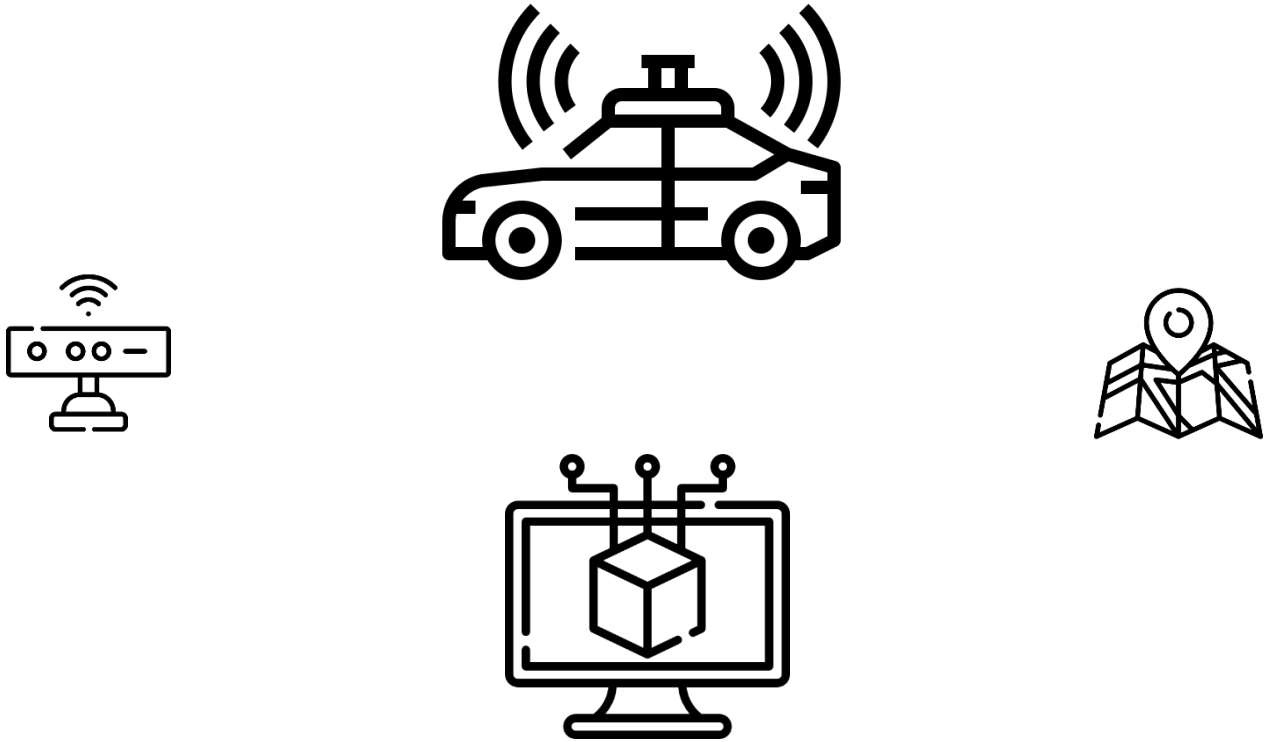
Institute of Automotive Technology
Faculty for Mechanical Engineering
Technical University of Munich

