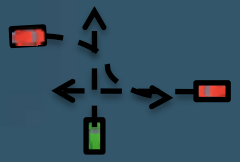
- Naive/Exhaustive checks (multiple interference test)
  - Test each pair of configurations at each timestamp for intersection



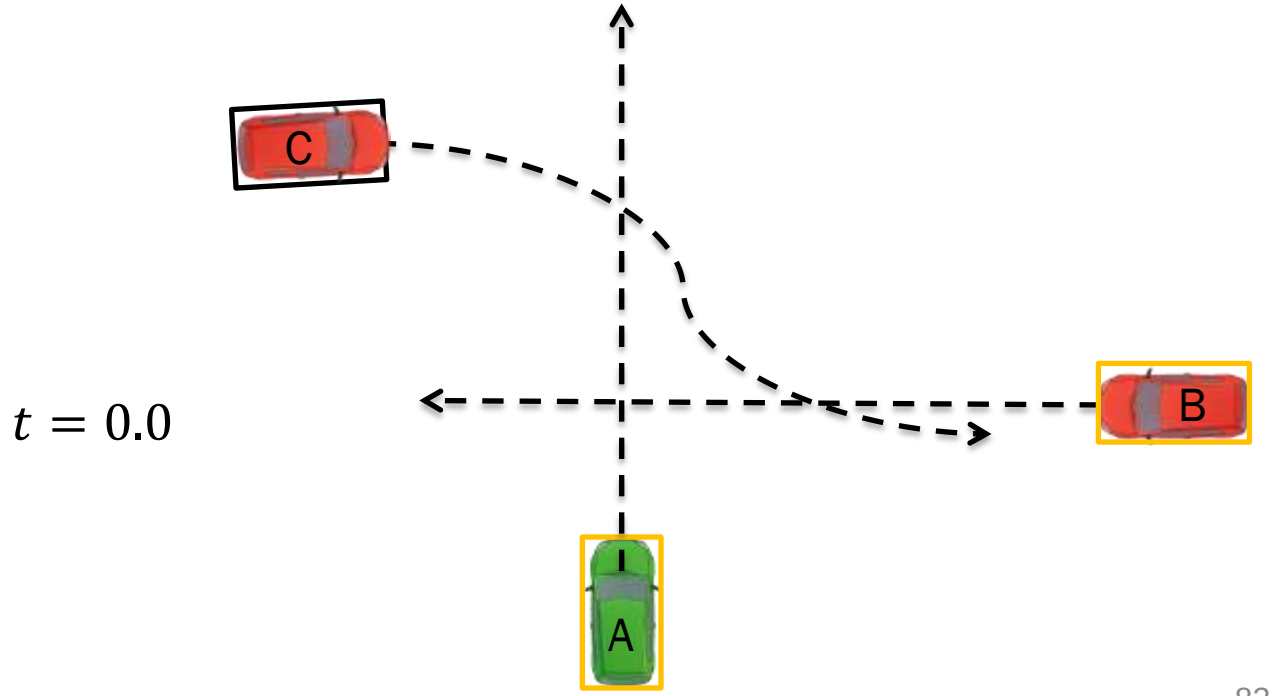
S. Cameron, “Modelling Solids in Motion”, PhD thesis, University of Edinburgh, 1984

V-CHARGE

Collision Detection with Dynamic Obstacles: Multiple Interference Test

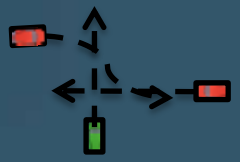


- Naive/Exhaustive checks (multiple interference test)
  - Test each pair of configurations at each timestamp for intersection

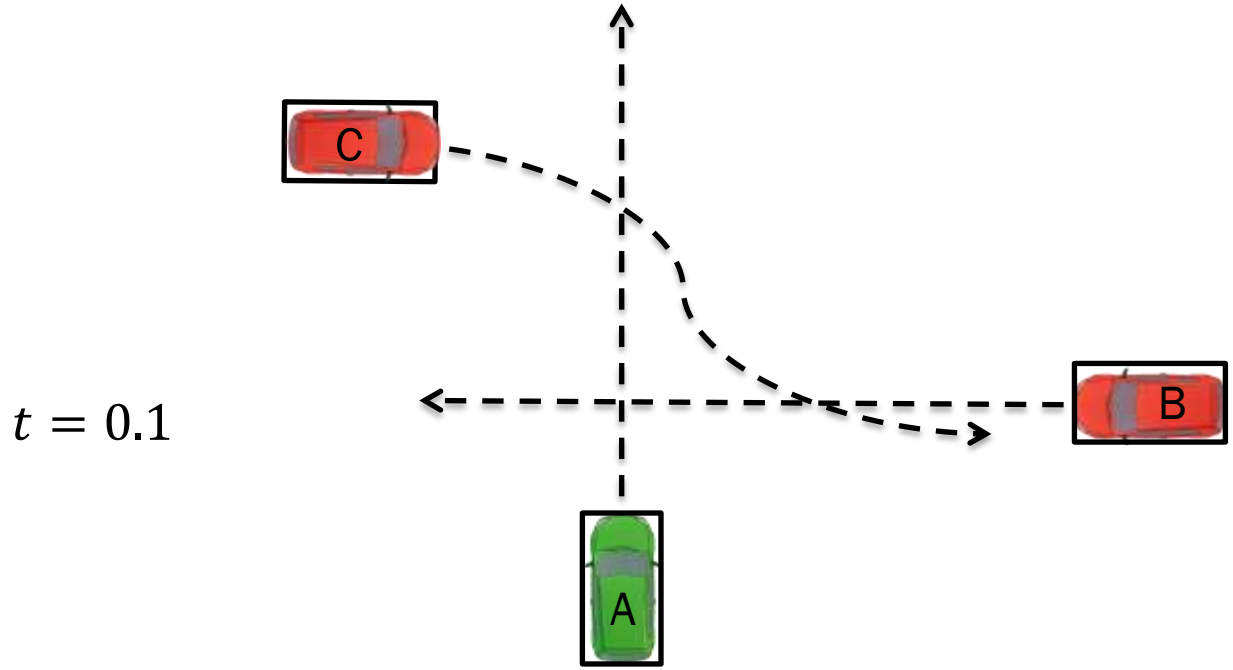


S. Cameron, “Modelling Solids in Motion”, PhD thesis, University of Edinburgh, 1984

Collision Detection with Dynamic Obstacles: Multiple Interference Test



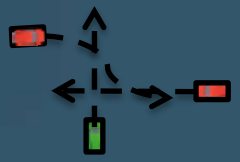
- Naive/Exhaustive checks (multiple interference test)
  - Test each pair of configurations at each timestamp for intersection



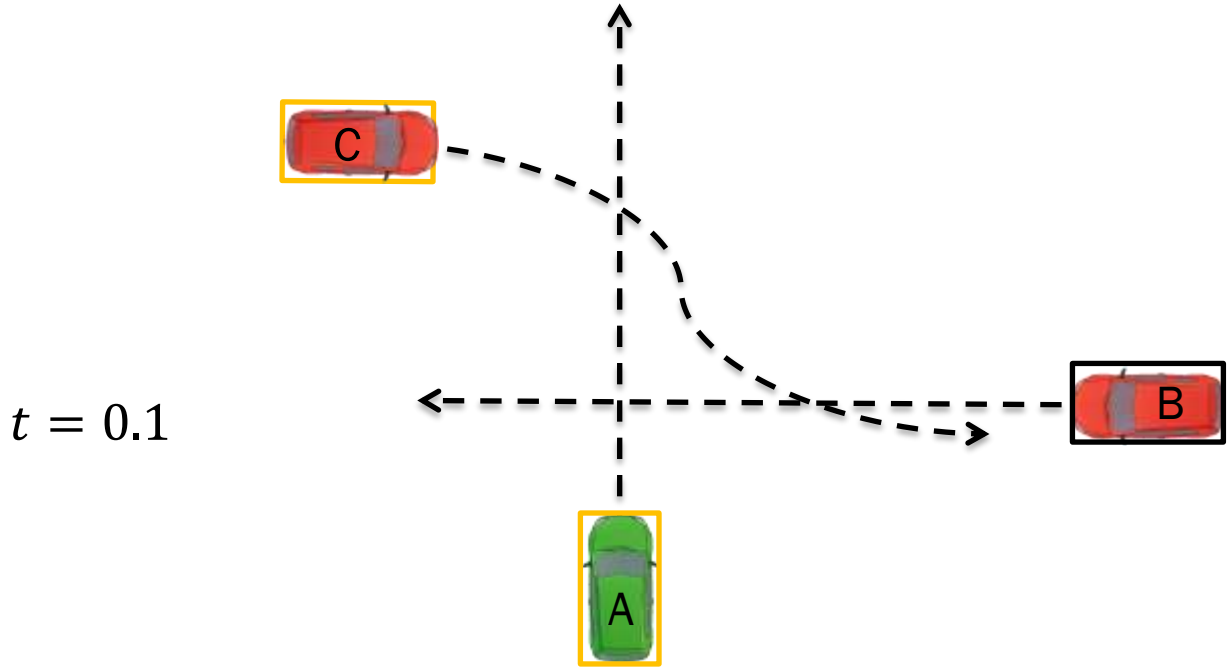
S. Cameron, “Modelling Solids in Motion”, PhD thesis, University of Edinburgh, 1984

V-CHARGE

Collision Detection with Dynamic Obstacles: Multiple Interference Test



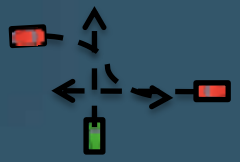
- Naive/Exhaustive checks (multiple interference test)
  - Test each pair of configurations at each timestamp for intersection



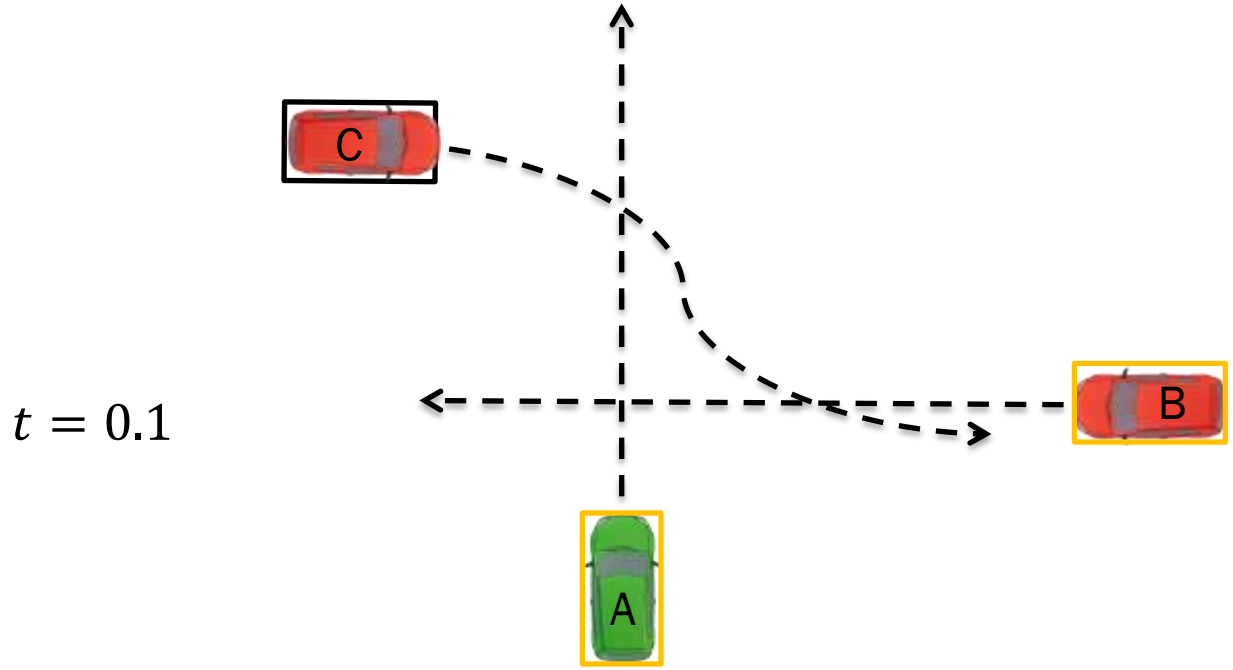
S. Cameron, “Modelling Solids in Motion”, PhD thesis, University of Edinburgh, 1984

V-CHARGE

Collision Detection with Dynamic Obstacles: Multiple Interference Test



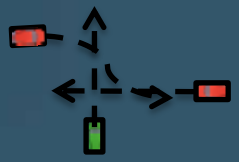
- Naive/Exhaustive checks (multiple interference test)
  - Test each pair of configurations at each timestamp for intersection



