Motion Planning with Differential Constraints

TSFS12: Autonomous Vehicles – Planning, Control, and Learning Systems

Lecture 4: Björn Olofsson <bjorn.olofsson@liu.se>

LINKÖPING UNIVERSITY

Purpose of this Lecture

- Give a background on **methods for motion planning** of systems with **differential motion constraints**.
- Study how **rapidly-exploring random trees (RRTs)** and extensions of such strategies can be used for motion planning.
- **Extend** the graph-search methods from Lecture 2 to autonomous vehicles with **differential motion constraints** using **lattice planning**.
- *Extra material*: Study how **path-constrained trajectory-planning problems** can be solved.

Expected Take-Aways from this Lecture

- Knowledge about how important **differential motion constraints** can be considered in motion planning.
- Be familiar with commonly used **methods for motion planning** targeting **autonomous vehicles**.
- Knowledge about some available **computer tools and libraries** for motion planning.

Literature Reading

The following book and article sections are the main reading material for this lecture. References to further reading are provided throughout the slides and at the end of the lecture slides.

- Sections 5.1-5.5 and 14.1-14.4 in LaValle, S. M: *Planning Algorithms*. Cambridge University Press, 2006.
- Sections 3.3.3 and 5 in Karaman, S., & E. Frazzoli: "Sampling-based algorithms for optimal motion planning". *The International Journal of Robotics Research*, 30(7), 846-894, 2011.
- Section 4.5 in Bergman, K: "Exploiting Direct Optimal Control for Motion Planning in Unstructured Environments ", Ph.D. Thesis No. 2133, Div. Automatic Control, Linköping Univ., 2021.

Outline of the Lecture

- The **motion-planning problem**.
- Methods:
  - **Rapidly-exploring random trees** (RRTs) for motion planning.
  - Planning using **motion primitives** and **state lattices**.
- **Tools and libraries** for motion planning.
- *Extra material*: **Decoupled approaches** to motion planning.

Context in the Architecture for an Autonomous Vehicle

In this lecture the focus will be on the motion-planning layer in the decision-making architecture.

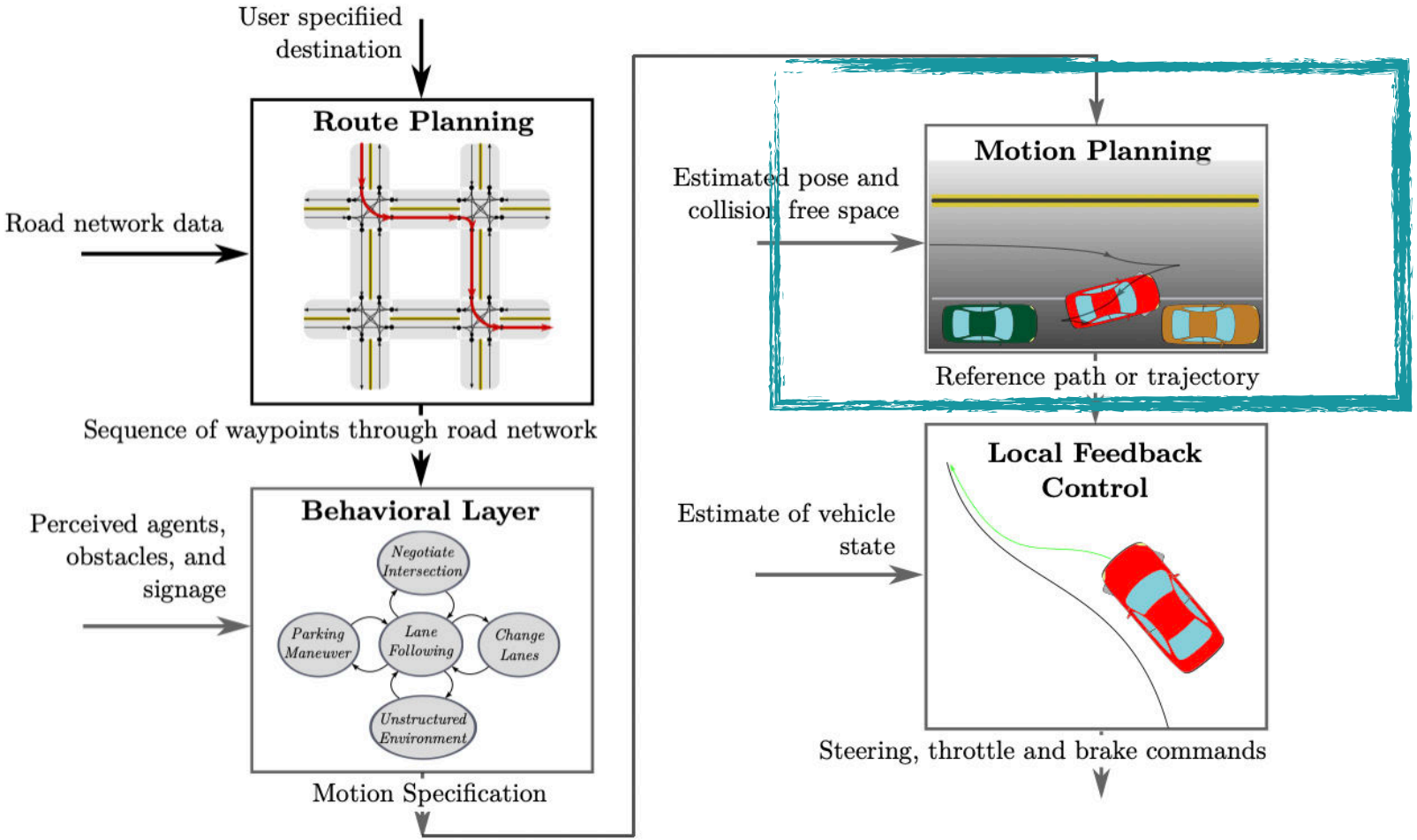
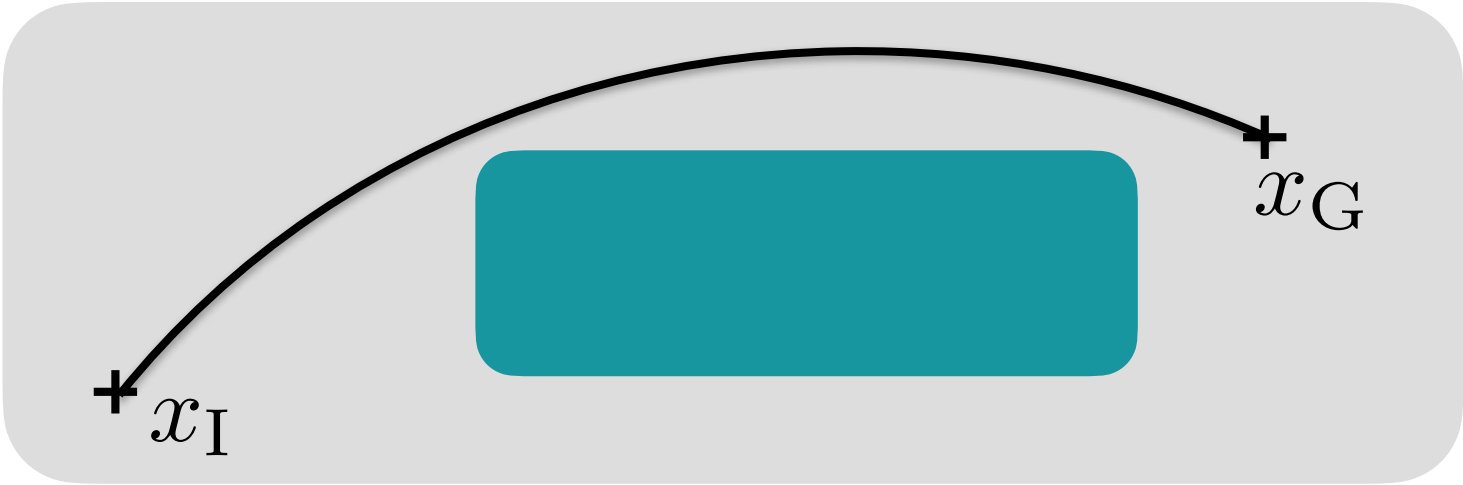


Figure from: Paden, B., Čáp, M., Yong, S. Z., Yershov, D., & Frazzoli, E: "A survey of motion planning and control techniques for self-driving urban vehicles". IEEE Transactions on Intelligent Vehicles, 1(1), 33-55, 2016.

The Motion-Planning Problem

Motion Planning – Introduction (1/3)

- Compute a strategy for transferring a vehicle **from an initial state to another desired state**.
- The state could be the **position**, **orientation**, and **derivatives thereof**, for the vehicle.



Motion Planning – Introduction (2/3)

- Constraints to be fulfilled during the motion (**feasibility**):
  - The **motion equations** of the vehicle.
  - **Obstacles**, **humans**, and **other vehicles** (stationary or time-varying).
  - **Physical limits** on control inputs (e.g., motor power in a car, maximum torque in a robot motor).

Motion Planning – Introduction (3/3)

- Constraints to be fulfilled during the motion (**feasibility**), cont'd:
  - Constraints on **internal states** (e.g., velocity, available friction between tires and road, torque transfer in clutches).
  - **Road constraints**.
- Often also desirable to **minimize** energy/power consumption, time, path length, or other performance criteria (**optimality**).

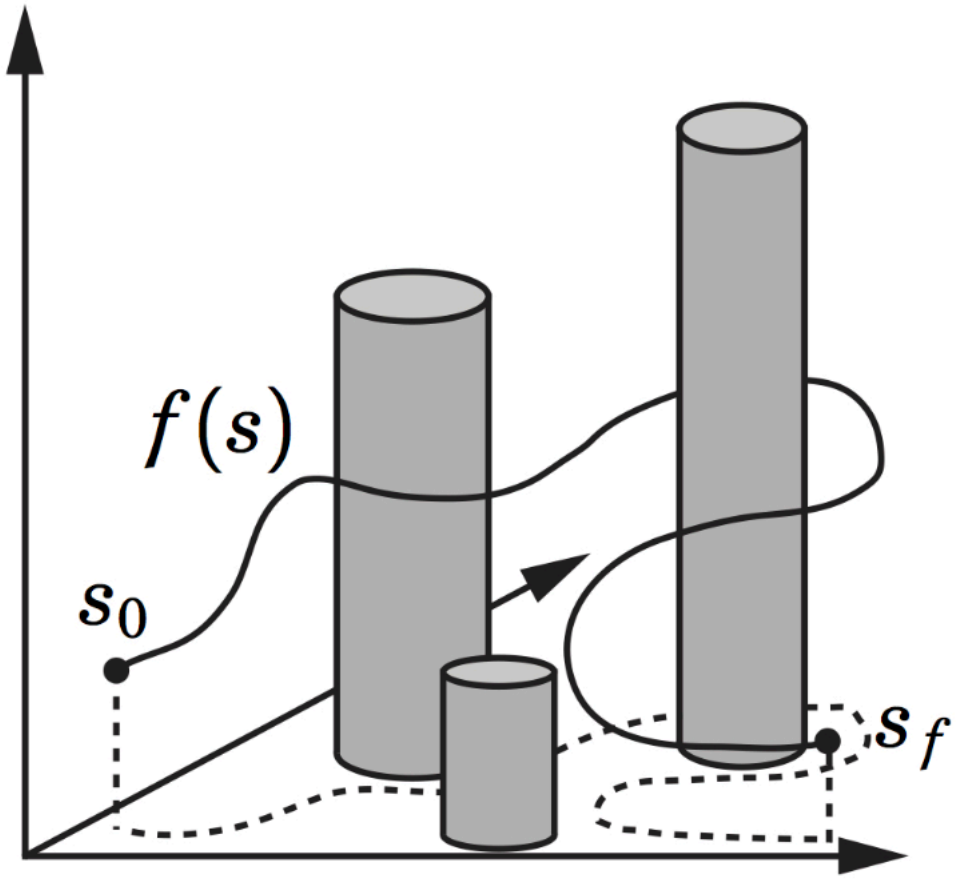
Some Important Terms and Concepts

- **Path**: A **curve in space** defining the geometric motion.

$$f(s) \in \mathbb{R}^n, \quad s \in [s_0, s_f]$$

- **Trajectory**: Is a path with a **time-parameterization** connected to it (i.e., also specifies the **velocity** or evolution of time along the path).