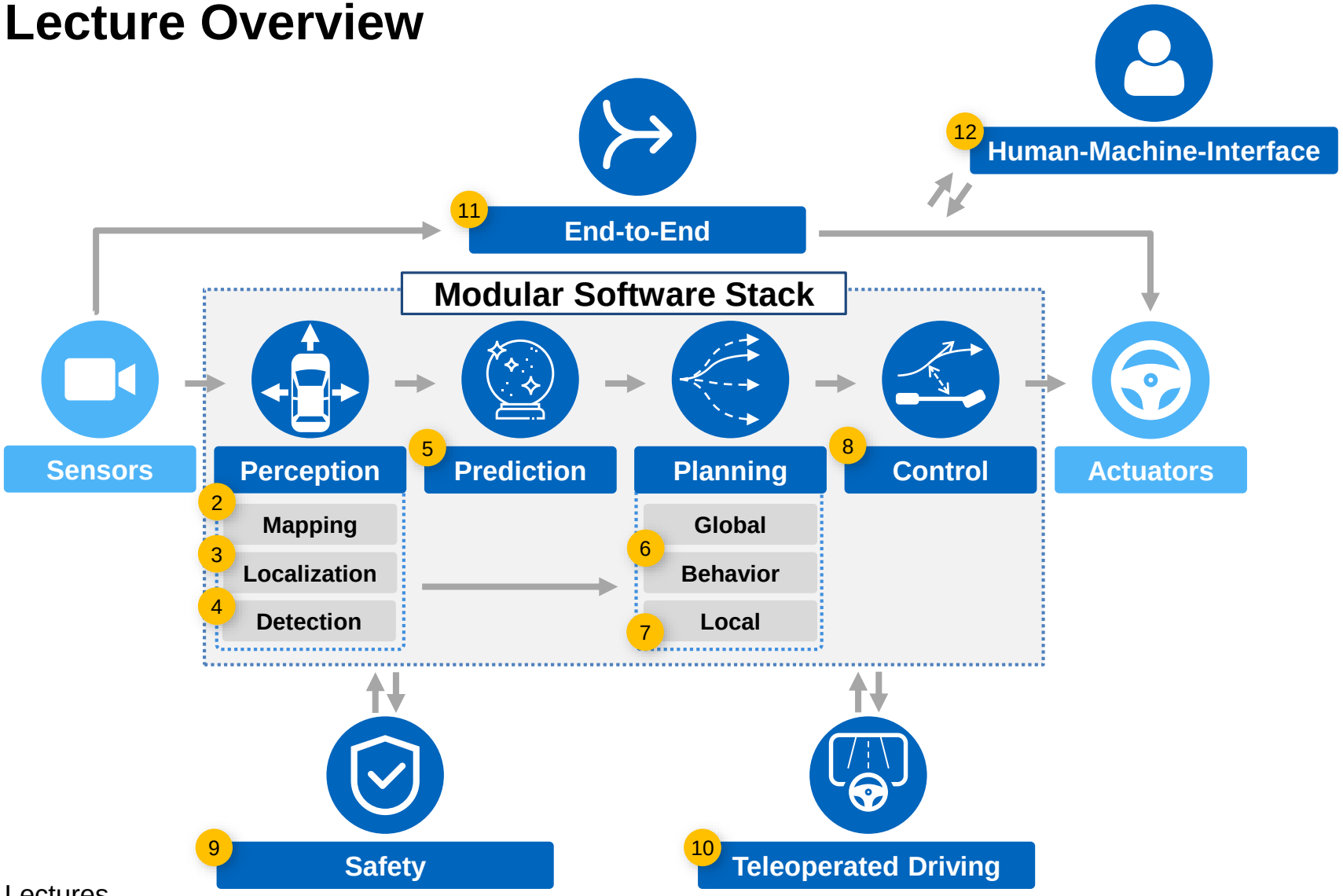
Autonomous Driving Software Engineering

Prof. Dr.-Ing. Markus Lienkamp

Phillip Karle, M. Sc.