S. Cameron, “Modelling Solids in Motion”, PhD thesis, University of Edinburgh, 1984

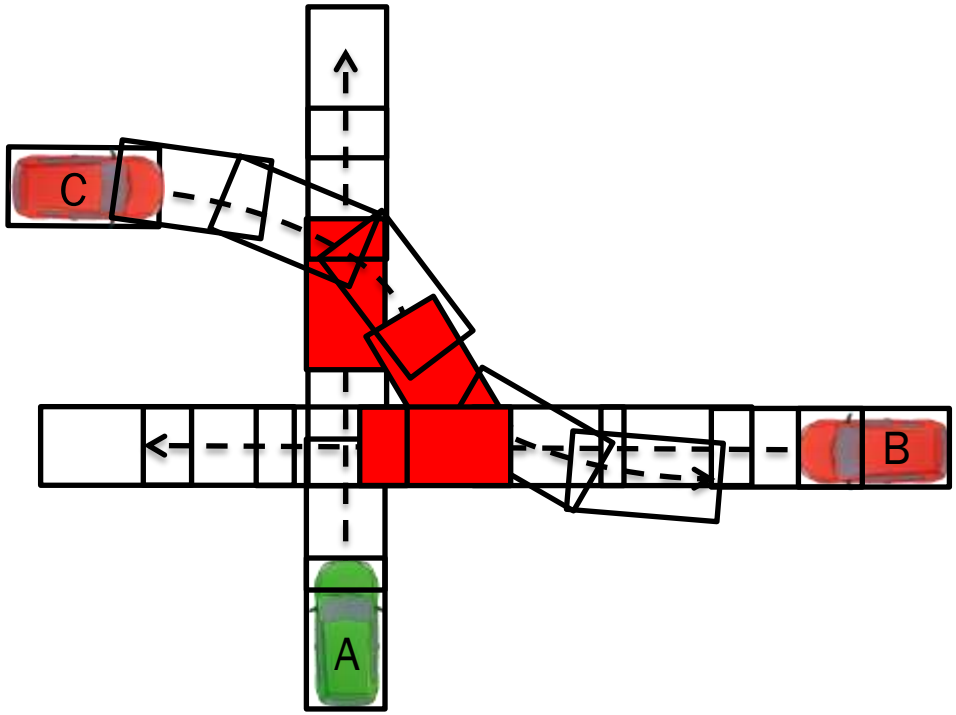
V-CHARGE

Collision Detection with Dynamic Obstacles: Multiple Interference Test



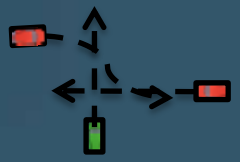
- Naive/Exhaustive checks (multiple interference test)
  - Test each pair of configurations at each timestamp for intersection

$t = 0.1$



S. Cameron, “Modelling Solids in Motion”, PhD thesis, University of Edinburgh, 1984

Collision Detection with Dynamic Obstacles: Multiple Interference Test

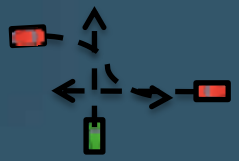


- Naive/Exhaustive checks (multiple interference test)
  - Test each pair of configurations at each timestamp for intersection

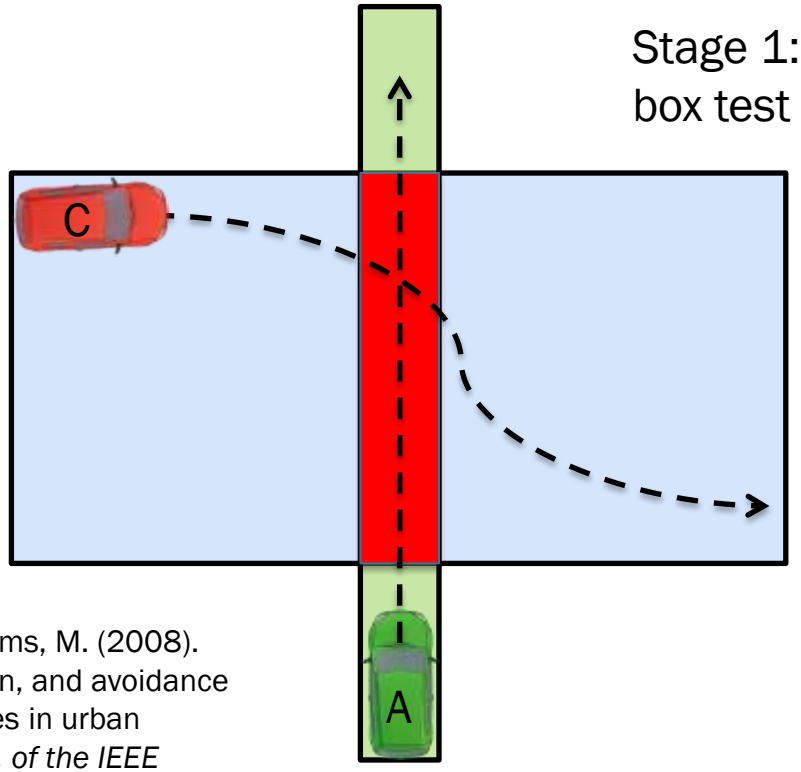
Testing oriented boxes is relatively expensive.

→ Conservative trivial tests to **prune potential collision pairs**

Collision Detection with Dynamic Obstacles



- Hierarchical Pruning
  - Conservative trivial tests to **prune potential collision pairs**

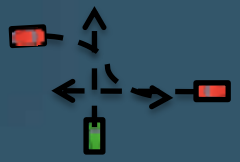


Stage 1: Axis-aligned bounding box test for **complete trajectories**

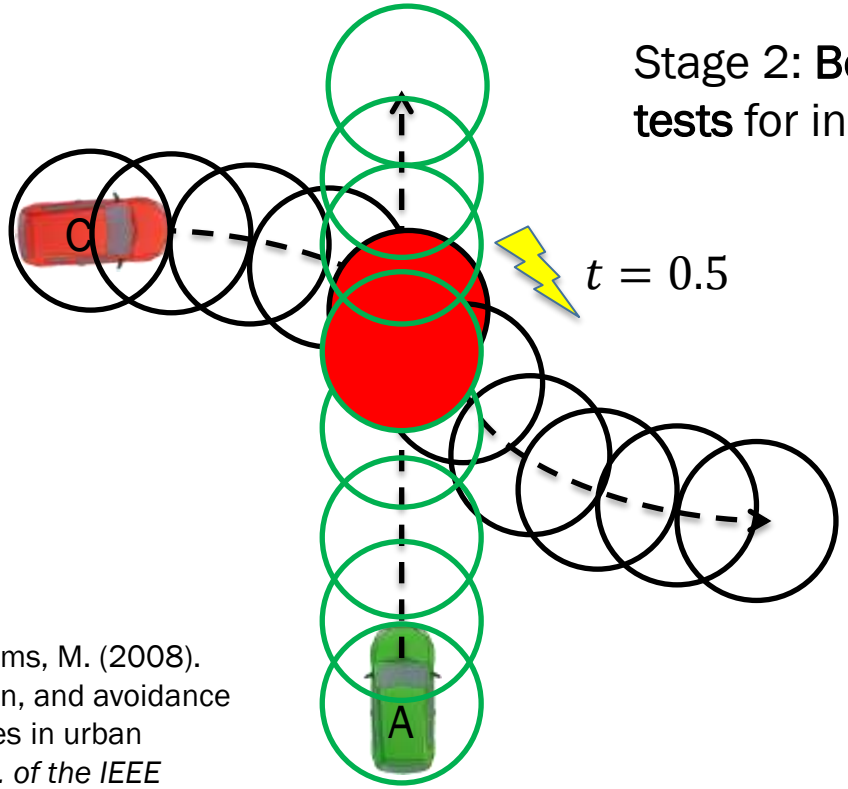
Ferguson, D., & Darms, M. (2008). Detection, prediction, and avoidance of dynamic obstacles in urban environments. *Proc. of the IEEE Intelligent Vehicles Symposium (IVS)*.

V-CHARGE

Collision Detection with Dynamic Obstacles



- Hierarchical Pruning
  - Conservative trivial tests to **prune potential collision pairs**

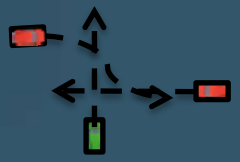


Stage 2: **Bounding spheres tests** for individual timesteps

Ferguson, D., & Darms, M. (2008). Detection, prediction, and avoidance of dynamic obstacles in urban environments. *Proc. of the IEEE Intelligent Vehicles Symposium (IVS)*.

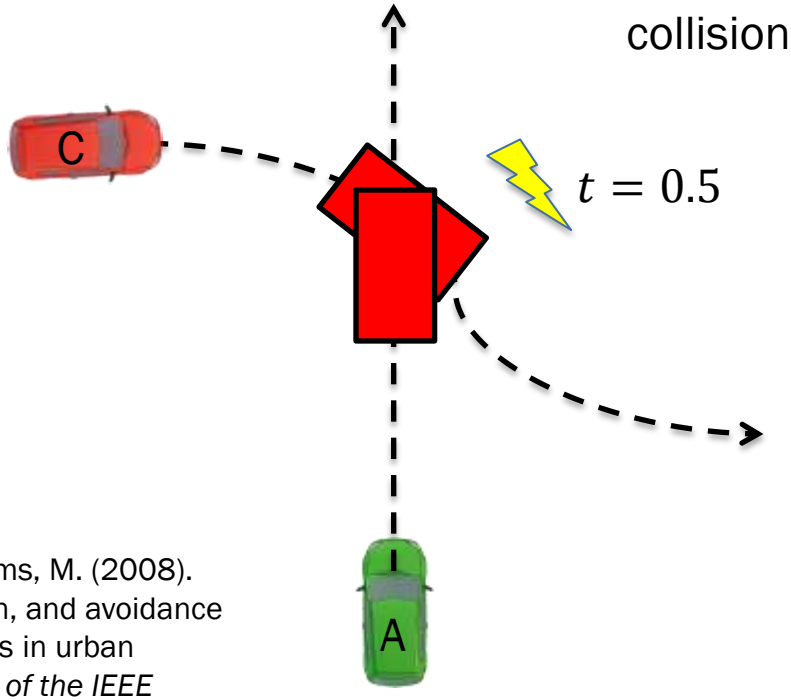
V-CHARGE

Collision Detection with Dynamic Obstacles



- Hierarchical Pruning
  - Conservative trivial tests to **prune potential collision pairs**

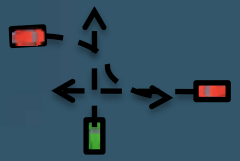
Stage 3: **Oriented box test** for collision candidates



Ferguson, D., & Darms, M. (2008). Detection, prediction, and avoidance of dynamic obstacles in urban environments. *Proc. of the IEEE Intelligent Vehicles Symposium (IVS)*.

V-CHARGE

Collision Detection with Dynamic Obstacles

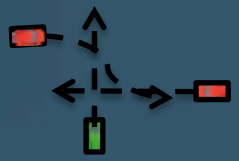


- Hierarchical Pruning
  - Conservative trivial tests to **prune potential collision pairs**

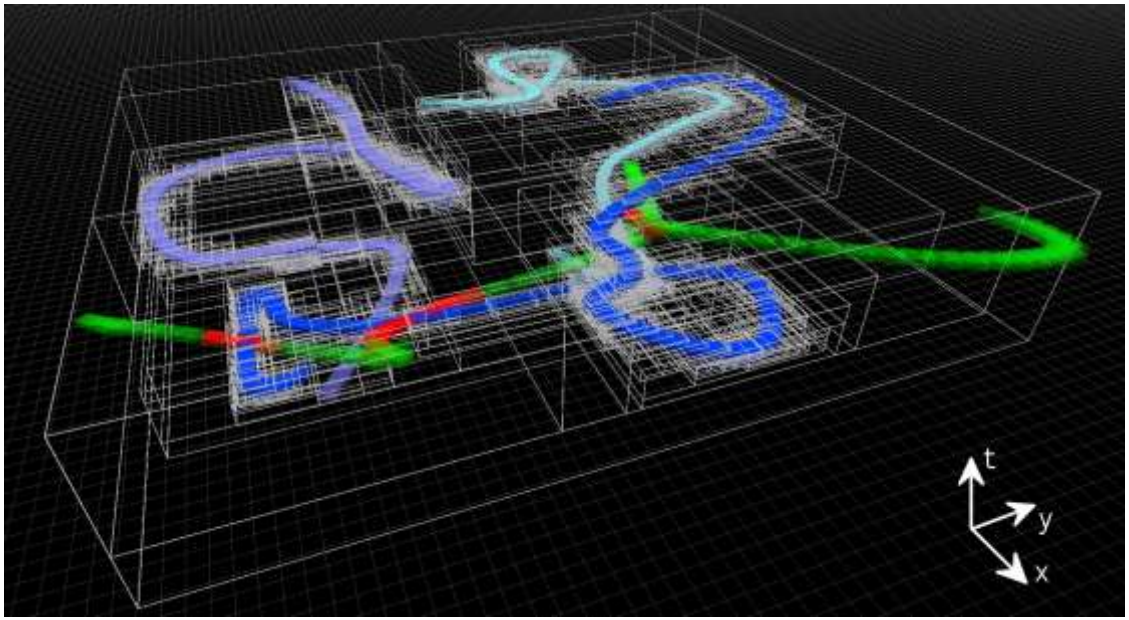
Still have to perform bounding spheres test for each timestep if axis-aligned trajectory test returns a potential collision. Complexity grows linearly with number of obstacles.