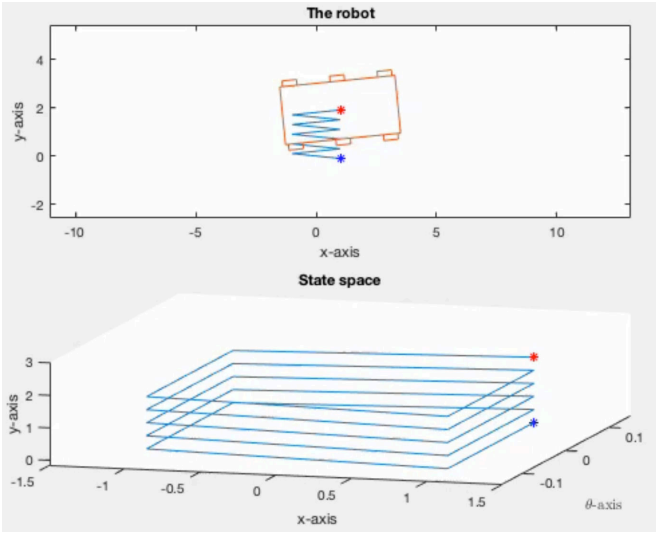
$$f(s(t)), \quad t \in [t_0, t_f]$$



Nilsson, H., and B. Olofsson: "Optimal tracking and identification of paths for industrial robots". Dept. Automatic Control, Lund Univ., 2010.

Motion Planning – Compare Lectures 2 & 3

- In Lecture 2, **discrete motion planning** was considered (example from Hand-in Exercise 1 to the right).
- In this lecture, the **differential motion constraints** are also considered.
- Recall the motion needed to make **parallel transitions** with a car from Lecture 3 (***cf.* parallel parking**).



Differential Motion Constraints

- As seen in Lecture 3 on modeling of autonomous vehicles, many systems can be described by **differential equations** on the format:

$$\dot{x} = g(x, u) \qquad \begin{matrix} x - \text{states} \\ u - \text{inputs} \end{matrix}$$

- An essential task in **high-performance motion planning** is to take this constraint into account to make the motion plan feasible.

The Configuration Space

- The manifold defining the set of possible transformations of a vehicle from one state to another is called the **configuration space** $\mathcal{C}$.
- The configuration space can be viewed as the **state space** in which the vehicle moves.
- The search for a **feasible motion plan** is done in the configuration space $\mathcal{C}$.

LaValle, S. M: Planning Algorithms. Cambridge University Press, 2006.

The Obstacle Region and the Free Space

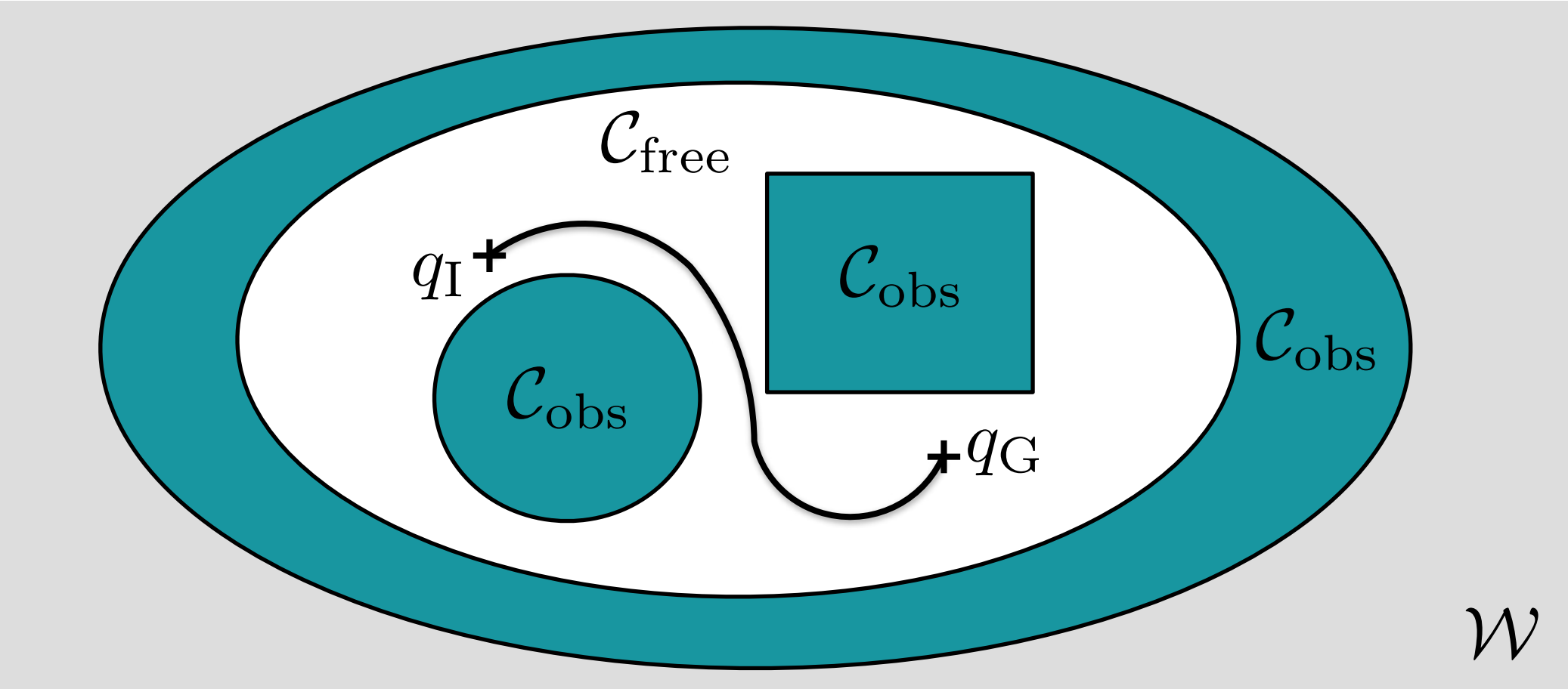
- Assume a **world** $\mathcal{W}$ and an **obstacle region** $\mathcal{O}$. Further, let the **configuration** of the autonomous vehicle $\mathcal{AV}$ be defined by q in the configuration space $\mathcal{C}$.
- The **obstacle region** $\mathcal{C}_{\text{obs}}$ can then be defined as:

$$\mathcal{C}_{\text{obs}} = \{q \in \mathcal{C} \mid \mathcal{AV}(q) \cap \mathcal{O} \neq \emptyset\}$$

- The **free space** is accordingly defined as $\mathcal{C}_{\text{free}} = \mathcal{C} \setminus \mathcal{C}_{\text{obs}}$.

LaValle, S. M: Planning Algorithms. Cambridge University Press, 2006.

Conceptual Illustration of the Configuration Space for Planning



The State Space

- The **configuration space** considers only the configuration q of the system.
- To consider also **higher-order derivatives** required for differential constraints (e.g., velocity and acceleration), the **phase space** is used.
- The phase space $\mathcal{X}$ has **higher dimension** than the configuration space (typically both position and velocity variables).
- In **motion planning**, the state space is the configuration space or the phase space, depending on the planning task.

LaValle, S. M: Planning Algorithms. Cambridge University Press, 2006.

Example: The State Space for a Double Integrator

- The differential equation for a **double integrator** with the input u is

$$\ddot{q} = u$$

- With the states $x = (q, \dot{q})$ and the state space $\mathcal{X} = \mathbb{R}^2$, the **state-space dynamic equations** for the double integrator can be written as

$$\dot{x}_1 = x_2,$$

$$\dot{x}_2 = u$$

Obstacle Region in the Phase Space

- The **obstacle region** in the phase space is defined as

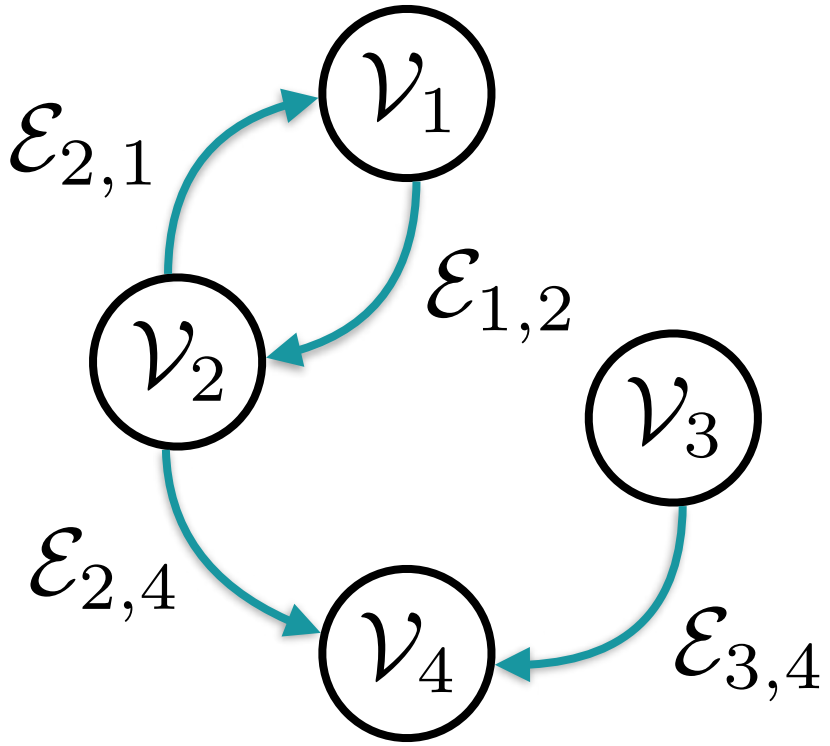
$$\mathcal{X}_{\text{obs}} = \{x \in \mathcal{X} \mid \Xi(x) \in \mathcal{C}_{\text{obs}}\}$$

- The mapping Ξ gives the **configuration variables** from the complete set of states and $\mathcal{X}_{\text{free}} = \mathcal{X} \setminus \mathcal{X}_{\text{obs}}$.
- If the state space is the configuration space, then it **coincides** with $\mathcal{C}_{\text{obs}}$.

LaValle, S. M: Planning Algorithms. Cambridge University Press, 2006.

Directed Graphs for Motion Planning

- A **directed graph** $\mathcal{G}$, comprising **vertices** $\mathcal{V}$ and **edges** $\mathcal{E}$.
- The direction of an edge between two vertices indicates the **transition direction**.
- A **vertex** is also commonly known as a **node**.



The Path-Planning Problem

- Let q_{I} be the **initial configuration** and q_{G} the **desired goal configuration**.
- The **path-planning problem** consists of computing a path f for the autonomous vehicle $\mathcal{AV}$ according to

$$f : [0, 1] \rightarrow \mathcal{C}_{\mathrm{free}}, \text{ with } f(0) = q_{\mathrm{I}}, \ f(1) = q_{\mathrm{G}}$$

- The path must **only reside in** $\mathcal{C}_{\mathrm{free}}$.

LaValle, S. M: Planning Algorithms. Cambridge University Press, 2006.

The Motion-Planning Problem w. Differential Constraints

- Let x_{I} be the **initial state** and x_{G} the desired **goal state**.
- The motion-planning problem **with differential constraints** is to compute the control inputs $u(t)$ for the vehicle $\mathcal{AV}$ such that the corresponding state trajectory $x(t)$ satisfies

$$x(t) \in \mathcal{X}_{\mathrm{free}}, \quad x(0) = x_{\mathrm{I}}, \quad x(T) = x_{\mathrm{G}}$$

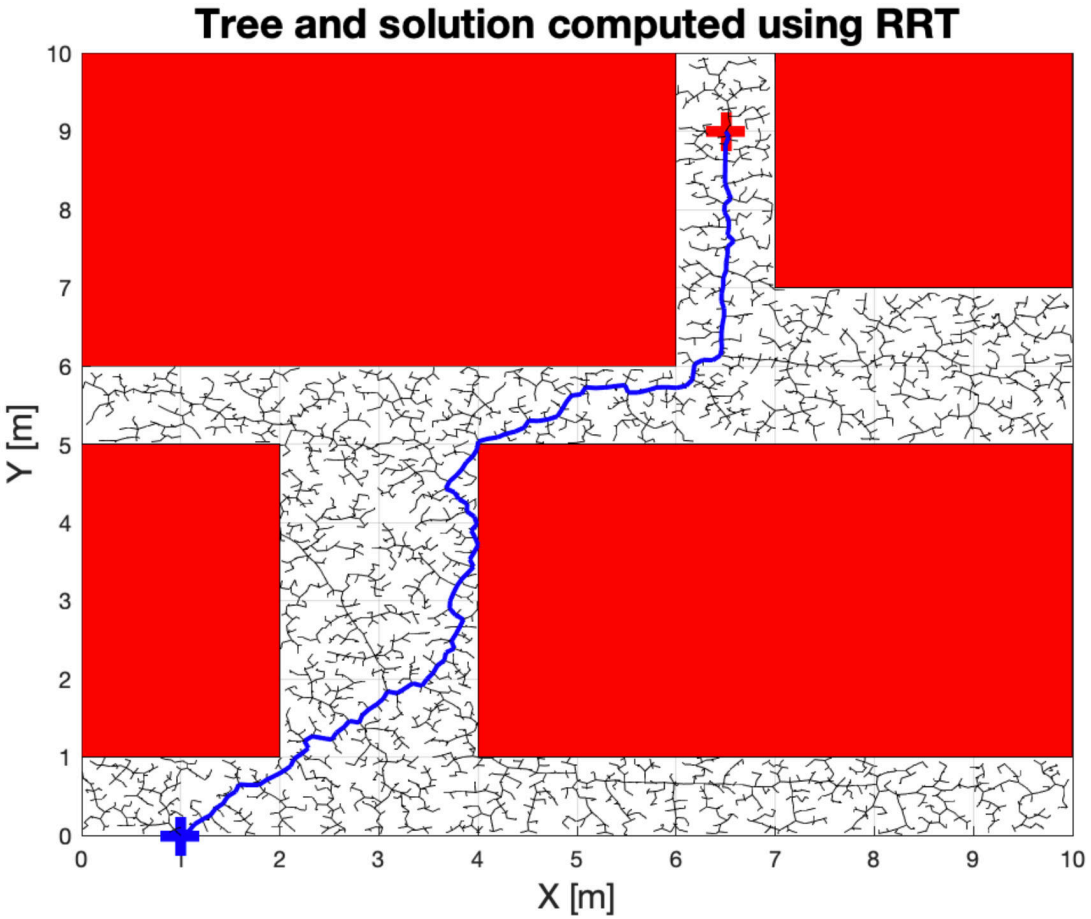
- Often **constraints** on $u(t)$ and other **specifications** on $x(t)$.

LaValle, S. M: Planning Algorithms. Cambridge University Press, 2006.

Rapidly-Exploring Random Trees (RRTs) for Motion Planning

The Basic Idea of RRTs (1/2)

- The basic idea behind **rapidly-exploring random trees** (RRTs) is to **incrementally construct** a tree in the state space by **sampling**.
- A **tree** is a graph where any two vertices are connected by **exactly one path** (each **vertex** corresponds to a **state** and the **edge** is the **feasible motion segment** between two vertices).



LaValle, S. M., & J. J. Kuffner Jr: "Randomized kinodynamic planning". The International Journal of Robotics Research, 20(5), 378-400, 2001.

The Basic Idea of RRTs (2/2)

- The **sampling** can be **random** or any other strategy leading to a **dense sequence of samples**.
- RRT computes a **feasible path**, whereas **RRT*** also aims to **minimize a cost** associated with the motion.

LaValle, S. M., & J. J. Kuffner Jr: "Randomized kinodynamic planning". The International Journal of Robotics Research, 20(5), 378-400, 2001.
Karaman, S., & E. Frazzoli: "Sampling-based algorithms for optimal motion planning". The International Journal of Robotics Research, 30(7), 846-894, 2011.

Sampling in the State Space

- Construction of the tree relies on **efficient sampling** of the state space ($\mathcal{C}_{\text{free}}$ or $\mathcal{X}_{\text{free}}$).
- **Sequence of the samples** important, not only the set of samples.
- Sampling can be both **random** and **deterministic**.
- Special attention required when sampling in **orientation spaces** such as the 3D rotation group $\text{SO}(3)$.

Basic Version of RRT (1/2)

Algorithm 1: RRT w/o. Differential Constraints

```
1  V ← {q_I}, E ← ∅;
2  for i = 1, ..., N do:
3      q_rand ← Sample;
4      q_nearest ← Nearest(G = (V, E), q_rand);
5      q_new ← Steer(q_nearest, q_rand);
6      if ObstacleFree(q_nearest, q_new) then
7          V ← V ∪ {q_new};
8          E ← E ∪ {(q_nearest, q_new)};
9  return G = (V, E);
```


LaValle, S. M., & J. J. Kuffner Jr: "Randomized kinodynamic planning". The International Journal of Robotics Research, 20(5), 378-400, 2001.

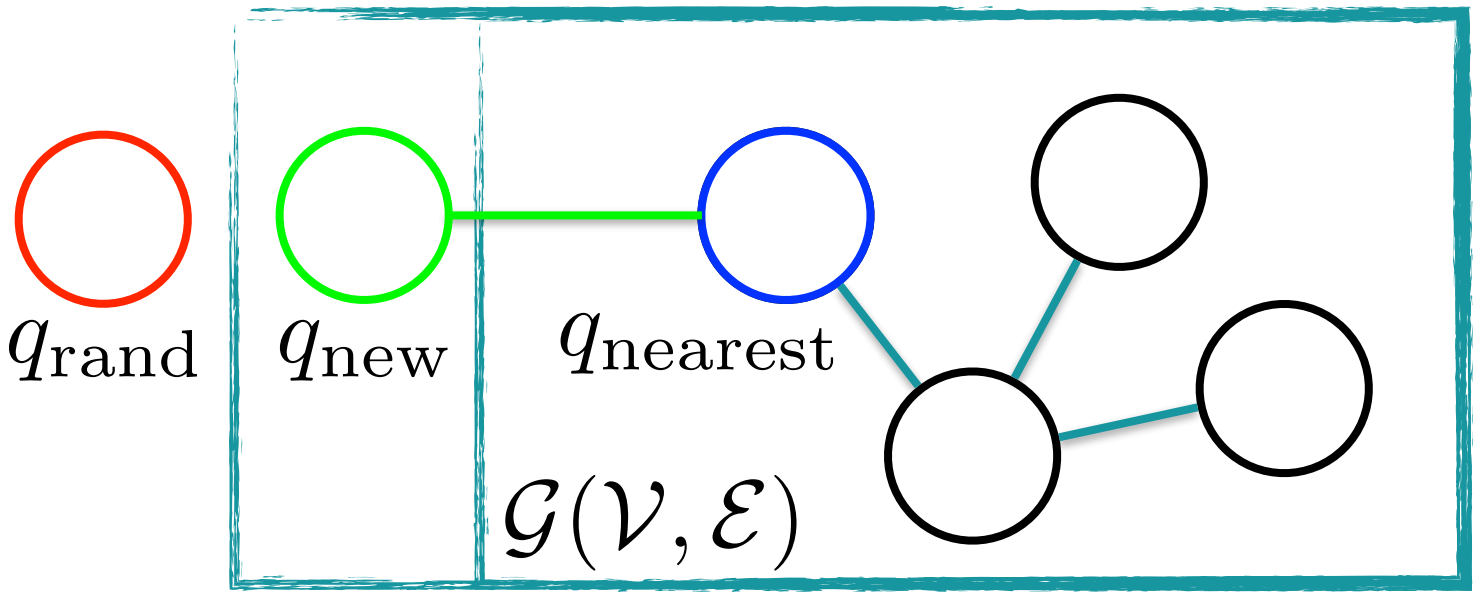
Basic Version of RRT (2/2)

- **Sample**: Gives a sample in the free state space.
- **Nearest**: Provides the vertex in the tree that is closest to the sampled state.
- **Steer**: In general this is a so called two-point boundary value problem (TPBV). Construct a path from the nearest vertex towards the sampled state, often with a maximum step length (alternative strategies exist).
- **ObstacleFree**: Checks whether the path from the closest vertex in the graph to the new state is collision free.

RRT with Biased Sampling

- To speed up the process of reaching the **goal state**, a **bias** towards it can be introduced in the sampling.
- In practice: with a certain **pre-defined probability** β the sampled state is the goal state.
- In general, the sampling could be biased to be, e.g., in the **direction of the road** or **directions implied by the environment geometry**.

RRT – Example w/o. Obstacles



Sample

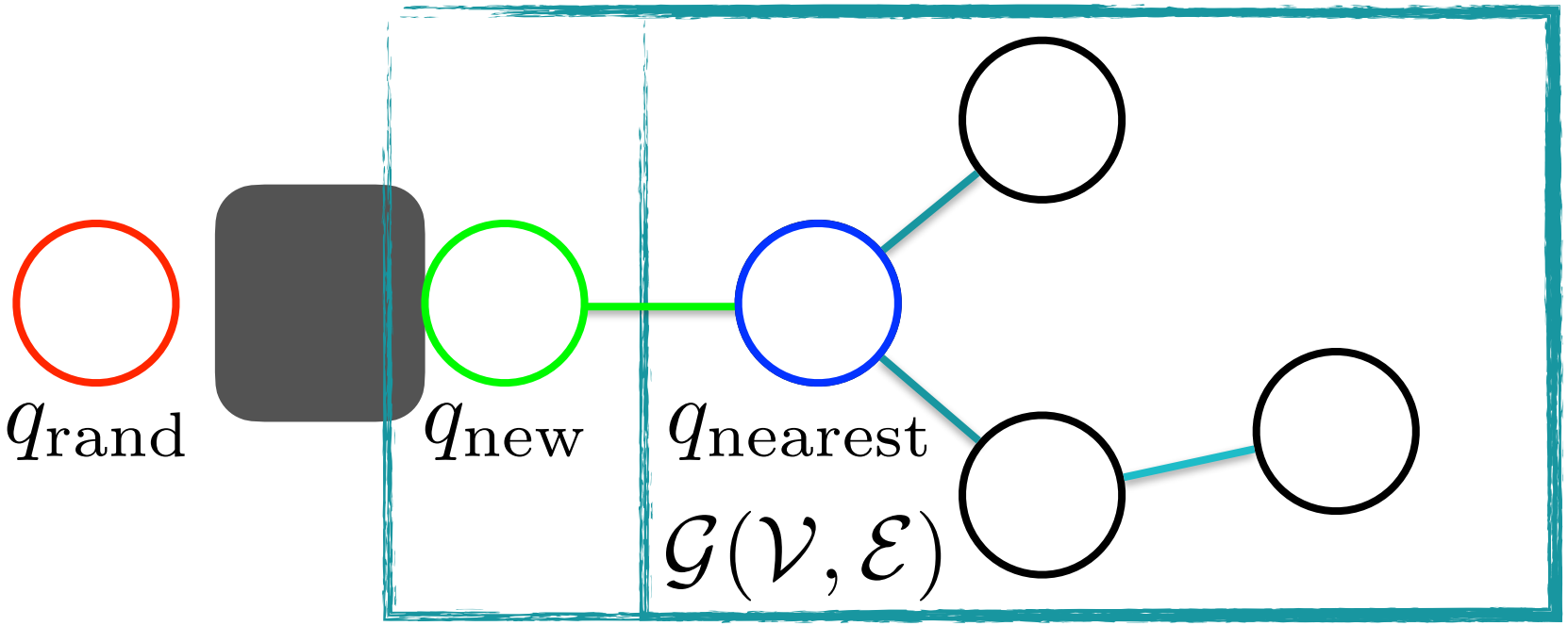
Nearest

Steer

ObstacleFree

RRT – Example w. Obstacle

The steer function moves forward towards the sampled node until the obstacle is detected.