→ **Bounding volume hierarchy** (e.g. Axis-Aligned Bounding Box Tree) in workspace-time space provides **more efficient pruning**

Collision Detection with Dynamic Obstacles



- **Bounding volume hierarchy** (e.g. Axis-Aligned Bounding Box Tree) in workspace-time space provides more efficient pruning
  - Multiple interference test against all obstacles in workspace-time space at once



Non-colliding ego configurations

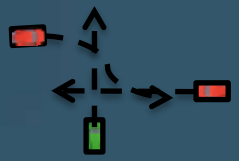
Colliding ego configurations

Obstacle configurations

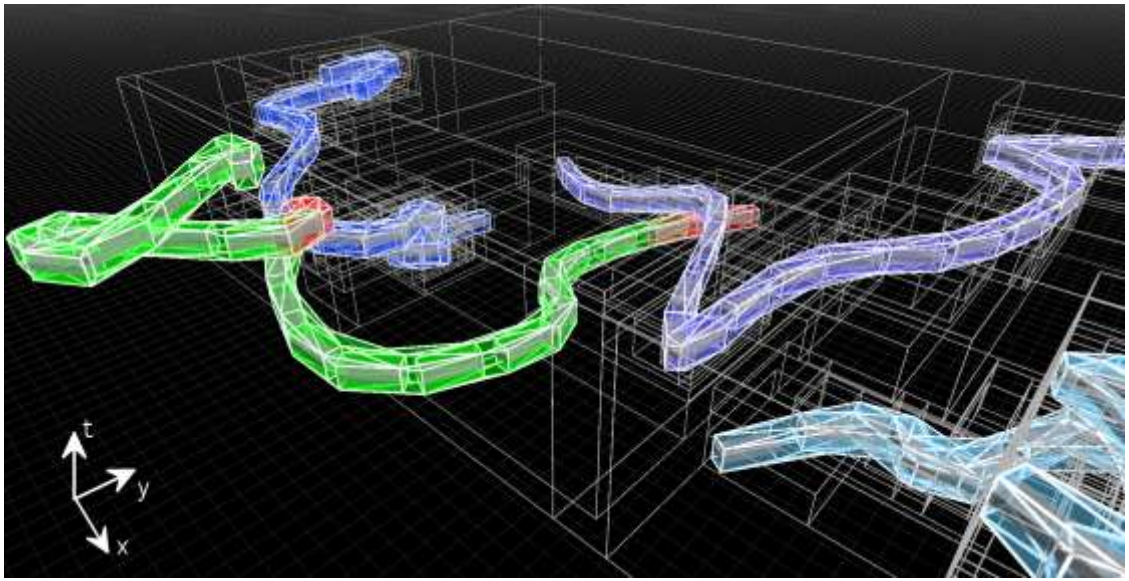
Axis-aligned bounding boxes for obstacles

V-CHARGE

Collision Detection with Dynamic Obstacles



- **Bounding volume hierarchy** (e.g. Axis-Aligned Bounding Box Tree) in workspace-time space provides more efficient pruning
  - Convex-hull computation closes gaps in workspace-time space



Non-colliding ego configurations

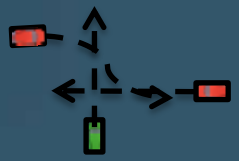
Colliding ego configurations

Obstacle configurations

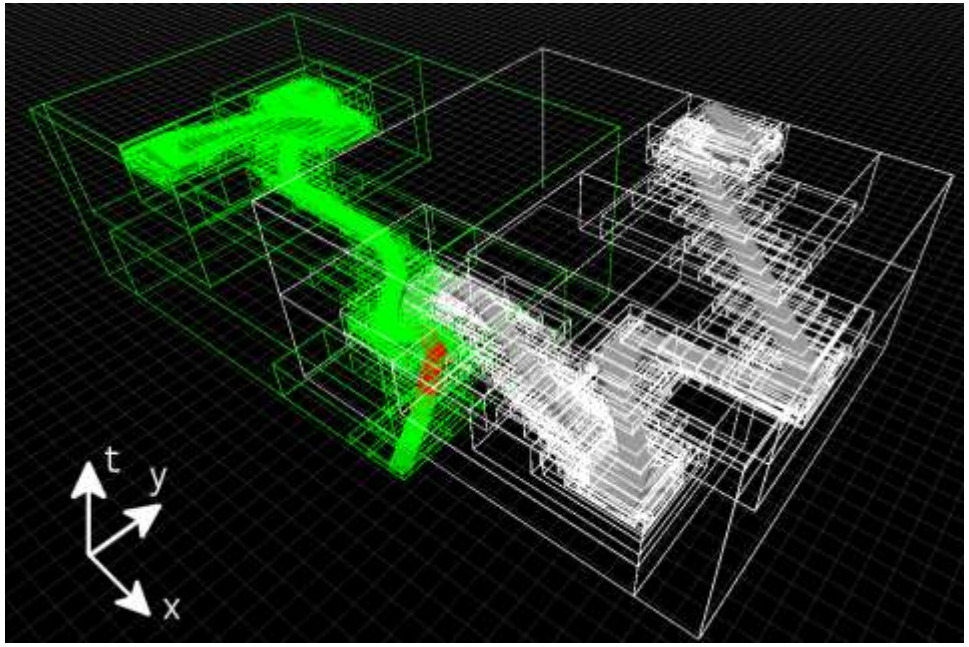
Axis-aligned bounding boxes for obstacles

V-CHARGE

Collision Detection with Dynamic Obstacles



- **Bounding volume hierarchy** (e.g. Axis-Aligned Bounding Box Tree) in workspace-time space provides more efficient pruning
  - Testing two AABB-Trees for collision further reduces computation times
    → offline computation of AABB-Tree data structure for ego motions



Non-colliding ego configurations

Colliding ego configurations

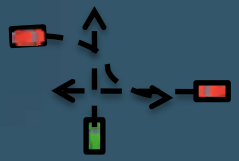
Obstacle configurations

Axis-aligned bounding boxes for obstacle configurations

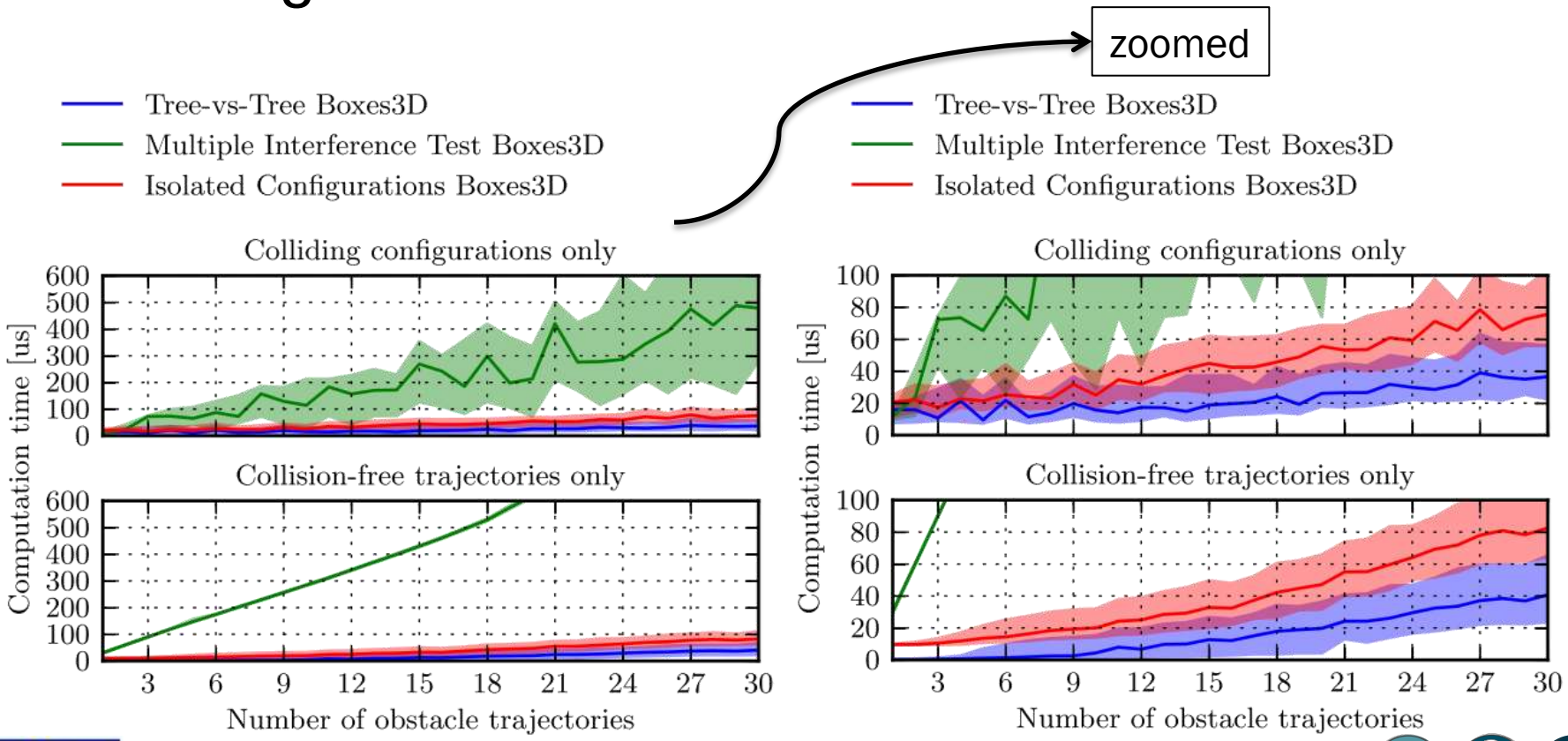
Axis-aligned bounding boxes for ego configurations

V-CHARGE

Collision Detection with Dynamic Obstacles



- **Bounding volume hierarchy approach:** Profiling



V-CHARGE

Overview

- The Basic Motion Planning Problem
  - I/O
  - Notions and Terms
- Selected algorithms for motion planning
  - The Dynamic Window Approach
  - Graph-Search, State-Lattices
  - Randomized Approaches: RRT, RRT*
- Collision Detection
- Detailed case studies from DARPA Urban Challenge
  - Boss (Carnegie Mellon)
  - AnnieWay (Karlsruhe/Munich)
  - Junior (Stanford)
- The V-Charge Motion Planning Framework
- Getting started for the technical session

The DARPA Urban Challenge 2007

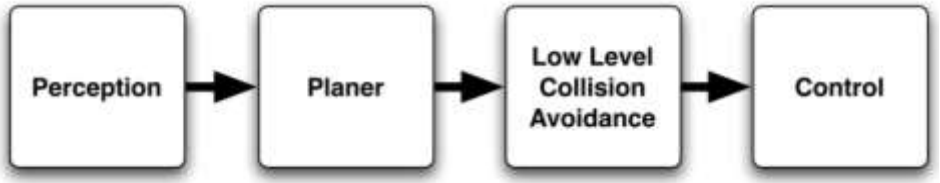
Annieway
Karlsruhe/Munich
not finished

Boss
Carnegie Mellon
1st place

Junior
Stanford
2nd place

V-CHARGE

The DARPA Urban Challenge: Team Annieway (Karlsruhe/Munich)

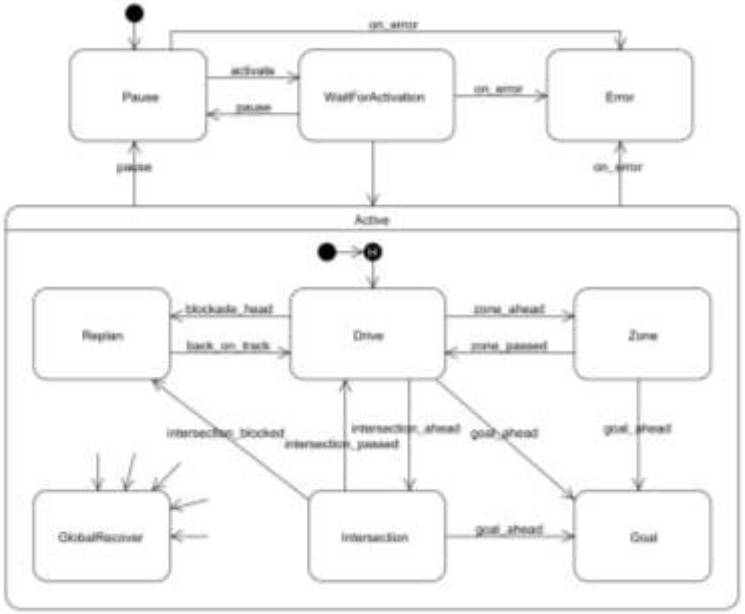


T. Gindele and D. Jagszent, “Design of the planner of Team AnnieWAY’s autonomous vehicle used in the DARPA Urban Challenge 2007,” *Intelligent Vehicles Symposium*, 2008

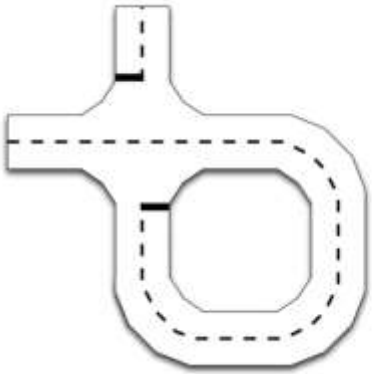
V-CHARGE

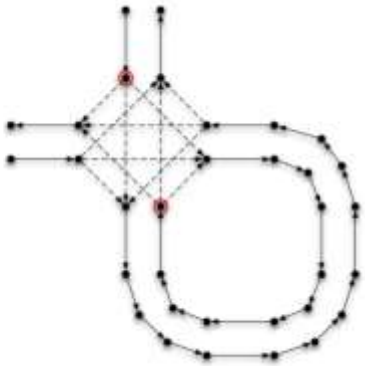
The DARPA Urban Challenge: Team Annieway (Karlsruhe/Munich)

- High-level state machine
  - regular driving on lanes
  - turning at intersections with oncoming traffic
  - lane changing maneuvers
  - vehicle following and passing
  - following order of precedence at 4-way stops
  - merging into moving traffic



- A*-search on roadgraph



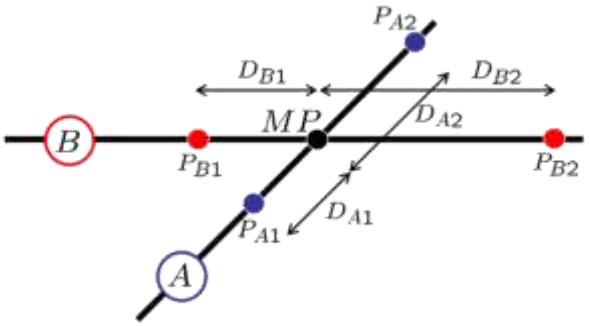


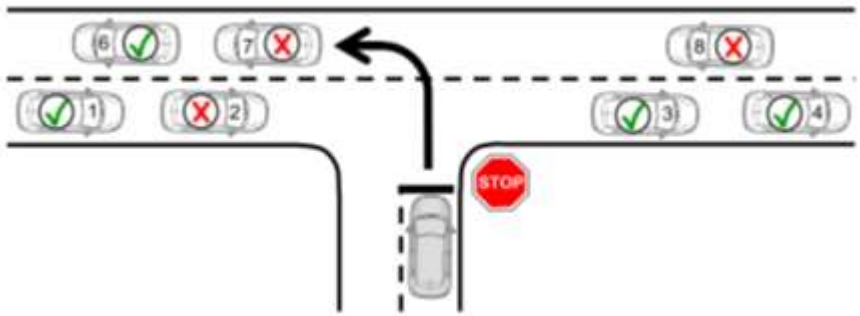
T. Gindele and D. Jagszent, “Design of the planner of Team AnnieWAY’s autonomous vehicle used in the DARPA Urban Challenge 2007”, Intelligent Vehicles Symposium, 2008.