Sample

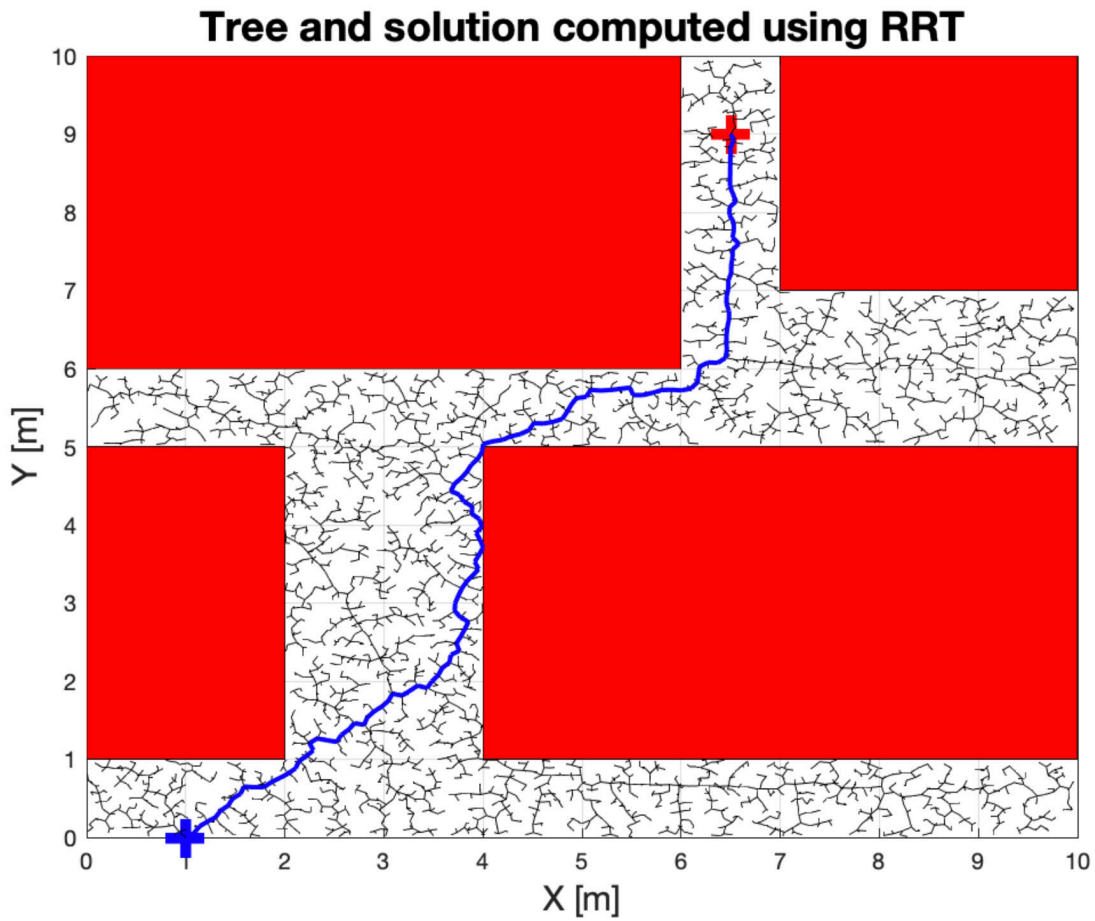
Nearest

Steer

ObstacleFree

RRT – Example

- Plan a **path** from the initial state (blue cross) to the goal state (red cross) using **RRT**.
- The red boxes are **obstacles**.
- A particle moving in a 2D world.
- How to handle **differential motion constraints**?



RRT

RRT with Differential Motion Constraints (1/2)

Algorithm 2: RRT w. Differential Constraints

$\mathcal{V} \leftarrow \{x_I\}$, $\mathcal{E} \leftarrow \emptyset$;
for $i = 1, \ldots, N$ **do**:
  $x_{\text{rand}} \leftarrow \text{Sample}$;
  $x_{\text{nearest}} \leftarrow \text{Nearest}(\mathcal{G} = (\mathcal{V}, \mathcal{E}), x_{\text{rand}})$;
  $(x_{\text{new}}, \tilde{u}^p) \leftarrow \text{Steer}(x_{\text{nearest}}, x_{\text{rand}})$;
  if $\text{ObstacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
    $\mathcal{V} \leftarrow \mathcal{V} \cup \{x_{\text{new}}\}$;
    $\mathcal{E} \leftarrow \mathcal{E} \cup \{(x_{\text{nearest}}, x_{\text{new}}; \tilde{u}^p)\}$;
return $\mathcal{G} = (\mathcal{V}, \mathcal{E})$;

LaValle, S. M., & J. J. Kuffner Jr: "Randomized kinodynamic planning". The International Journal of Robotics Research, 20(5), 378-400, 2001.

RRT with Differential Motion Constraints (2/2)

- **Sample**: Gives a sample in the free state space.
- **Nearest**: Provides the vertex in the tree that is closest to the sampled state.
- **Steer**: Apply the input $\tilde{u}^p$ from a pre-defined discrete set of possible control input trajectories, leading as close to the sampled state as possible.
  - The input trajectories are often of the same time duration.
  - A simulation of the motion equations can be used to compute the trajectories obtained with the different inputs.
- **ObstacleFree**: Checks whether the path from the closest vertex to the new state is collision free.

Introduction to RRT*

- **RRT*** is an extension of RRT targeting **optimal sampling-based motion planning**.
- Compared to RRT, **more steps** are performed **each time a new vertex** is added to the tree.
- In this lecture, **RRT* without differential motion constraints** are considered.
- **Extensions** covering differential motion constraints exist, but are not covered in the lectures, however, a possible mini-project.

Karaman, S., & E. Frazzoli: "Sampling-based algorithms for optimal motion planning". The International Journal of Robotics Research, 30(7), 846-894, 2011.

RRT* – Optimal Motion Planning (1/2)

Algorithm 3: RRT* w/o. Differential Constraints

```
1  V ← {q_I}, E ← ∅;
2  for i = 1, ..., N do:
3      q_rand ← Sample;
4      q_nearest ← Nearest(G = (V, E), q_rand);
5      q_new ← Steer(q_nearest, q_rand);
6      if ObstacleFree(q_nearest, q_new) then
7          Q_near ← Near(G = (V, E), q_new, γ);
8          V ← V ∪ {q_new};
9          q_min ← q_nearest; c_min ← Cost(q_nearest) + CostL(Line(q_nearest, q_new));
10         foreach q_near ∈ Q_near do
11             if CollisionFree(q_near, q_new) &&
12             Cost(q_near) + CostL(Line(q_near, q_new)) < c_min then
13                 q_min ← q_near; c_min ← Cost(q_near) + CostL(Line(q_near, q_new));
14         E ← E ∪ {(q_min, q_new)};
15         foreach q_near ∈ Q_near do
16             if CollisionFree(q_new, q_near) &&
17             Cost(q_new) + CostL(Line(q_new, q_near)) < Cost(q_near) then
18                 q_parent ← Parent(q_near);
19                 E ← (E \ {(q_parent, q_near)}) ∪ {(q_new, q_near)};
20 return G = (V, E);
```

Use path with minimum cost for connecting the new state to the tree.

Re-wire the vertices in a neighborhood of the new state that can reduce the cost.

Karaman, S., & E. Frazzoli: "Sampling-based algorithms for optimal motion planning". The International Journal of Robotics Research, 30(7), 846-894, 2011.

RRT* – Optimal Motion Planning (2/2)

- Same functions as in the RRT algorithm, but with certain additional functions:
  - **Near**: Provides all vertices within a ball with radius γ in the tree.
  - **Cost**: The cost-to-come to the current vertex from the tree root (initial state).
  - **CostL**: The cost of moving along a straight line from the specified vertices.
  - **Line**: Gives the path between two vertices as a straight line.
  - **Parent**: Gives the vertex that is parent to the current vertex in the tree.
- Note that in RRT* **more steps are done** each time a **new state** is obtained:
  - Make sure that the new vertex is connected along the path with **minimum cost**.
  - **Re-wire** the existing tree so as to try to **minimize the cost** gradually.