V-CHARGE

The DARPA Urban Challenge: Team Annieway (Karlsruhe/Munich)

- Situational awareness module enhances capabilities of state-machine
  - Enforce spatial and temporal gap to moving objects along lanes
  - Simple feasibility check of maneuver
  - Assumptions:
    - Constant velocity of other traffic participants
    - Constant acceleration of ego vehicle until desired velocity



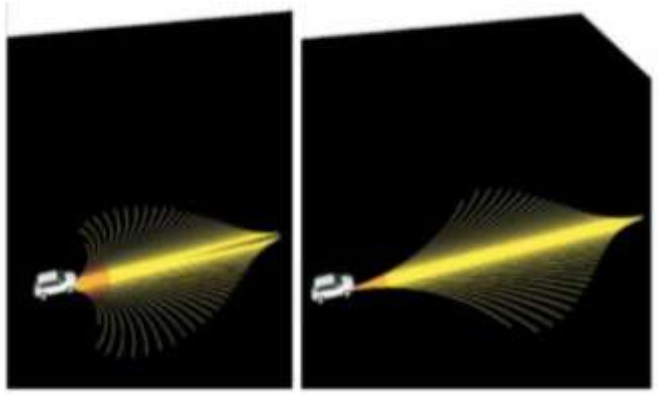


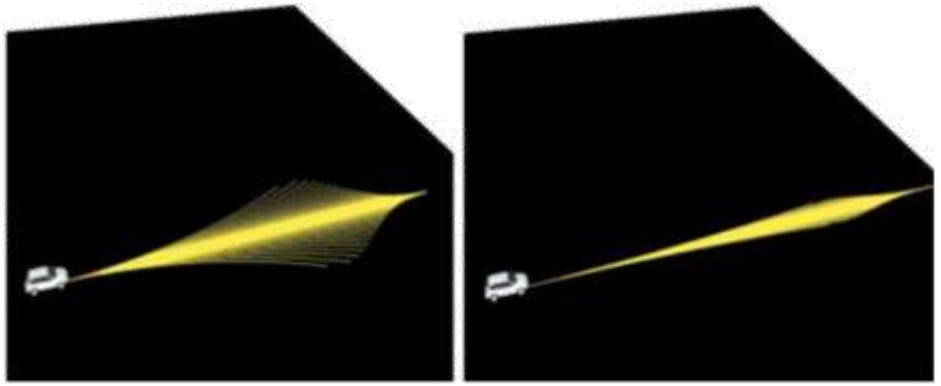
M. Werling and T. Gindele, “A robust algorithm for handling moving traffic in urban scenarios”, Intelligent Vehicles Symposium, 2008.

V-CHARGE

The DARPA Urban Challenge: Team Annieway (Karlsruhe/Munich)

- Pre-computed sets of motion primitives for different initial velocities
  - Constant steering angles → circular arcs (Dynamic Window Approach)
  - Arc lengths shorter for high curvatures to avoid endpoints behind vehicle

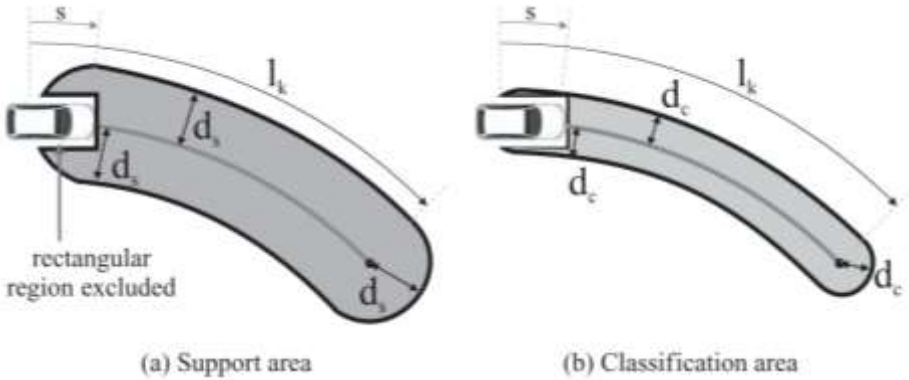


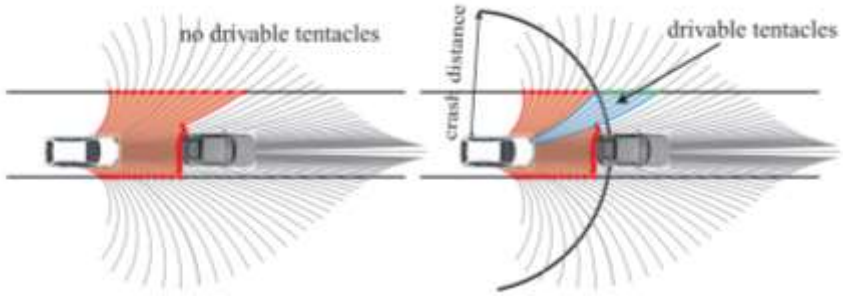


F. von Hundelshausen et al., “Driving with tentacles: Integral structures for sensing and motion”, Journal of Field Robotics, vol. 25, no. 9, pp. 640–673, 2008.

The DARPA Urban Challenge: Team Annieway (Karlsruhe/Munich)

- Classification area
  - collision detection
  - drivable/non-drivable classification based on *crash-distance*
- Support area
  - evaluation of cost function
- Speed-dependent size of areas





F. von Hundelshausen et al., “Driving with tentacles: Integral structures for sensing and motion”, Journal of Field Robotics, vol. 25, no. 9, pp. 640–673, 2008.

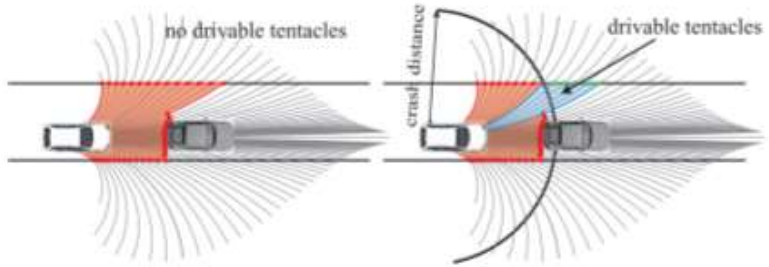
V-CHARGE

The DARPA Urban Challenge: Team Annieway (Karlsruhe/Munich)

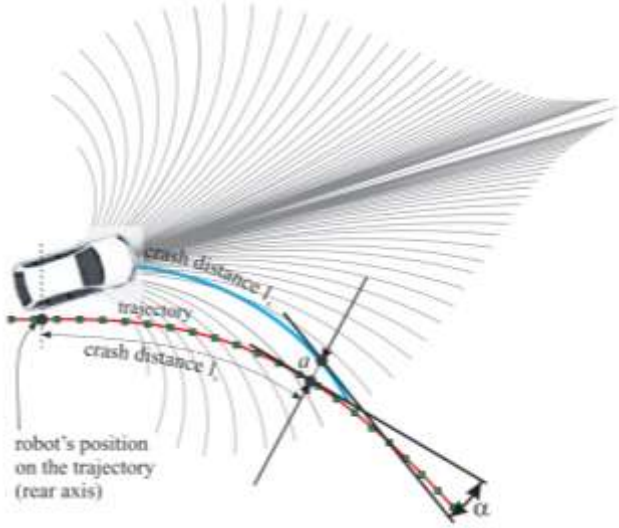
- Cost Function:
  - *clearance value*: $f_{\mathrm{c}} = f(d_{\mathrm{obs}})$
    - d_{obs}: distance to closest obstacle along trajectory
  - *flatness value*: $f_{\mathrm{f}} = f(T_{\mathrm{avg}})$
    - T_{avg}: averaged terrain flatness over support area
  - *trajectory value*: $f_{\mathrm{t}} = f(a, b)$
    - a : alignment of trajectory with a reference path
    - b: lateral offset of trajectory with a reference path

→ $f_{\mathrm{res}} = a_o f_{\mathrm{c}} + a_1 f_{\mathrm{f}} + a_2 f_{\mathrm{t}}$

clearance value:



trajectory value:



F. von Hundelshausen et al., "Driving with tentacles: Integral structures for sensing and motion", Journal of Field Robotics, vol. 25, no. 9, pp. 640–673, 2008.

V-CHARGE

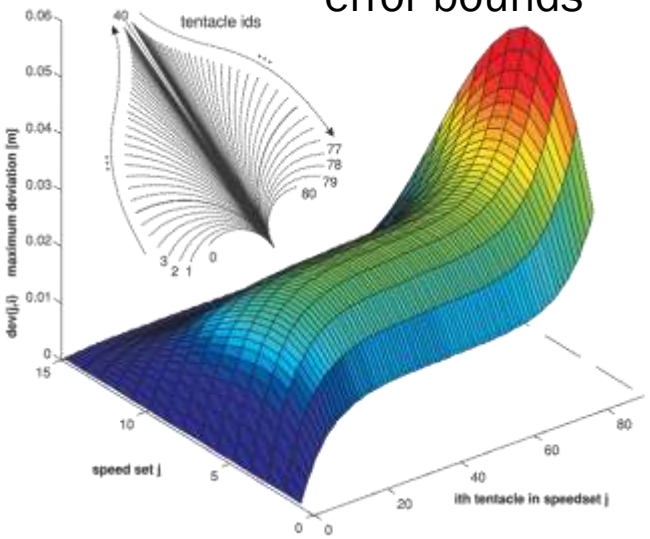
The DARPA Urban Challenge: Team Annieway (Karlsruhe/Munich)

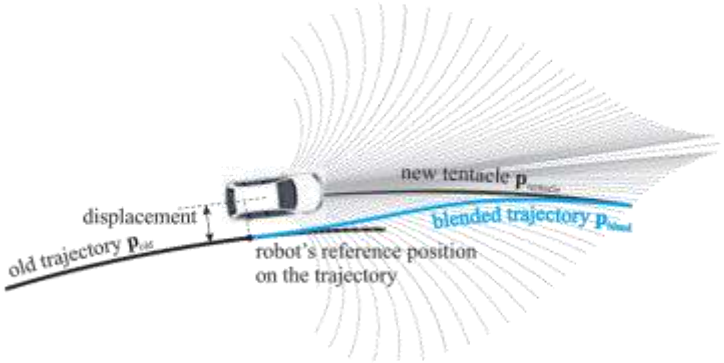
- Discussion
  - Non-drivable motion primitives due to negligence of initial steering angle

    → tedious work on deriving error bounds
    → temporal smoothing of steering commands

  - Post-processing of trajectories
    - Reset of error states upon replanning causes problems for underlying controller
      → blending with previous trajectory
  - Lots of tuning-parameters

error bounds



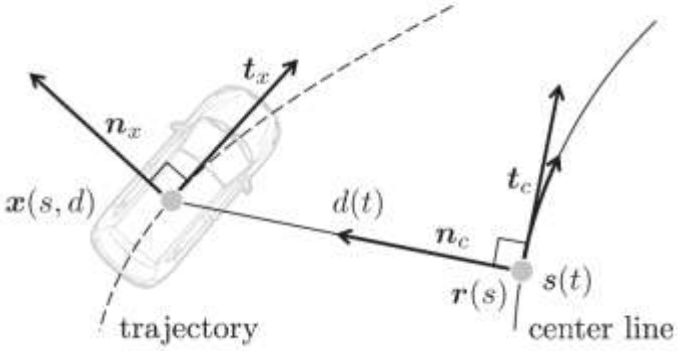


F. von Hundelshausen et al., "Driving with tentacles: Integral structures for sensing and motion", Journal of Field Robotics, vol. 25, no. 9, pp. 640–673, 2008.