RRT* – Example

Sample

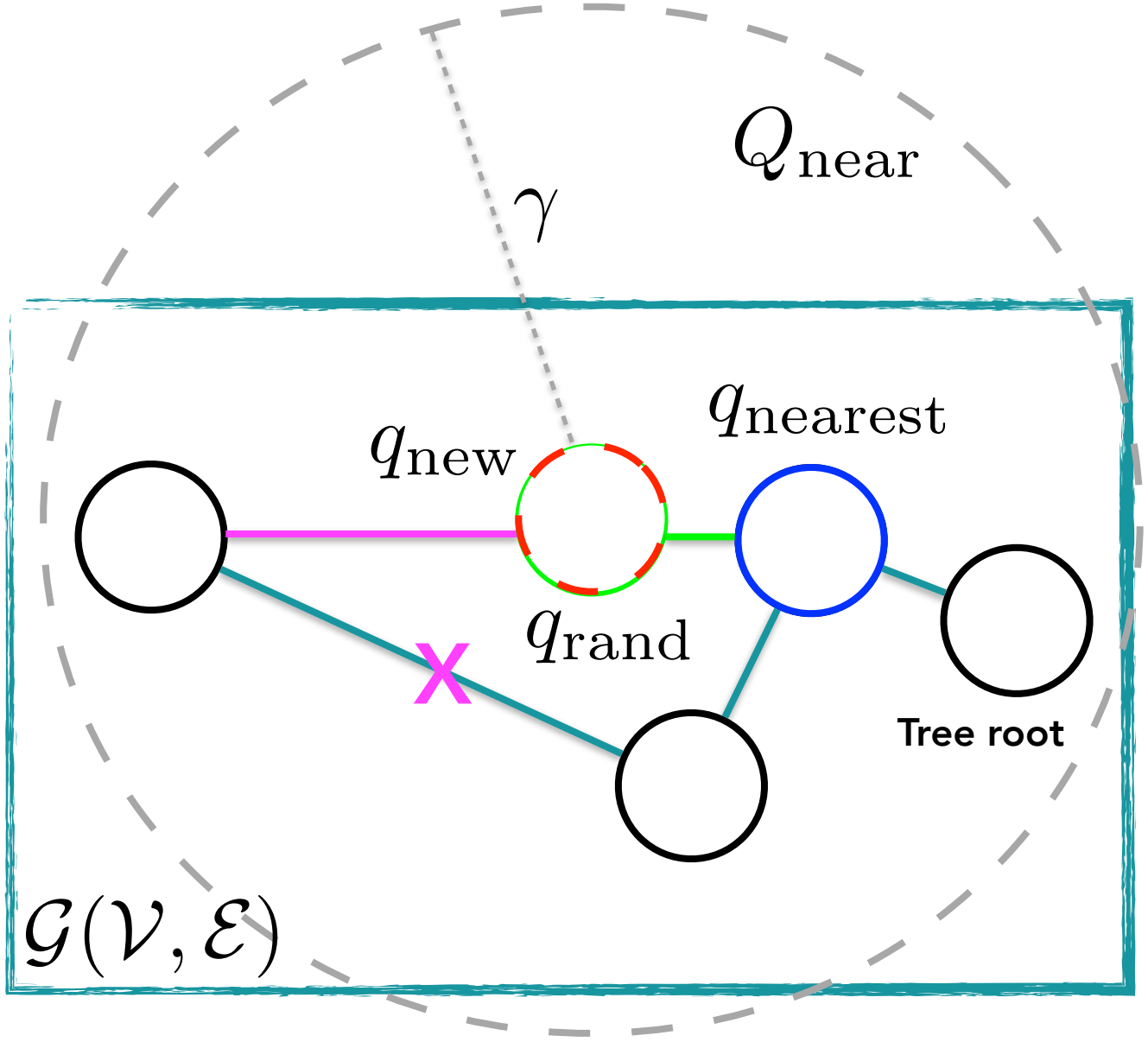
Nearest

Steer

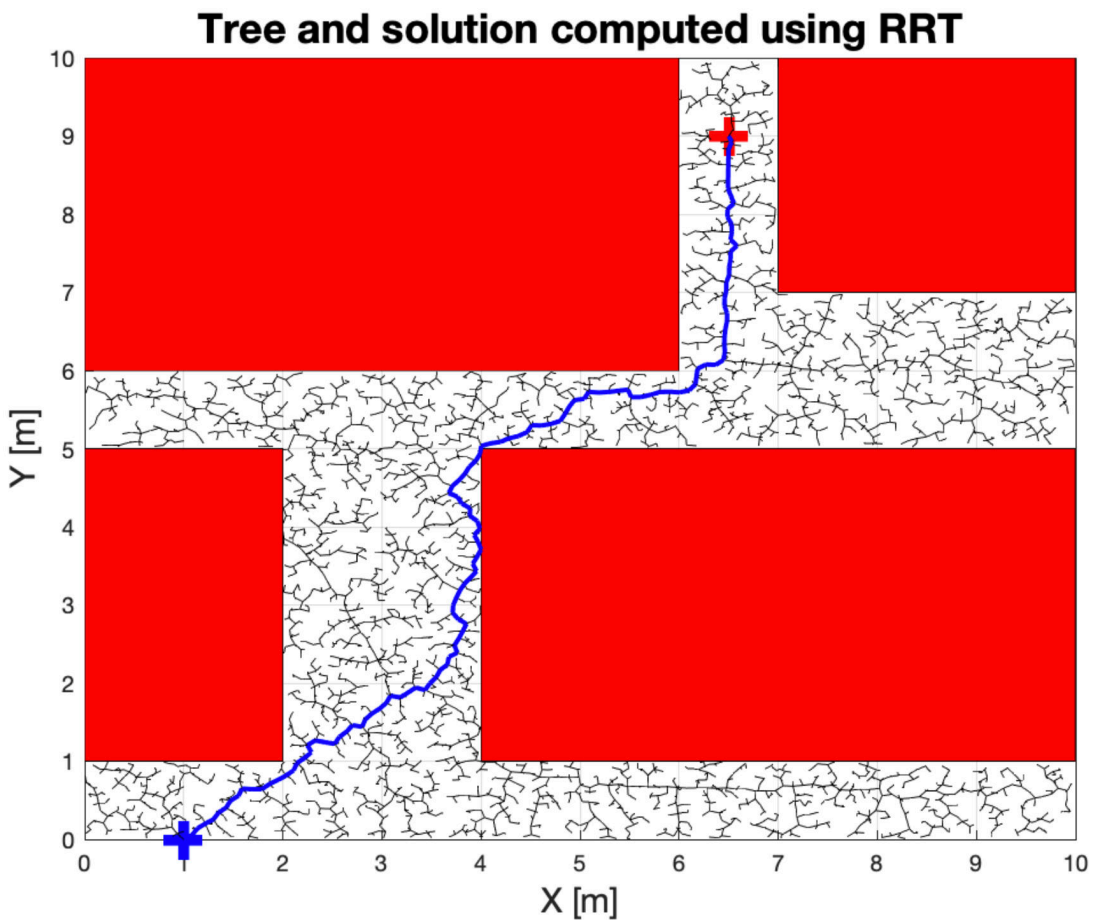
Near

ConnectMinCost

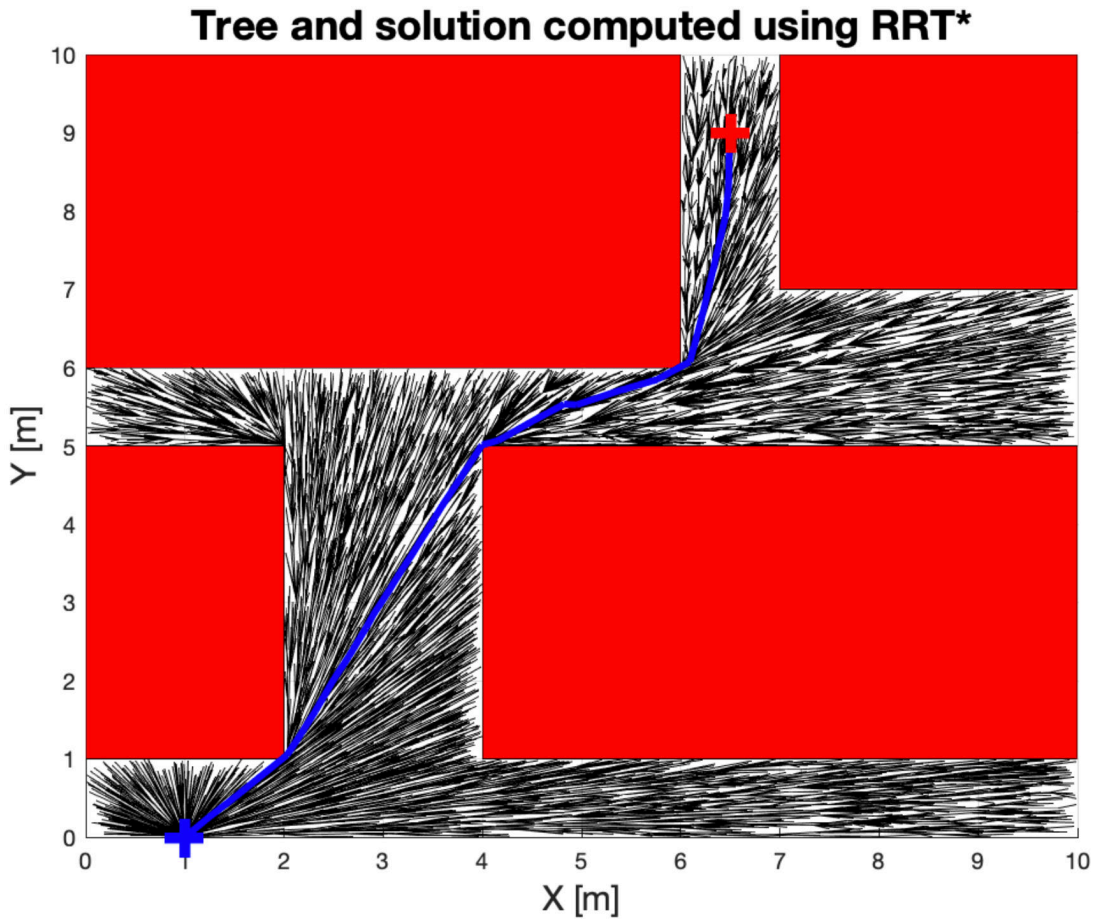
Re-Wire



Comparison of RRT and RRT*



RRT



RRT*

RRT – Implementation Aspects

- Choice of norm for **measure of closeness** between vertices.
- **Bi-directional growth** of two RRTs (or even more) simultaneously.
- **Sampling strategies** dedicated to specific state spaces.
- **Kd-trees** for efficient search in the tree.
- **CL-RRTs** sample in the space of **reference values** for the closed-loop controllers (decreases state dimension and allows unstable dynamics), see Lecture 9.

Planning Using Motion Primitives and State Lattices

General Idea with Lattice Planning (1/2)

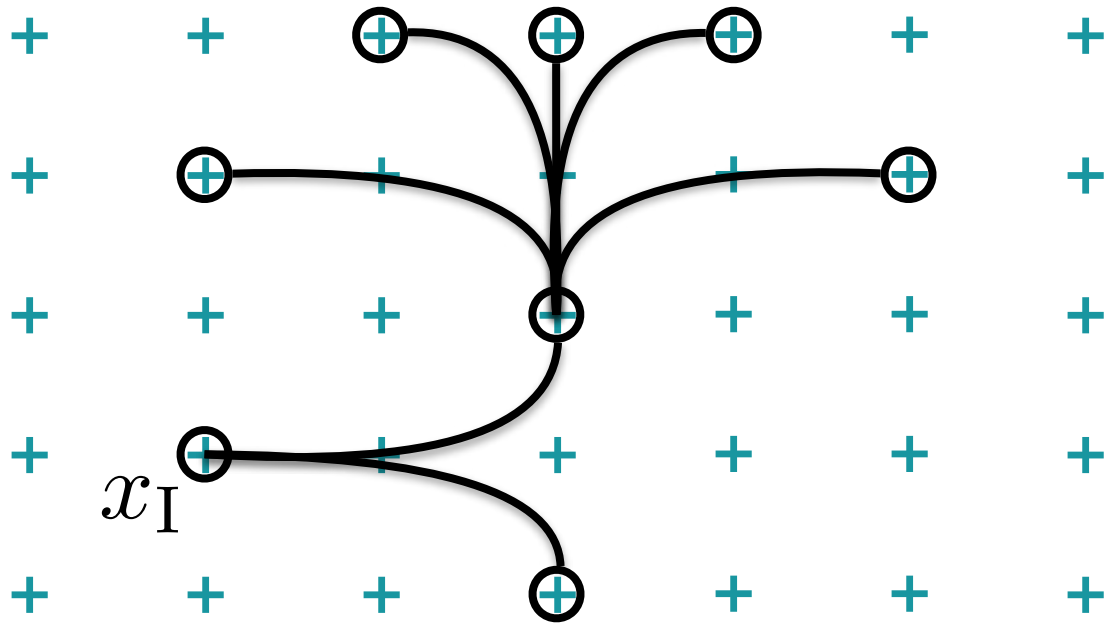
- In contrast to the discrete graph-search problems in Lecture 2, a **motion-planning problem** with a **continuous-time differential motion constraint** is an infinite-dimensional problem.
- No graph with a **finite set of nodes** exists.
- Idea: Select a **finite number of control-signal segments** (***motion primitives***) and choose them such that the start and end states of their corresponding trajectories are on a **discrete lattice** (i.e., a regular grid)

  $\Longrightarrow$ Back on the graph-search problem from Lecture 2.

Likhachev, M., & D. Ferguson: "Planning long dynamically feasible maneuvers for autonomous vehicles". The International Journal of Robotics Research, 28(8), 933-945, 2009.

General Idea with Lattice Planning (2/2)

- The solution is a sequence of motion primitives.