V-CHARGE

Follow-up:
Team Annieway (Karlsruhe/Munich)

- Generation of candidate trajectories in Frenet frame
- Assumption: Separated integrators in lateral (d) and longitudinal direction (s)



Frenet-Frame

$$\dot{\xi} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \xi + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u, \qquad \xi = \begin{bmatrix} d \\ \dot{d} \\ \ddot{d} \end{bmatrix} \text{ or } \xi = \begin{bmatrix} s \\ \dot{s} \\ \ddot{s} \end{bmatrix}$$

- Goal: Minimize cost functional

$$J_\xi := \int_0^\tau f_0(u)\,\mathrm{d}t + (h(\xi(t),t))_\tau, \quad f_0(u) := \frac{1}{2}u^2(t)$$

$$(h(\xi(t),t))_\tau := \left(k_\tau t + \frac{1}{2}k_{\xi_1}[\xi_1(t) - \xi_{\mathrm{ref}}(t)]^2\right)_\tau$$

M. Werling, et al., “Optimal trajectories for time-critical street scenarios using discretized terminal manifolds”, *International Journal of Robtic Research*, Dec. 2011.

V-CHARGE

Follow-up:
Team Annieway (Karlsruhe/Munich)

- all initial and terminal states fixed → $\xi_1(t)$ is quintic polynomial
  - 6 boundary constraints, $\boldsymbol{\xi}(0) = \boldsymbol{\xi}_0$, $\boldsymbol{\xi}(\tau) = \boldsymbol{\xi}_\tau$
- terminal point $\xi_1(t)$ free → $\xi_1(t)$ is quartic polynomial
  - 5 boundary constraints, $\boldsymbol{\xi}(0) = \boldsymbol{\xi}_0$, $\xi_2(\tau) = \xi_{\tau,2}$, $\xi_3(\tau) = \xi_{\tau,3}$

$$\dot{\boldsymbol{\xi}} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \boldsymbol{\xi} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u$$

M. Werling, et al., “Optimal trajectories for time-critical street scenarios using discretized terminal manifolds”, *International Journal of Robtic Research*, Dec. 2011.

Follow-up:
Team Annieway (Karlsruhe/Munich)

- Boundary constraints by reference path
  - Align first and second derivative at time τ with reference path (terminal manifold)

$$\left(\begin{bmatrix} \xi_2(t) - \dot{\xi}_{\text{ref}}(t) \\ \xi_3(t) - \ddot{\xi}_{\text{ref}}(t) \end{bmatrix} \right)_{\tau} = \mathbf{0}$$

  - → $\xi_1(t)$ is quartic polynomial

Problem 1: optimal time τ and optimal terminal state $\boldsymbol{\xi}$ unknown.

Problem 2: no-collision-constraint not incorporated

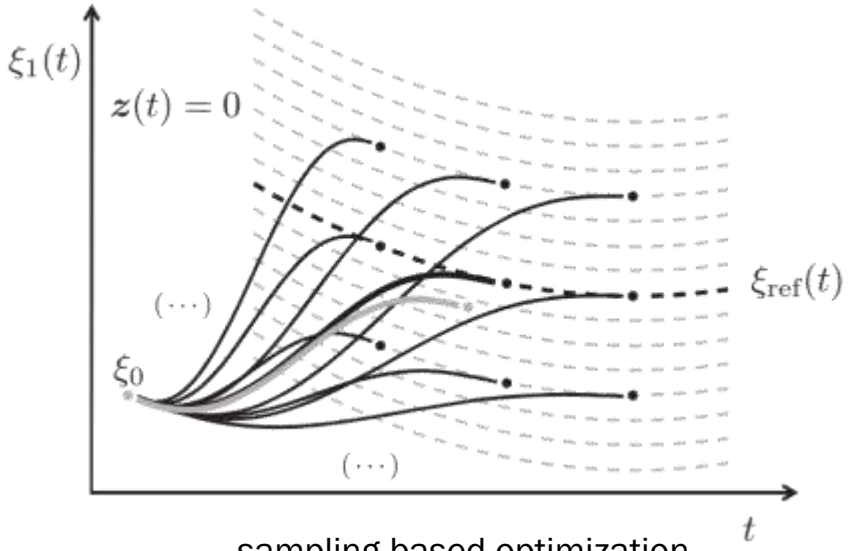
M. Werling, et al., “Optimal trajectories for time-critical street scenarios using discretized terminal manifolds”, *International Journal of Robtic Research*, Dec. 2011.

Follow-up:
Team Annieway (Karlsruhe/Munich)

Problem 1: optimal time τ and optimal terminal state $\boldsymbol{\xi}$ unknown.

Problem 2: no-collision-constraint not incorporated

→sampling-based optimization

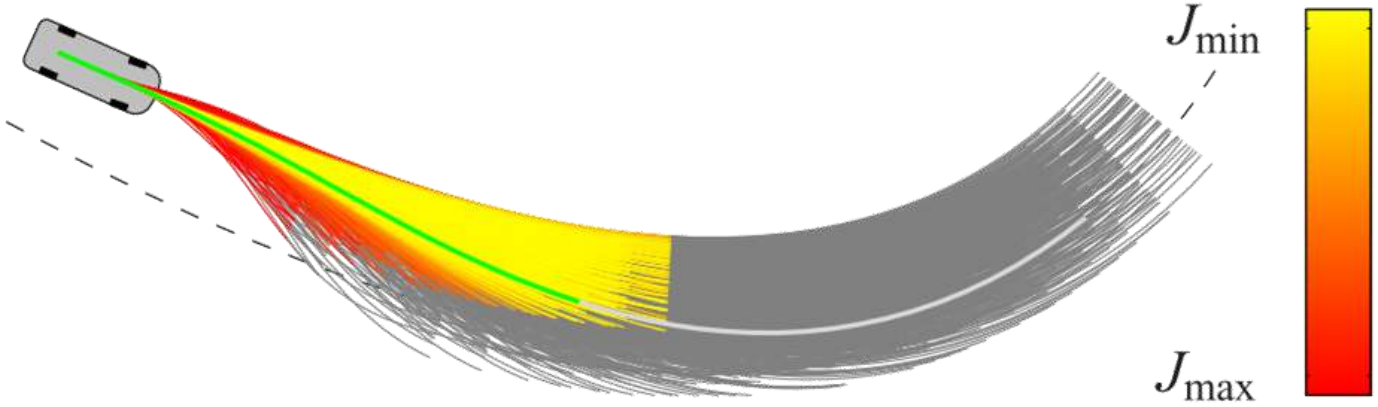


sampling-based optimization

M. Werling, et al., “Optimal trajectories for time-critical street scenarios using discretized terminal manifolds”, *International Journal of Robtic Research*, Dec. 2011.

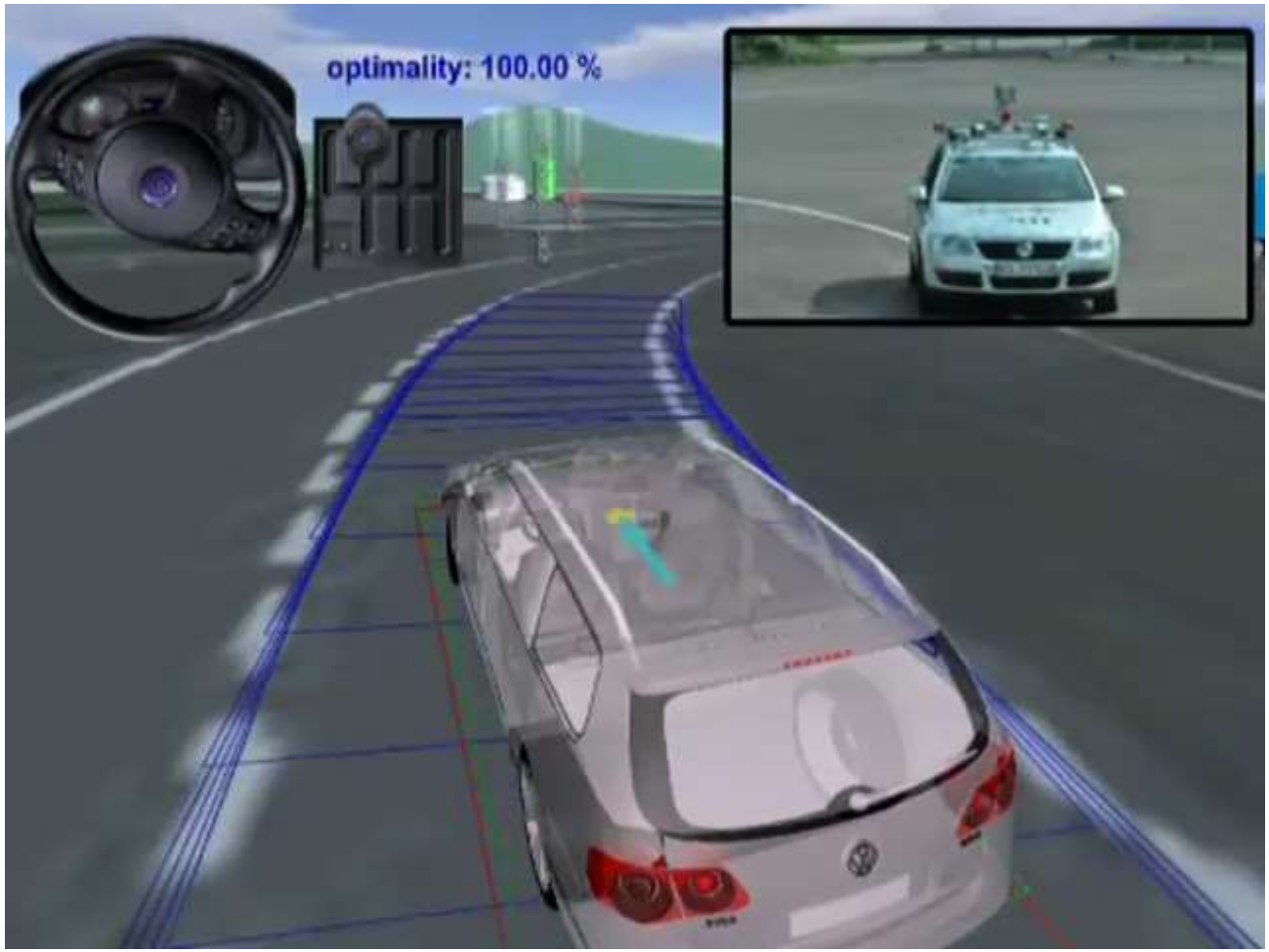
Follow-up:
Team Annieway (Karlsruhe/Munich)

- sampling-based optimization
  - create longitudinal and lateral trajectory sets in Frenet frame
  - merge sets
  - transform back to Cartesian frame
  - evaluate system constraints
    - trajectories are not inherently driveable by construction
  - evaluate objective function
  - check for collision (static + dynamic)
  - execute lowest cost trajectory



M. Werling, et al., “Optimal trajectories for time-critical street scenarios using discretized terminal manifolds”, *International Journal of Robtic Research*, Dec. 2011.

Follow-up:
Team Annieway (Karlsruhe/Munich)

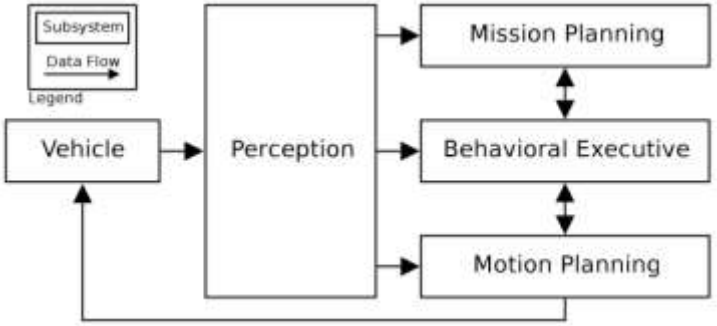


M. Werling, et al., “Optimal trajectories for time-critical street scenarios using discretized terminal manifolds”, *International Journal of Robtic Research*, Dec. 2011.

V-CHARGE

The DARPA Urban Challenge: Team Boss (Carnegie Mellon)

- Team Boss
  - Mission Planning
    - Computes expected time to reach waypoints
  - Behavioral Executive
    - List of waypoints (e.g. …,follow lane, park, …)
    - High-level management
      - goal-assignment
      - on-road driving
        - speed adaptation
        - lane-change maneuvers
      - intersection handling
  - Motion Planning
    - On-road driving
    - Unstructured zone navigation

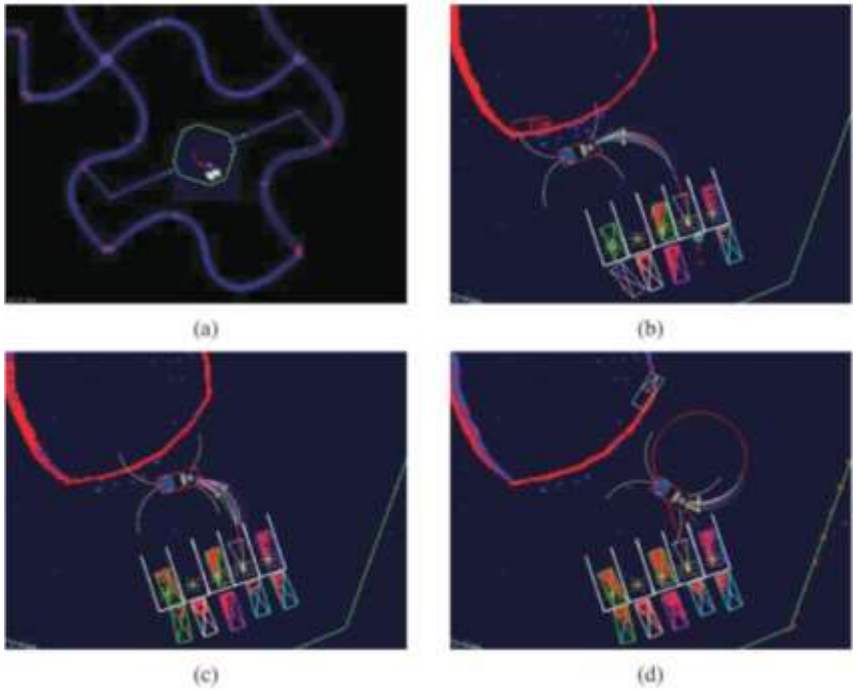


C. Baker and J. Dolan, "Traffic interaction in the urban challenge: Putting boss on its best behavior", IEEE Int. Conf. Intell. Robot. Syst., 2008.