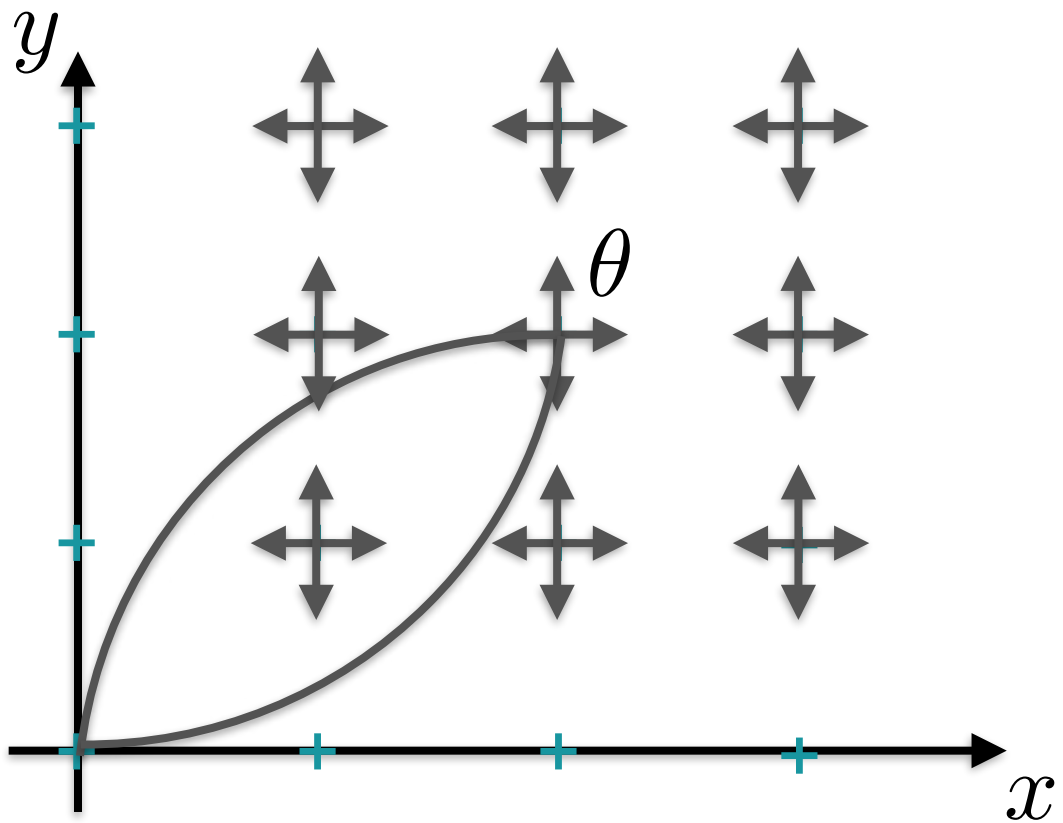
Likhachev, M., & D. Ferguson: "Planning long dynamically feasible maneuvers for autonomous vehicles". The International Journal of Robotics Research, 28(8), 933-945, 2009.

State Lattice – Discretization of State Space

- To transform the original continuous motion-planning problem to a discrete (finite-dimensional) problem, the state space is **discretized** into the set $\mathcal{X}_d$.
- All state variables are discretized, e.g., using a uniform grid, which results in a so called **state lattice**.
- The planning is then relying on using a **sequence of motion segments**, each bringing the vehicle from one state in the discrete set $\mathcal{X}_d$ to another (never outside this set).

State Lattice – Example

- An example where the translational coordinates (x, y) are discretized **uniformly** and four different orientations θ are allowed at each positional state.
- The arrows indicate the set of **allowed orientations**.



Motion Primitives

- The **transition** from one state in $\mathcal{X}_d$ to the next must obey the motion equations of the system (i.e., a differential constraint).
- To encode a (discrete) set of **possible state transitions** within the lattice, a set of **motion primitives** $\mathcal{MP}$ is computed.
- Each motion primitive must start and end at one state in the set $\mathcal{X}_d$ (though not necessary that all states are connected).

Motion Primitives – Examples

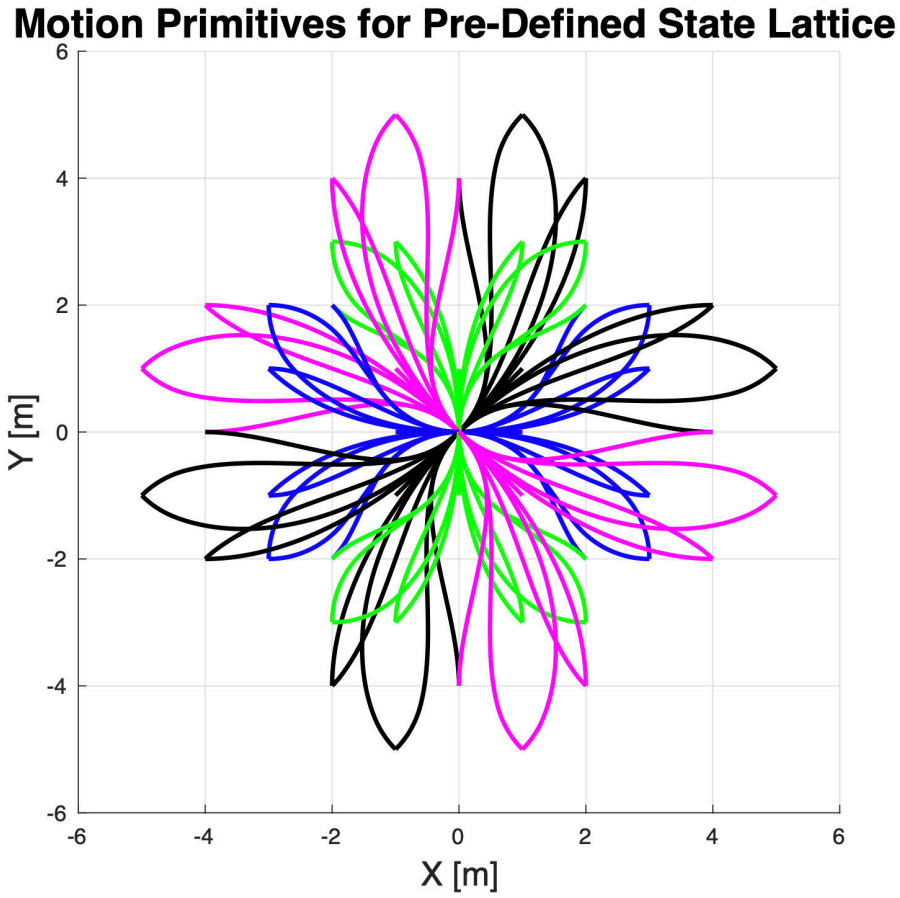
- Consider a **vehicle** with dynamics

$$\dot{x} = v\cos(\theta),$$
$$\dot{y} = v\sin(\theta),$$
$$\dot{\theta} = v/L\tan(u)$$

x, y – translational coordinates
θ – orientation coordinate
v – velocity
u – steering angle (input)
L – wheelbase

- A **sample set of motion primitives** (constant velocity) with the discretized orientation angles

$$\Theta_d = \{0, \pi/4, \pi/2, 3\pi/4, \pi, -3\pi/4, -\pi/2, -\pi/4\}$$



Computation of Motion Primitives

- For certain types of vehicles, **analytical solutions** exist for a dynamically feasible motion between an initial state and another desired state (e.g., Dubins path, see Lecture 3).
- In other cases, **numerical optimal control** can be used to compute (optimal) motion segments to be used as motion primitives (see Lecture 5 and Hand-in Exercise 2).
- Typically a **cost** (e.g., path length, maneuver time, utilization of control inputs) is associated with each motion primitive.
- **Constraints on control inputs** introduced in the computations of the primitives.
- Utilize **symmetries** and translational **invariances** in the computations (translation, rotations, and mirroring).

Bergman, K: "Exploiting Direct Optimal Control for Motion Planning in Unstructured Environments ", Ph.D. Thesis, Div. Automatic Control, Linköping Univ., 2021.
Ljungqvist, O: "Motion planning and feedback control techniques with applications to long tractor-trailer vehicles", Ph.D. Thesis, Div. Automatic Control, Linköping Univ., 2020.

Planning on the Lattice Using Motion Primitives (1/4)

- Recall the **graph-search methods** studied in Lecture 2.
- Using the (finite set of) motion primitives, the **motion-planning problem** can now be solved using graph search.
- In each state $x_d \in \mathcal{X}_d$ in the discretized state space, the **possible transitions** are given by

$$\tilde{u}^p \in \mathcal{MP}(x_d)$$

Planning on the Lattice Using Motion Primitives (2/4)

- After application of a possible motion primitive, a **collision check** gives whether the motion is feasible (given current obstacles), i.e., it must be verified that

$$\mathrm{Path}(x_d, \tilde{u}^p) \in \mathcal{X}_{\mathrm{free}}$$

- Graph-search methods such as **Dijkstra's** and **A*** applicable for the planning.
- The search is **terminated** when x_{G} has been selected for expansion.

Planning on the Lattice Using Motion Primitives (3/4)

- Recall the A* graph-search algorithm from Lecture 2.

Algorithm 4: A* with Motion Primitives

```
1  C(x_I) = 0;
2  Q.insert(x_I, C(x_I) + h(x_I));
3  V(x_I) = x_I;
4  P(x_I) = ∅;
5  while Q ≠ ∅ do
6      x = Q.pop();
7      if x = x_G
8          return SUCCESS;
9      foreach ũ^p ∈ MP(x) do
10         x' = NextState(x, ũ^p);
11         if Path(x, ũ^p) ∉ X_free then
12             continue;
13         if not V(x') || C(x') > C(x) + Cost(x, x') then
14             V(x') = x;
15             P(x') = ũ^p;
16             C(x') = C(x) + Cost(x, x');
17             Q.insert(x', C(x') + h(x'));
18 return FAILURE;
```

Q – priority queue
$C(x)$ – cost-to-come to state x
$h(x)$ – heuristic cost-to-go to goal state for state x
$V(x)$ – parent node to state x
$P(x)$ – motion primitive used to reach state x

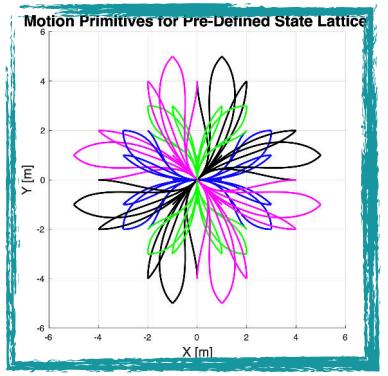
Planning on the Lattice Using Motion Primitives (4/4)

- Same functions as in the **A* graph-search algorithm** from Lecture 2, but with some additional ones:
  - **NextState**: Provides the state reached after application of the considered motion primitive.
  - **Path**: Gives the path between the state x and the next state by application of the specific motion primitive $\tilde{u}_p$.
  - **Cost**: The cost to move between two specified states.

Lattice Planning – Implementation Aspects

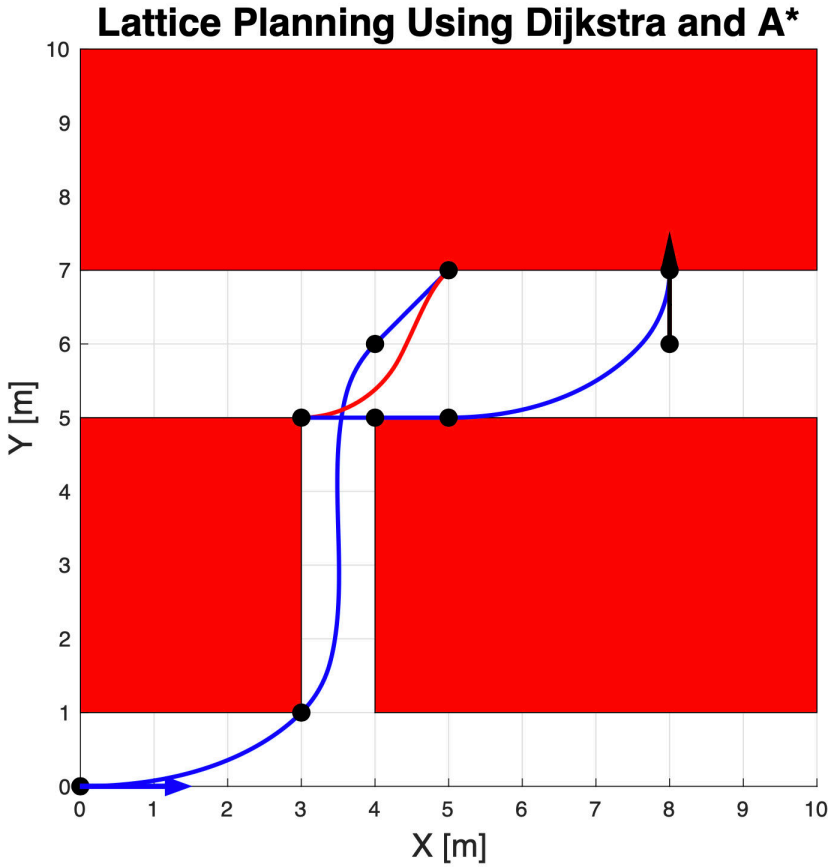
- Choice of **state discretization non-trivial task**, closely connected to the specific vehicle dynamics and targeted planning situations.
- Cardinality of the sets $\mathcal{X}_d$ and $\mathcal{MP}$ influences the **online computational time**, since the graph-search problem increases in size.
- The set of motion primitives can be **computed offline**.
- **Tabulated values for computing heuristics** for distance between two states (e.g., representing the cost-to-go in free space).

Ljungqvist, O: "Motion planning and feedback control techniques with applications to long tractor-trailer vehicles", Ph.D. Thesis, Div. Automatic Control, Linköping Univ., 2020.



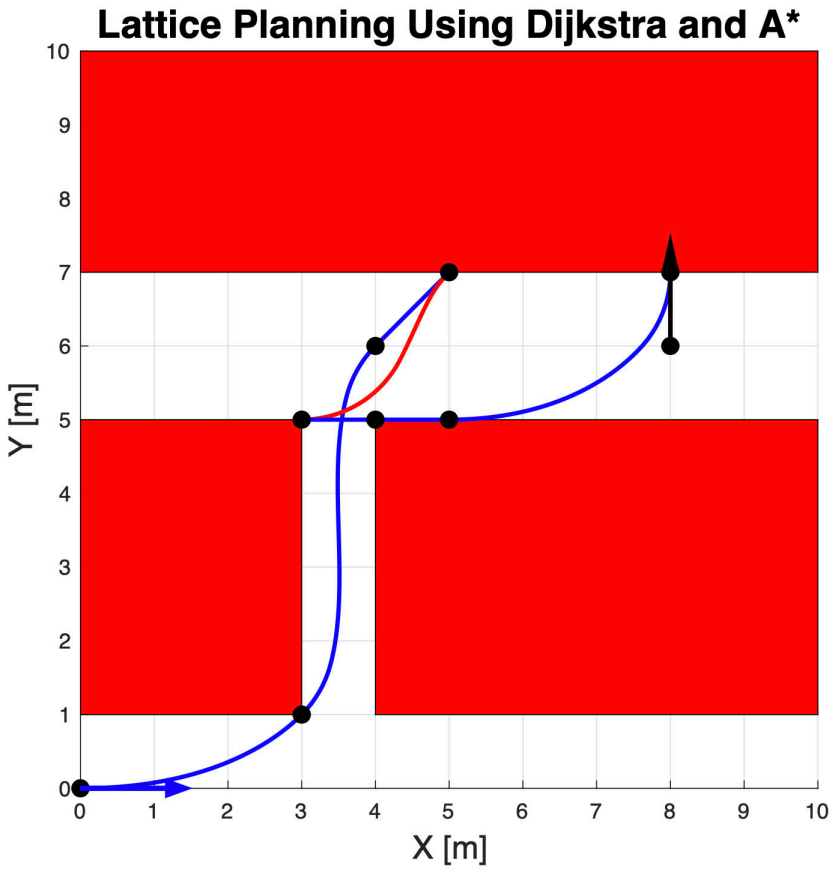
Example: Lattice Planning (1/2)

- The task is to **move from the blue arrow to the black arrow**, using the set of motion primitives shown on a previous slide.
- Application of the Dijkstra and A* algorithms from **Hand-in Exercise 1** gives the results in the plot to the right (blue lines – forward, red lines – reverse).



Example: Lattice Planning (2/2)

- Some observations in the results:
  - The finite set of motion primitives **implies restrictions on the motion**.
  - The **choice of graph-search strategy** leads to different solutions.
- Further studied in Hand-in Exercise 2.



Tools and Libraries for Motion Planning

Open Motion Planning Library

- **Library** containing several different state-of-the-art algorithms for **sampling-based motion planning**.
- A subset of the planners also support **optimal motion planning** and planning with **differential constraints**.
- Possible to interface with **visualization** and **collision-checking** software libraries.
- Homepage: https://ompl.kavrakilab.org

ROS Package MoveIT

- A ROS (Robot Operating System) package for **motion planning**.
- Includes, e.g., **interfaces to planners** from:
  - Open Motion Planning Library,
  - Search Based Planning Library
- **Visualization** is possible using the tool RViz.
- C++ and Python interfaces are available.
- Homepage: https://moveit.ros.org

Decoupled Approaches to Motion Planning

The material in the following section is slightly more advanced and can be considered as extra material in the course. The methods presented here will not be part of the hand-in exercises.

Decoupled Approach to Motion Planning

- The motion-planning problem can be **decoupled** into the following two phases to **reduce complexity**:
  - **path planning** and
  - **trajectory planning** (~velocity planning).
- The second problem is then often called **path-constrained trajectory planning**.

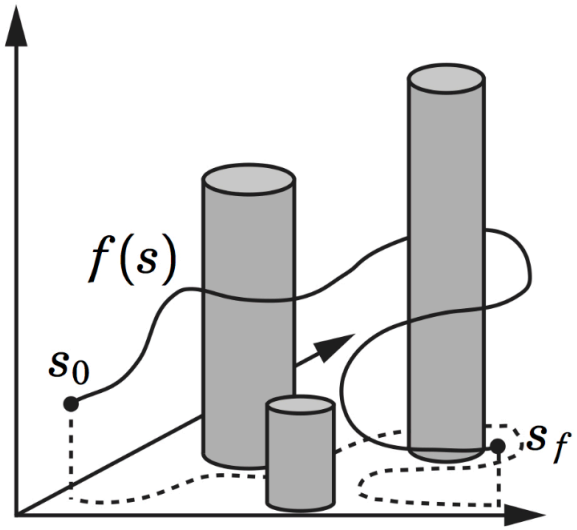
LaValle, S. M: Planning Algorithms. Cambridge University Press, 2006.

Parameterization of Vehicle Dynamics on Path (1/2)

- Assume that a **path** is described by a (two-times differentiable) function of a (scalar) **path coordinate**

$$q = f(s), \qquad s \in [s_0, s_f], \qquad q \in \mathbb{R}^n$$

- The idea is now to describe the vehicle dynamics using this additional **path-tracking requirement**.



Dahl, O: "Path Constrained Robot Control", Ph.D. Thesis, Dept. Automatic Control, Lund Univ., 1992.

Parameterization of Vehicle Dynamics on Path

- Using the **chain rule**, the following relations can be shown

$$\dot{q} = \frac{\mathrm{d}f}{\mathrm{d}t} = \frac{\mathrm{d}f}{\mathrm{d}s}\frac{\mathrm{d}s}{\mathrm{d}t} = f'(s)\dot{s},$$

$$\begin{aligned}\ddot{q} = \frac{\mathrm{d}^2 f}{\mathrm{d}t^2} &= \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\mathrm{d}f}{\mathrm{d}s}\frac{\mathrm{d}s}{\mathrm{d}t}\right) \\ &= \frac{\mathrm{d}^2 f}{\mathrm{d}s^2}\frac{\mathrm{d}s}{\mathrm{d}t}\frac{\mathrm{d}s}{\mathrm{d}t} + \frac{\mathrm{d}f}{\mathrm{d}s}\frac{\mathrm{d}^2 s}{\mathrm{d}t^2} \\ &= f''(s)\dot{s}^2 + f'(s)\ddot{s}\end{aligned}$$

$$(\,)' \sim \frac{\mathrm{d}}{\mathrm{d}s}$$

$$\dot{(\,)} \sim \frac{\mathrm{d}}{\mathrm{d}t}$$

Dahl, O: "Path Constrained Robot Control", Ph.D. Thesis, Dept. Automatic Control, Lund Univ., 1992.

Example: Particle Motion in Two Dimensions (1/2)

- Assume that we have a **particle moving in a plane** like a 2D printer, where the **force** in each direction can be controlled

$$m_1\ddot{q}_1 = u_1, \quad m_2\ddot{q}_2 = u_2, \qquad -1 \leq u_1, u_2 \leq 1$$

- Using the relations on the previous slide, it is obtained

$$m_1(f_1'(s)\ddot{s} + f_1''(s)\dot{s}^2) = u_1,$$
$$m_2(f_2'(s)\ddot{s} + f_2''(s)\dot{s}^2) = u_2$$

Example: Particle Motion in Two Dimensions (2/2)

- The path to be followed is a **straight line** described by

$$f_1 = k_1 s, \qquad f_2 = k_2 s$$

- With this path and the considered vehicle dynamics, the problem is to **determine the path acceleration** $\ddot{s}$ (and thereby the control inputs u_1, u_2) such that

$$\frac{\mathrm{d}s}{\mathrm{d}t} = \dot{s}, \qquad -1 \leq m_1 k_1 \ddot{s} \leq 1, \qquad s(0) = s_0, \quad s(t_f) = s_f,$$
$$\frac{\mathrm{d}^2 s}{\mathrm{d}t^2} = \ddot{s} \qquad -1 \leq m_2 k_2 \ddot{s} \leq 1 \qquad \dot{s}(0) = 0, \quad \dot{s}(t_f) = 0$$

Transformation to Fixed Interval (1/3)

- The **time duration** of the motion is **unknown *a priori***, and thus the final time is part of the problem to compute.
- **Transformation** to a **fixed interval** (over values of the path coordinate), allows reduction of the **problem complexity**.

Dahl, O: "Path Constrained Robot Control", Ph.D. Thesis, Dept. Automatic Control, Lund Univ., 1992.

Transformation to Fixed Interval (2/3)

- For a general **second-order system**, the system dynamics can be written as follows

$$F(s, \dot{s}, \ddot{s}, u) = 0$$

- Introduce **new variables** according to

$$\alpha(s(t)) = \ddot{s}(t), \qquad \beta(s(t)) = \dot{s}(t)^2$$

Dahl, O: "Path Constrained Robot Control", Ph.D. Thesis, Dept. Automatic Control, Lund Univ., 1992.

Transformation to Fixed Interval (2/3)

- Using the **chain rule again**, it is obtained that

$$\ddot{s}(t) = \frac{\mathrm{d}}{\mathrm{d}t}\sqrt{\beta(s(t))} = \frac{1}{2\sqrt{\beta(s(t))}}\beta'(s(t))\dot{s} = \frac{1}{2}\beta'(s(t))$$

$$\Rightarrow \quad \beta'(s) = 2\alpha(s)$$

- Assume now that the **cost function** to **minimize** in this motion-planning problem is the **time**. This cost can be written in the **path velocity** as follows

$$t_f - t_0 = \int_{t_0}^{t_f} \mathrm{d}t = \int_{s_0}^{s_f} \frac{1}{\dot{s}}\mathrm{d}s = \int_{s_0}^{s_f} \frac{1}{\sqrt{\beta}}\mathrm{d}s$$

Dahl, O: "Path Constrained Robot Control", Ph.D. Thesis, Dept. Automatic Control, Lund Univ., 1992.

Optimization Problem for Path Tracking (1/2)

- With the transformations on the previous slides, the **time-optimal trajectory-planning problem for path following** can be stated as

$$\begin{aligned} \underset{\alpha(s),\ s\in[s_0,s_f]}{\text{minimize}} \quad & \int_{s_0}^{s_f} \frac{1}{\sqrt{\beta}}\mathrm{d}s \\ \text{subject to} \quad & \beta' = 2\alpha, \quad F(s,\beta,\alpha,u) = 0, \\ & u_{\min} \leq u \leq u_{\max} \end{aligned}$$

- The **solution** tells us **how fast we can go along the desired path**, while taking the motion constraints and control-input constraints into account.