V-CHARGE

The DARPA Urban Challenge: Team Boss (Carnegie Mellon)

- Motion Planning:
  - Unstructured:
    - Global:
      - Anytime D* graph-search
      - multiresolution 4D state-lattice (x, y , θ, v)
      - Maximum-of-two heuristic
    - Local:
      - Set of concatenations of two motion primitives (diverging and returning to path). *Non-holonomic trajectory generator* presented before.
  - On-road:
    - Global:
      - Take lane center
    - Local:
      - Motion primitives with final lateral offset to reference path). *Non-holonomic trajectory generator* presented before.



D. Ferguson, T. M. Howard, and M. Likhachev, “Motion planning in urban environments”, Journal of Field Robotics, vol. 25, no. 11–12, pp. 939–960, 2008.

The DARPA Urban Challenge: Team Boss (Carnegie Mellon)

Success recipies:

- Fast computation times ensure smooth behavior
  - Preprocessing suggested wherever possible
- Detailed global planning stage increases system performance
  - Minimize divergence between planning stages
- Accurate vehicle model minimizes divergence between planning and execution
  → higher speeds are safely driveable

Source: http://www.carpictures.com/vehicle/07K4E463930352/Chevrolet-Tahoe-Wins-Driverless-DARPA-Urban-Challenge-2007

D. Ferguson, T. M. Howard, and M. Likhachev, “Motion planning in urban environments”, Journal of Field Robotics, vol. 25, no. 11–12, pp. 939–960, 2008.

The DARPA Urban Challenge: Team Junior (Stanford)

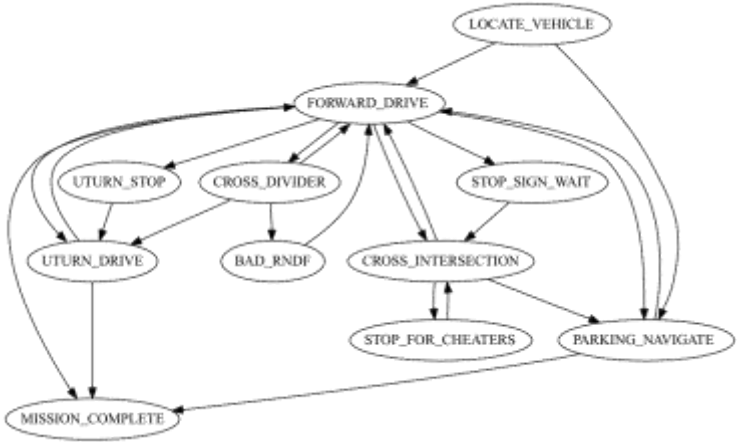
- Team Junior

M. Montemerlo et al., “Junior: The Stanford entry in the Urban Challenge”, Journal of Field Robotics, vol. 25, no. 9, pp. 569–597, 2008.

V-CHARGE

The DARPA Urban Challenge: Team Junior (Stanford)

- Motion Planning:
  - High-level state-machine



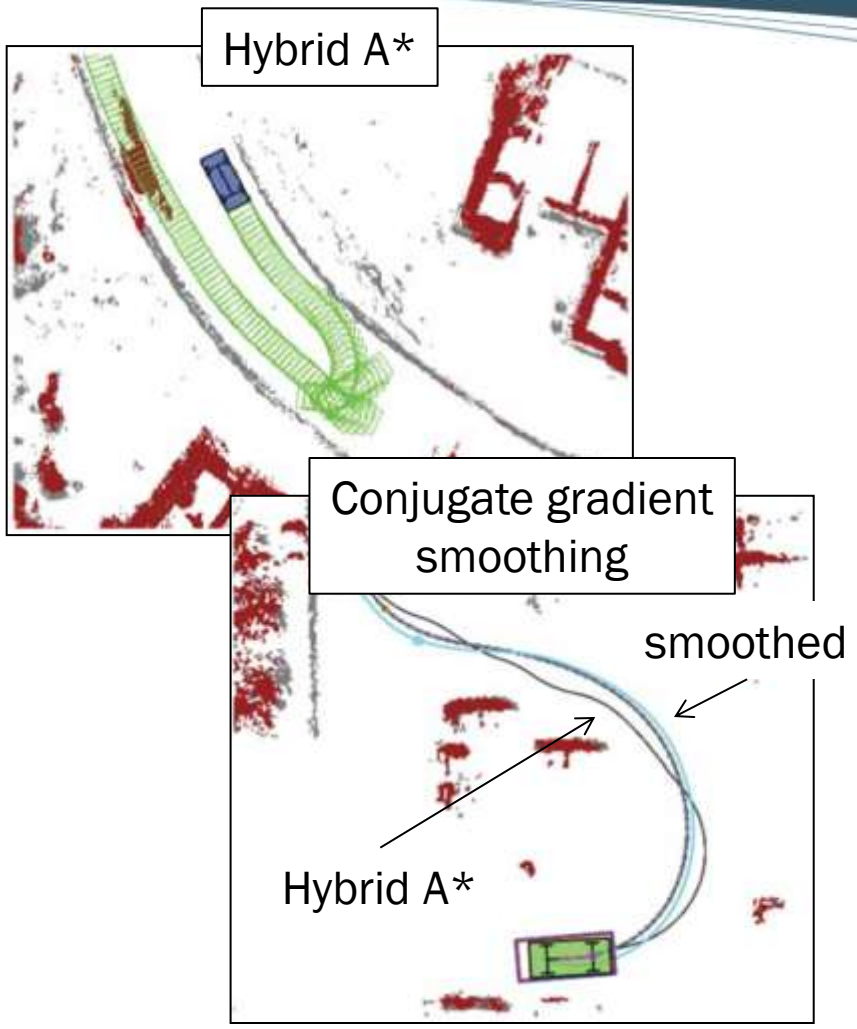
  - Graph-search on roadmap provides cost for every location to goal

M. Montemerlo et al., “Junior: The Stanford entry in the Urban Challenge”, Journal of Field Robotics, vol. 25, no. 9, pp. 569–597, 2008.

V-CHARGE

The DARPA Urban Challenge: Team Junior (Stanford)

- Motion Planning:
  - Hybrid A* for navigation in unstructured space
    - Maximum-of-two heuristic
  - Post-smoothing of paths by hybrid A* with conjugate gradient method



M. Montemerlo et al., “Junior: The Stanford entry in the Urban Challenge”, Journal of Field Robotics, vol. 25, no. 9, pp. 569–597, 2008.

The DARPA Urban Challenge: Team Junior (Stanford)

- Conjugate gradient smoothing
  - Post-optimization of

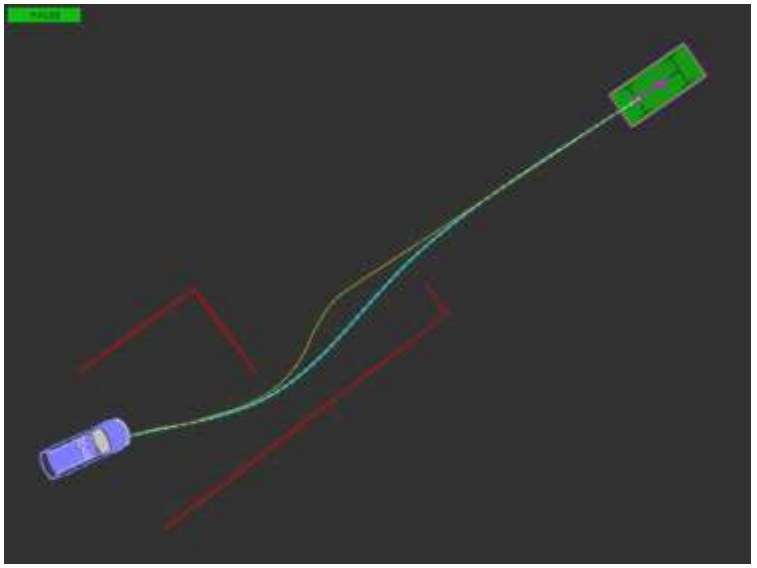
$$w_o \sum_{i=1}^{N} \sigma_o \left(|\mathbf{x}_i - \mathbf{o}_i| - d_{\max}\right) + w_\kappa \sum_{i=1}^{N-1} \sigma_\kappa \left(\frac{\Delta\phi_i}{|\Delta x_i|} - \kappa_{\max}\right)$$

$$+ w_s \sum_{i=1}^{N-1} (\Delta\mathbf{x}_{i+1} - \Delta\mathbf{x}_i)^2$$

Obstacle distance penalty

Maximum curvature violation penalty

Maximum curvature violation penalty



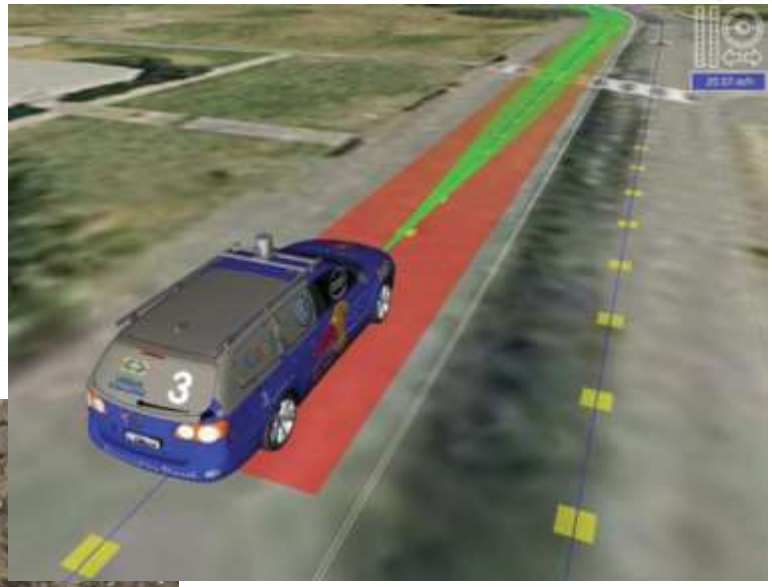
Hybrid A* solution

Conjugate gradient solution

D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, “Path Planning for Autonomous Vehicles in Unknown Semi-structured Environments,” *International Journal of Robic Research*, 2010.

The DARPA Urban Challenge: Team Junior (Stanford)

- Motion Planning:
  - Local trajectory roll-out
    - Joining local cost with cost-to-go from global planning stage



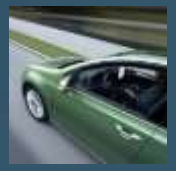
M. Montemerlo et al., “Junior: The Stanford entry in the Urban Challenge”, Journal of Field Robotics, vol. 25, no. 9, pp. 569–597, 2008.

Overview

- The Basic Motion Planning Problem
  - I/O
  - Notions and Terms
- Selected algorithms for motion planning
  - The Dynamic Window Approach
  - Graph-Search, State-Lattices
  - Randomized Approaches: RRT, RRT*
- Collision Detection
- Detailed case studies from DAR
  - Boss (Carnegie Mellon)
  - AnnieWay (Karlsruhe/Munich)
  - Junior (Stanford)
- The V-Charge Motion Planning Framework
- Getting started for the technical session

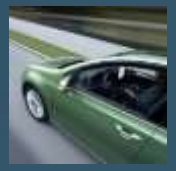
Important for technical session!

The V-Charge Car

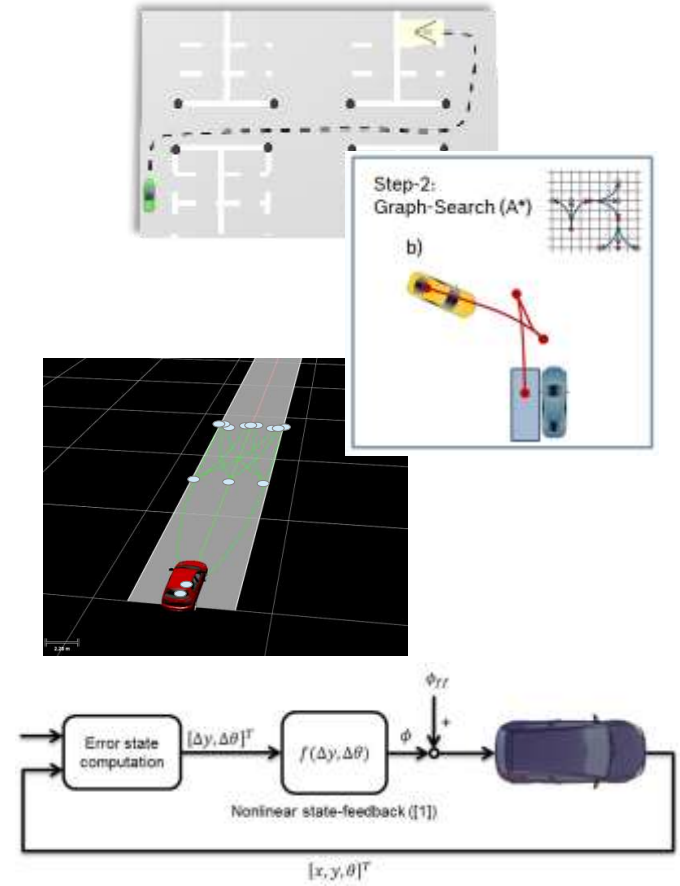


V-CHARGE

The V-Charge Car

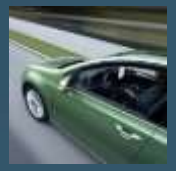


- **Hierarchical task decomposition**
  - **Mission planning on a topological map**
    - Graph-search on road-graph
  - **Free-Space/Parking planning**
    - State-lattice approach
  - **Online partial motion planning**
    - System-compliant tree-search guided by reference path
  - **Trajectory/Path controller**
    - Low-level nonlinear state-feedback controller with direct feed-forward feed-through

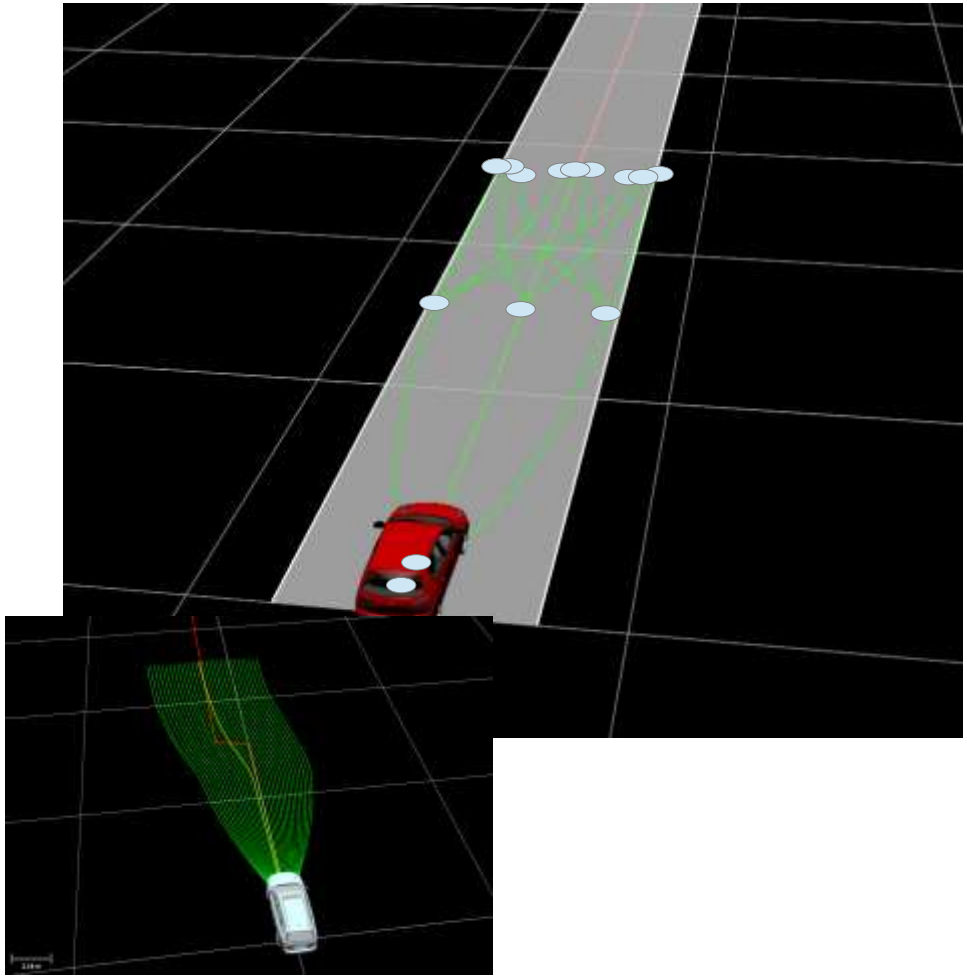


V-CHARGE

The V-Charge Car: Online Local Motion Planning

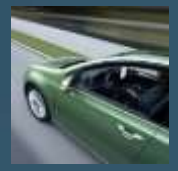


- Controlling simulated car towards a target state manifold
- Sample: $\boldsymbol{p} = [d, v, k_v]$
- Forward simulation of ODE with simulated controller in the loop
  - Controller regulates system towards sample from state manifold
  - Satisfy all vehicle limits
  - Able to handle discontinuity of the reference path
- Creating a tree of motions
  - Compensating for computation time
  - Accounting for system dead time
  - Simulating complex maneuvers with few parameters



U. Schwesinger; M. Rufli; P. Furgale; R. Siegwart: “A Sampling-Based Partial Motion Planning Framework for System-Compliant Navigation along a Reference Path”, *Intelligent Vehicles Symposium (IVS)*, 2013

The V-Charge Car: Online Local Motion Planning



- Minimization of cost function

$$\boldsymbol{u}^* = \min_{\boldsymbol{u}}(a_0 \int_{t_0}^{t_f} J_{lat}(t)dt + a_1 J_{lat}(t_f) + a_2 \int_{t_0}^{t_f} J_{lon}(t)dt + a_3 J_{lon}(t_f) + a_4 J_{obs} + a_5 J_{reg})$$

subject to

$$\dot{\boldsymbol{x}}(t) - \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t), t) = 0 \; \forall \; t$$

$$\boldsymbol{c}(t) \in C_{free} \forall \; t$$

$$\mathbf{u}_{\min} < \boldsymbol{u}(t) < \mathbf{u}_{\max} \; \forall \; t$$

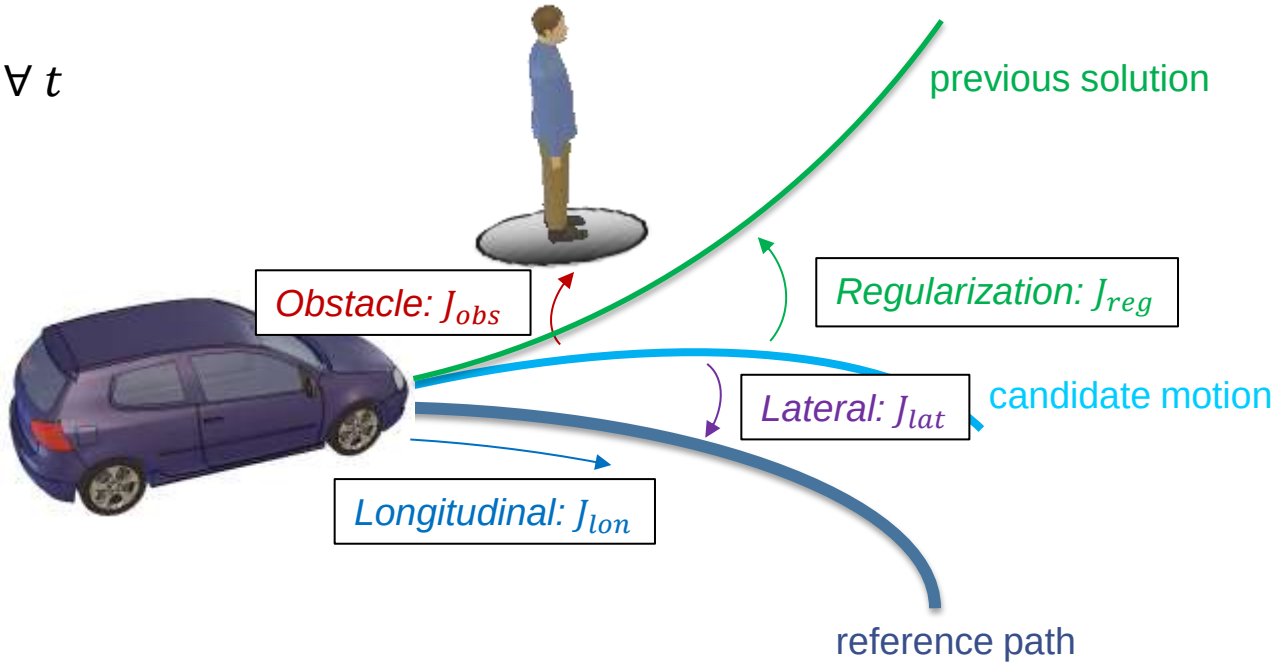
$$\boldsymbol{x}_{\min} < \boldsymbol{x}(t) < \boldsymbol{x}_{\max} \forall \; t$$

$$\boldsymbol{p} = [x, y]^T$$

$$\boldsymbol{c} = [\boldsymbol{p}, \theta]^T$$

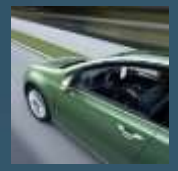
$$\boldsymbol{x} = [\boldsymbol{c}, v, \dot{v}, \ddot{v}, \phi, \dot{\phi}]$$

$$\boldsymbol{u} = [\phi, \dot{v}]^T$$



V-CHARGE

The V-Charge Car: Lateral Cost Term



$$\boldsymbol{u}^* = \min_{\boldsymbol{u}}(a_0 \int_{t_0}^{t_f} J_{lat}(t)dt + a_1 J_{lat}(t_f) + a_2 \int_{t_0}^{t_f} J_{lon}(t)dt + a_3 J_{lon}(t_f) + a_4 J_{obs} + a_5 J_{reg})$$

subject to

$\dot{\boldsymbol{x}}(t) - \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t), t) = 0 \; \forall \; t$

$\boldsymbol{c}(t) \in C_{free} \forall \; t$

$\mathbf{u}_{\min} < \boldsymbol{u}(t) < \mathbf{u}_{\max} \; \forall \; t$

$\boldsymbol{x}_{\min} < \boldsymbol{x}(t) < \boldsymbol{x}_{\max} \forall \; t$

$\boldsymbol{p} = [x, y]^T$

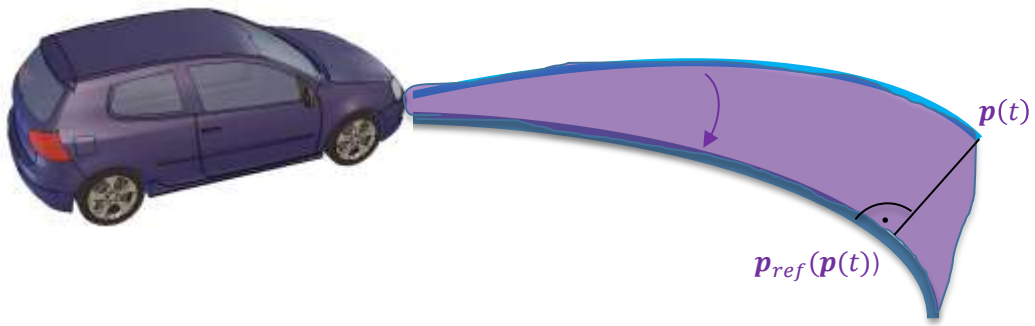
$\boldsymbol{c} = [\boldsymbol{p}, \theta]^T$

$\boldsymbol{x} = [\boldsymbol{c}, v, \dot{v}, \ddot{v}, \phi, \dot{\phi}]$

$\boldsymbol{u} = [\phi, \dot{v}]^T$

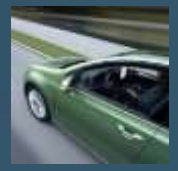
- Lateral cost term
  - Enforces small divergence from the reference path

$$J_{lat}(t) = ||\boldsymbol{p}(t) - \boldsymbol{p}_{ref}(\boldsymbol{p}(t))||$$



V-CHARGE

The V-Charge Car: Longitudinal Cost Term



$$\boldsymbol{u}^* = \min_{\boldsymbol{u}}(a_0 \int_{t_0}^{t_f} J_{lat}(t)dt + a_1 J_{lat}(t_f) + a_2 \int_{t_0}^{t_f} J_{lon}(t)dt + a_3 J_{lon}(t_f) + a_4 J_{obs} + a_5 J_{reg})$$

subject to

$\dot{\boldsymbol{x}}(t) - \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t), t) = 0 \; \forall \; t$