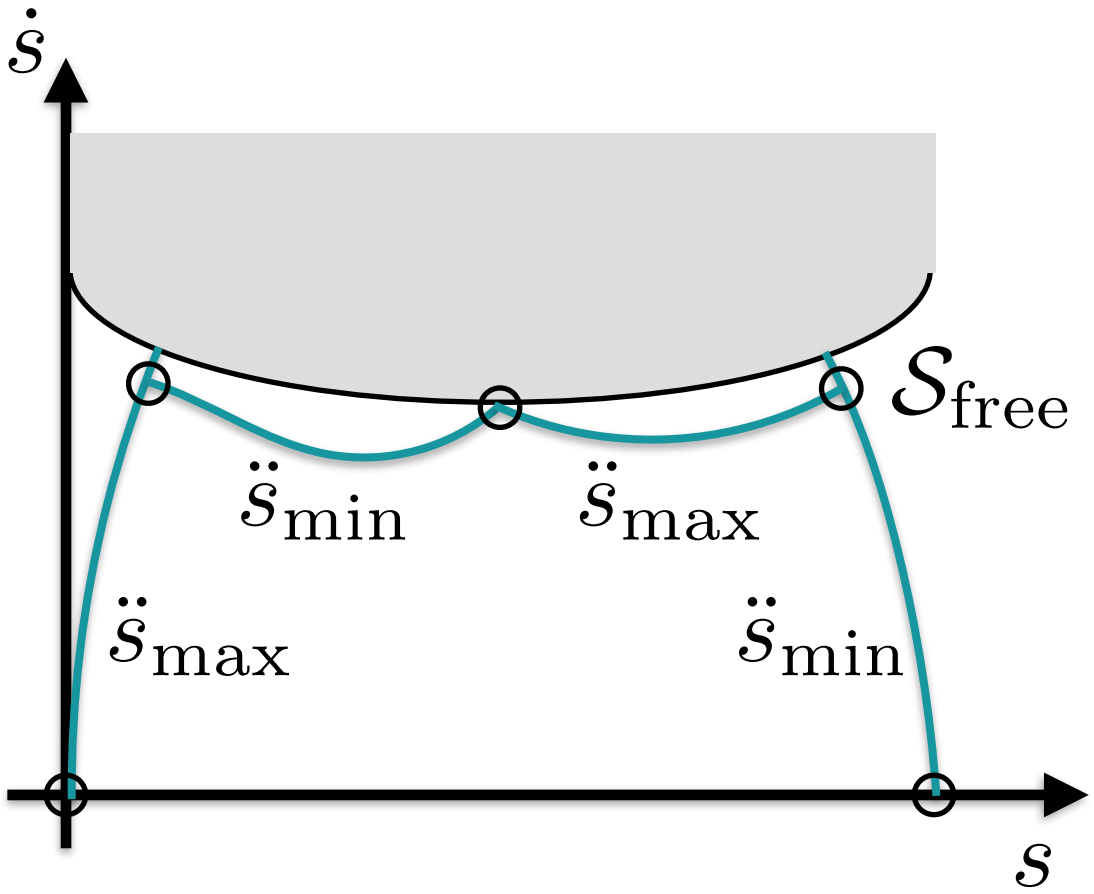
Optimization Problem for Path Tracking (2/2)

- The **optimization problem** can be solved using **numerical optimal control** (see Lecture 5) or by constructing the solution in the **phase plane** by integration (next slide).
- Note that the problem has been transformed such that
  - it is now over a **fixed interval** (time unknown *a priori*) and
  - the **state dimension is independent** of the number of degrees-of-freedom n in the considered vehicle dynamics.

Planning in the Phase Plane s—$\dot{s}$

- To compute the time-optimal solution, the path velocity $\dot{s}$ should be **as high as possible** along the path.
- The solution can be constructed by **integrating forward and backward** with maximum or minimum path acceleration $\ddot{s}$.
- The **allowed area in the phase plane** is described by

$$\mathcal{S}_{\text{free}} = \{(s, \dot{s}) : \ddot{s}_{\min}(s, \dot{s}) \leq \ddot{s}_{\max}(s, \dot{s})\}$$



Bobrow, J. E., S. Dubowsky, & J. S. Gibson: "Time-optimal control of robotic manipulators along specified paths", The International Journal of Robotics Research, 4(3), 3-17, 1985.

Example: Time-Optimal Motion for the Particle Model (1/4)

- Recall the previous example where the **particle is moving along a straight line** with constant limits on the forces. Let the parameters be

$$m_1 = 2, \quad m_2 = 1, \quad k_1 = 1, \quad k_2 = 1, \quad s_0 = 0, \quad s_f = 1$$

- Given the constraints on the inputs, the **time-optimal solution** should satisfy

$$-1 \leq u_1, u_2 \leq 1 \quad \Longrightarrow \quad \begin{aligned} -1 \leq m_1 k_1 \ddot{s} \leq 1, \\ -1 \leq m_2 k_2 \ddot{s} \leq 1 \end{aligned}$$

$$\ddot{s}_{\max} = \min_{i=1,2} \frac{u_i^{\max}}{m_i k_i} = \frac{1}{2},$$

$$\ddot{s}_{\min} = \max_{i=1,2} \frac{u_i^{\min}}{m_i k_i} = -\frac{1}{2}$$

Example: Time-Optimal Motion for the Particle Model (2/4)

- The time-optimal solution is of **bang-bang character**, i.e.,

$$\ddot{s} = \begin{cases} \ddot{s}_{\max}, & t_0 \leq t \leq t_1 \\ \ddot{s}_{\min}, & t_1 \leq t \leq t_f \end{cases}$$

- **Integration two times** and utilization of boundary values and continuity of path coordinate and path velocity gives

$$s = \begin{cases} \frac{1}{4}t^2, & t_0 \leq t \leq t_1 \\ -\frac{1}{4}t^2 + \sqrt{2}t - 1, & t_1 \leq t \leq t_f \end{cases} \qquad t_1 = \sqrt{2}, \qquad t_f = 2t_1$$

Example: Time-Optimal Motion for the Particle Model (3/4)

- Recall the **differential relations** with respect to the path coordinate

$$\beta'(s) = 2\alpha(s), \quad \alpha(s(t)) = \ddot{s}(t), \ \beta(s(t)) = \dot{s}(t)^2$$

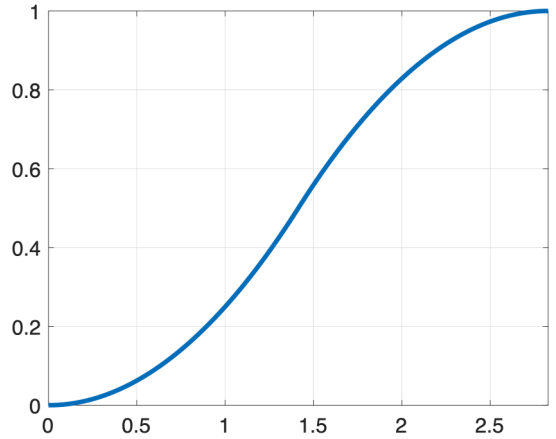
- **Path acceleration** and **path velocity** then computed as

$$\alpha = \begin{cases} \ddot{s}_{\max}, & s_0 \le s \le s_1 \\ \ddot{s}_{\min}, & s_1 \le s \le s_f \end{cases} \quad \boxed{\beta'(s) = 2\alpha(s)} \Longrightarrow \quad \sqrt{\beta} = \begin{cases} \sqrt{s}, & s_0 \le s \le s_1 \\ \sqrt{(-s+1)}, & s_1 \le s \le s_f \end{cases}$$

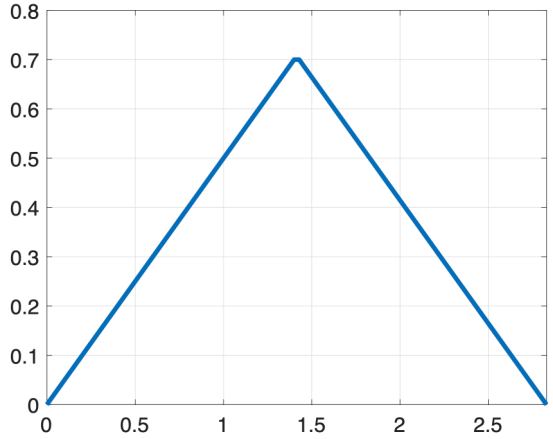
$$s_1 = \frac{1}{2} s_f, \quad s_f = 1$$

Example: Time-Optimal Motion for the Particle Model (4/4)

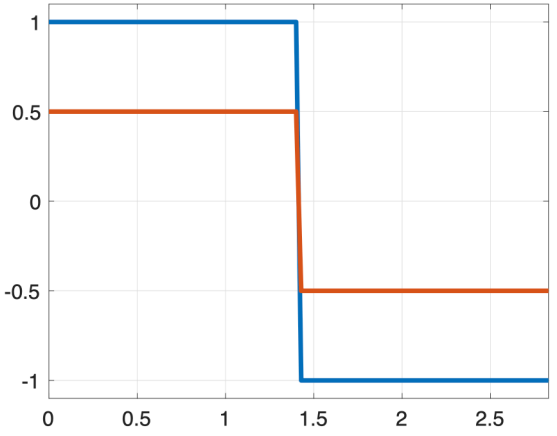
Path coordinate $s(t)$



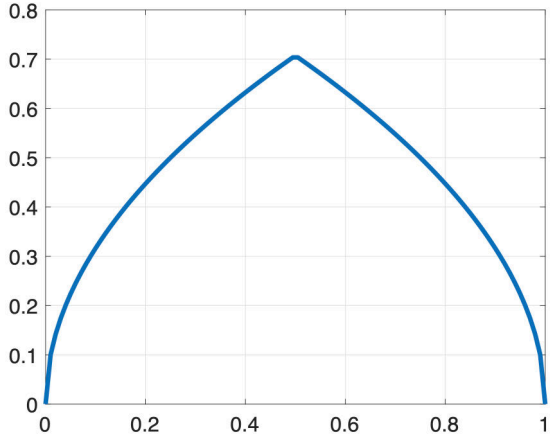
Path velocity $\dot{s}(t)$



$u_1(t)$ (blue), $u_2(t)$ (red)



Path velocity $\sqrt{\beta}(s)$



References and Further Reading

References and Further Reading (1/2)

All the following books and articles are not part of the reading assignments for the course, but cover the topics studied during this lecture in more detail.

- Bergman, K: ”Exploiting Direct Optimal Control for Motion Planning in Unstructured Environments ”, Ph.D. Thesis No. 2133, Div. Automatic Control, Linköping Univ., 2021.
- Bobrow, J. E., S. Dubowsky, & J. S. Gibson: ”Time-optimal control of robotic manipulators along specified paths". *The International Journal of Robotics Research*, 4(3), 3-17, 1985.
- Dahl, O: ”Path Constrained Robot Control”, Ph.D. Thesis, Dept. Automatic Control, Lund Univ., 1992.
- Dubins, L. E., ”On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents”, American Journal of Mathematics, 79(3), 497-516, 1957.

References and Further Reading (2/2)

- Karaman, S., & E. Frazzoli: ”Sampling-based algorithms for optimal motion planning". *The International Journal of Robotics Research*, 30(7), 846-894, 2011.
- LaValle, S. M., & J. J. Kuffner Jr: "Randomized kinodynamic planning”. *The International Journal of Robotics Research,* 20(5), 378-400, 2001.
- LaValle, S. M: *Planning Algorithms*. Cambridge University Press, 2006.
- Likhachev, M., & D. Ferguson: ”Planning long dynamically feasible maneuvers for autonomous vehicles". *The International Journal of Robotics Research*, 28(8), 933-945, 2009.
- Ljungqvist, O: ”Motion planning and feedback control techniques with applications to long tractor-trailer vehicles”, Ph.D. Thesis, Div. Automatic Control, Linköping Univ., 2020.

www.liu.se

LINKÖPING UNIVERSITY