$\boldsymbol{c}(t) \in C_{free} \forall \; t$

$\mathbf{u}_{\min} < \boldsymbol{u}(t) < \mathbf{u}_{\max} \; \forall \; t$

$\boldsymbol{x}_{\min} < \boldsymbol{x}(t) < \boldsymbol{x}_{\max} \forall \; t$

$\boldsymbol{p} = [x, y]^T$

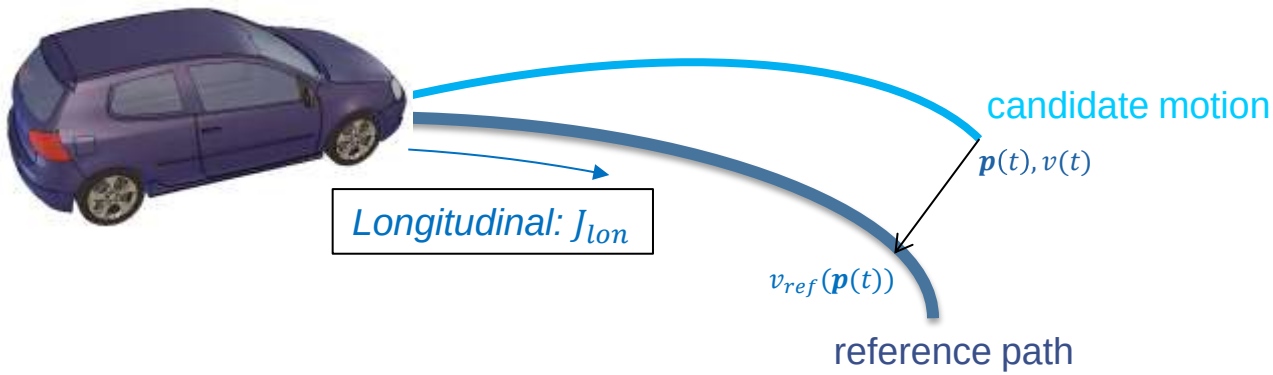
$\boldsymbol{c} = [\boldsymbol{p}, \theta]^T$

$\boldsymbol{x} = [\boldsymbol{c}, v, \dot{v}, \ddot{v}, \phi, \dot{\phi}]$

$\boldsymbol{u} = [\phi, \dot{v}]^T$

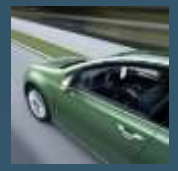
- Longitudinal cost term
  - Enforces progress along the reference path

$$J_{long} = \|v(t) - v_{ref}(\boldsymbol{p}(t))\|$$



V-CHARGE

The V-Charge Car: Obstacle Cost Term



$$\boldsymbol{u}^* = \min_{\boldsymbol{u}}(a_0 \int_{t_0}^{t_f} J_{lat}(t)dt + a_1 J_{lat}(t_f) + a_2 \int_{t_0}^{t_f} J_{lon}(t)dt + a_3 J_{lon}(t_f) + a_4 J_{obs} + a_5 J_{reg})$$

subject to

$\dot{\boldsymbol{x}}(t) - \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t), t) = 0 \; \forall \; t$

$\boldsymbol{c}(t) \in C_{free} \forall \; t$

$\mathbf{u}_{\min} < \boldsymbol{u}(t) < \mathbf{u}_{\max} \; \forall \; t$

$\boldsymbol{x}_{\min} < \boldsymbol{x}(t) < \boldsymbol{x}_{\max} \forall \; t$

$\boldsymbol{p} = [x, y]^T$

$\boldsymbol{c} = [\boldsymbol{p}, \theta]^T$

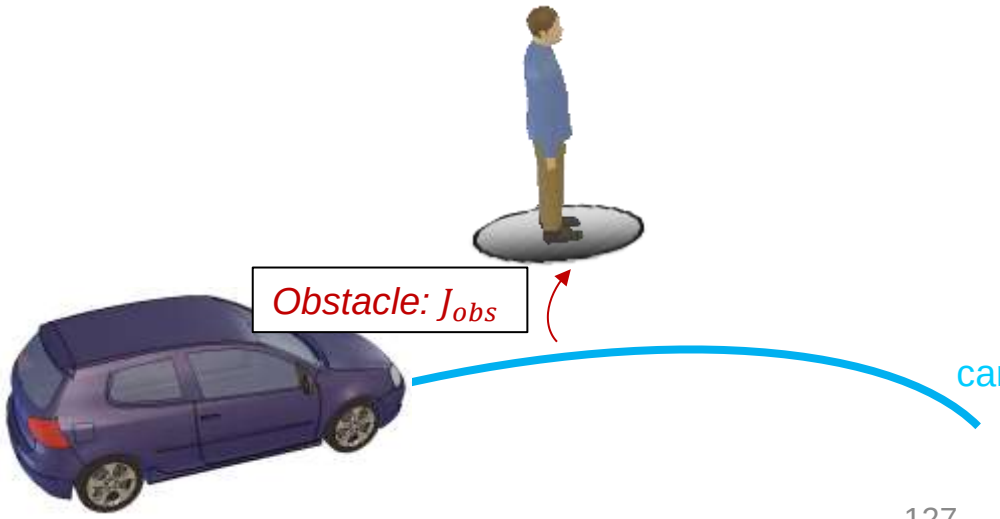
$\boldsymbol{x} = [\boldsymbol{c}, v, \dot{v}, \ddot{v}, \phi, \dot{\phi}]$

$\boldsymbol{u} = [\phi, \dot{v}]^T$

- Obstacle cost term
  - Penalizes close distances to obstacles
  - Important to deal with sensor noise and uncertainties in system model

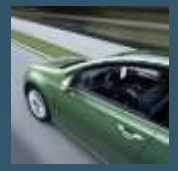
$J_{obs} = (1.0 - \min(d_{obs}(t), \epsilon)/\epsilon)$

$\epsilon > 0$: Penalization cutoff



V-CHARGE

The V-Charge Car: Regularization Cost Term



$$\boldsymbol{u}^* = \min_{\boldsymbol{u}}(a_0 \int_{t_0}^{t_f} J_{lat}(t)dt + a_1 J_{lat}(t_f) + a_2 \int_{t_0}^{t_f} J_{lon}(t)dt + a_3 J_{lon}(t_f) + a_4 J_{obs} + a_5 J_{reg})$$

subject to

$$\dot{\boldsymbol{x}}(t) - \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t), t) = 0 \; \forall \; t$$

$$\boldsymbol{c}(t) \in C_{free} \forall \; t$$

$$\mathbf{u}_{\min} < \boldsymbol{u}(t) < \mathbf{u}_{\max} \; \forall \; t$$

$$\boldsymbol{x}_{\min} < \boldsymbol{x}(t) < \boldsymbol{x}_{\max} \forall \; t$$

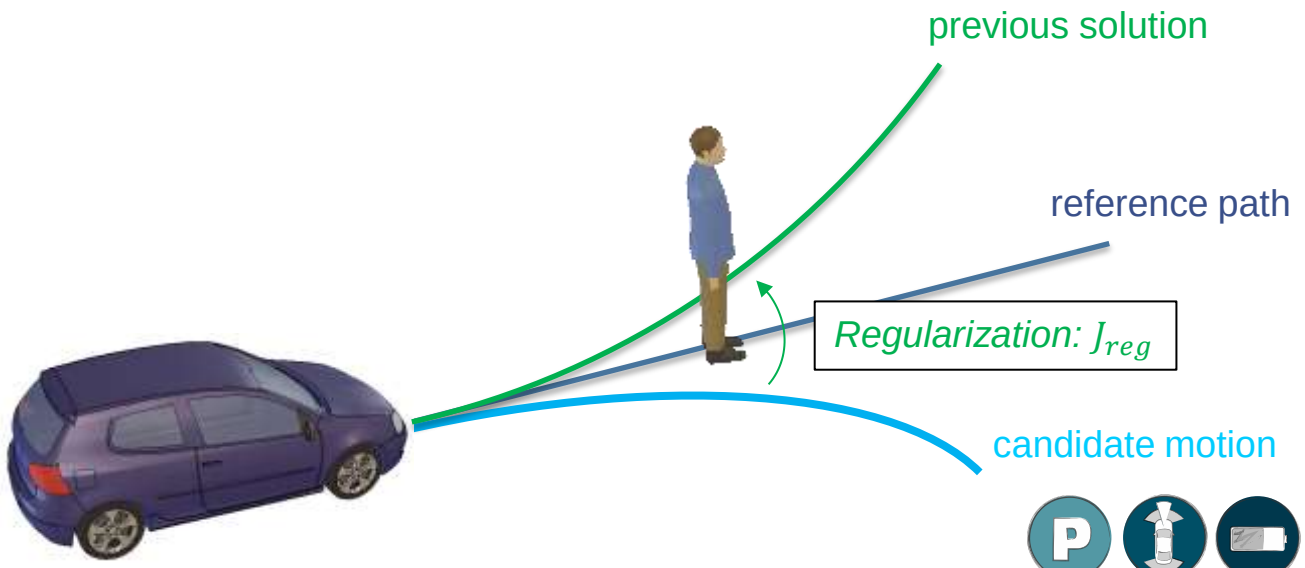
$$\boldsymbol{p} = [x, y]^T$$

$$\boldsymbol{c} = [\boldsymbol{p}, \theta]^T$$

$$\boldsymbol{x} = [\boldsymbol{c}, v, \dot{v}, \ddot{v}, \phi, \dot{\phi}]$$

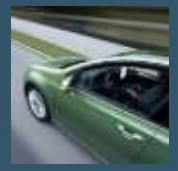
$$\boldsymbol{u} = [\phi, \dot{v}]^T$$

- Regularization cost term
  - Penalizes distance to previous solution
  - Avoids temporal oscillations between nearby local minima



V-CHARGE

The V-Charge Car: Normalization of cost terms



- Normalization of cost terms
  - Simplifies tuning of weights

$$\boldsymbol{u}^* = \min_{\boldsymbol{u}}(\underbrace{a_0 \int_{t_0}^{t_f} J_{lat}(t)dt}_{[0,1]} + \underbrace{a_1 J_{lat}(t_f)}_{[0,1]} + \underbrace{a_2 \int_{t_0}^{t_f} J_{lon}(t)dt}_{[0,1]} + \underbrace{a_3 J_{lon}(t_f)}_{[0,1]} + \underbrace{a_4 J_{obs}}_{[0,1]} + \underbrace{a_5 J_{reg}}_{[0,1]})$$

Overview

- The Basic Motion Planning Problem
  - I/O
  - Notions and Terms
- Selected algorithms for motion planning
  - The Dynamic Window Approach
  - Graph-Search, State-Lattices
  - Randomized Approaches: RRT, RRT*
- Collision Detection
- Detailed case studies from DARPA Urban Challenge
  - Boss (Carnegie Mellon)
  - AnnieWay (Karlsruhe/Munich)
  - Junior (Stanford)
- The V-Charge Motion Planning Framework
- Getting started for the technical session

- Objectives
  - Design a cost function to „score“ candidate trajectories
- Inputs
  - Set of candidate trajectories
    - List of tuples of (state + system inputs, collision flag, closest distance to any static obstacle)
  - Reference path distance map
  - Last solution trajectory

V-CHARGE

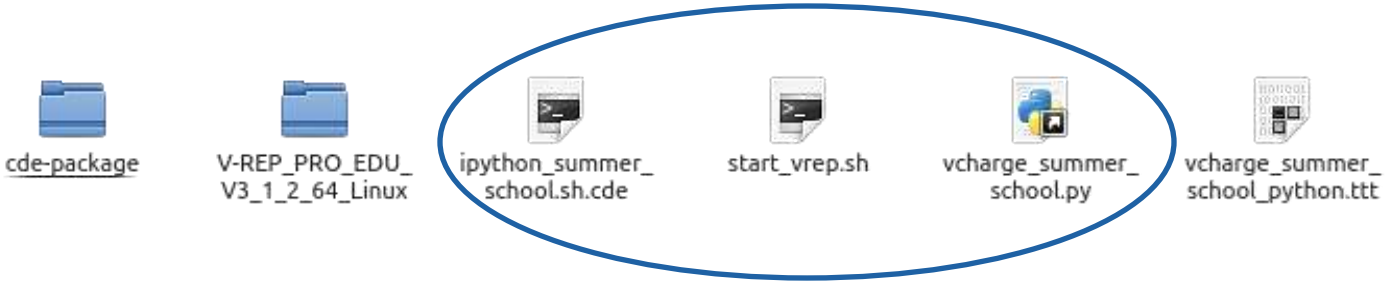
Technical Session: Motion Planning

```
#!/usr/bin/python

...

# TODO: Some parameters for you to play around with
parameters = {'useAngularRateForMotionPredictions': True,
              'plotTrajectorySetInVrep': True,
              'plotInPython': False }

...
Tracer()() # creates a breakpoint! Comment out to run the program seamlessly
...

def getBestTrajectory(trajectorySet, pathDistanceMap, lastSolution):
    ...
    # TODO: Fill in the weight vector here
    nCostTerms = 1
    weights = np.ones( shape=(1,nCostTerms) )
    ...

    # Compute cost for every trajectory
    for cnt,(trajectory,collisionFlag,closestDistanceToStaticObstacles) in enumerate(trajectorySet):
        # Only for candidate trajectories not in collision
        if not np.isinf(collisionFlag):

            # TODO: Fill in code here
            # ...
            costs[cnt, :] = np.zeros(shape=(weights.size,)) # TODO: fill this in correctly
        ...

...
```


V-CHARGE

Technical Session: Motion Planning

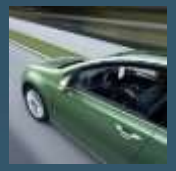
files of interest



- **./start_vrep.sh**
  - Starts the simulator
  - No need to start the simulation, done by the python script
- **./ipython_summer_school.cde**
  - Will launch ipython in the CDE virtualization environment
  - In ipython, type **run bin/vcharge_summer_school_python.py** to run your program
- **<editor> vcharge_summer_school.py**
  - Will open the python program to modify in your favorite editor

V-CHARGE

Thank you for your attention



V-CHARGE