Lecture Overview



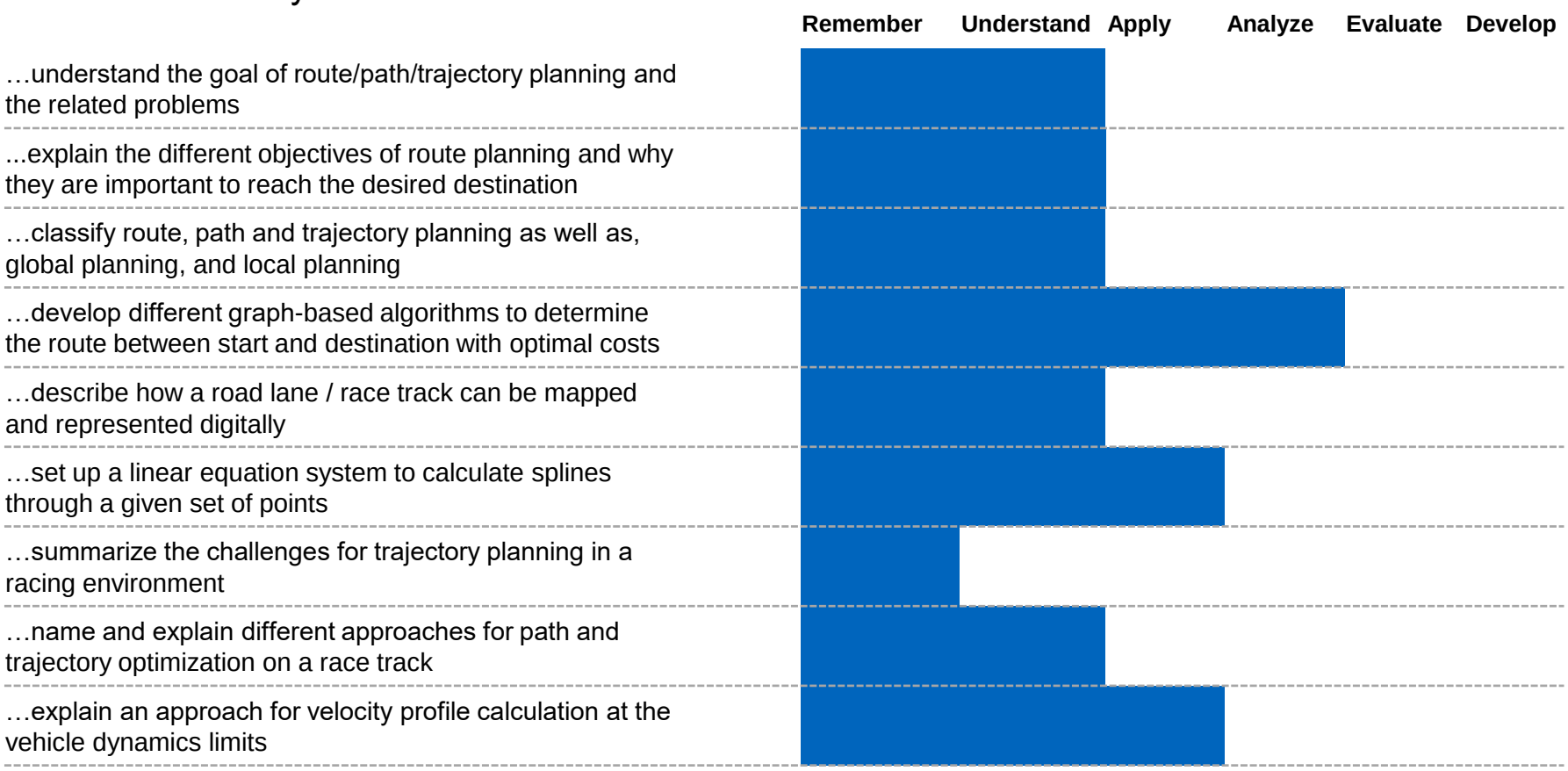
X = Lectures

Source: https://www.youtube.com/watch?v=7NGEM5jcXV4

Objectives for Lecture 6: Global Planning

Depth of understanding

After the lecture you are able to…



Global Planning
Prof. Dr. Markus Lienkamp

Rainer Trauth, M. Sc.

Agenda

1. **Introduction**
2. Global Planning:
    1. Dijkstra-Algorithm
    2. A*-Algorithm
3. Behavior Planning
4. Global Trajectory Planning on the Racetrack
5. Summary

Introduction – Motivation

Map representation

https://mapstyle.withgoogle.com/

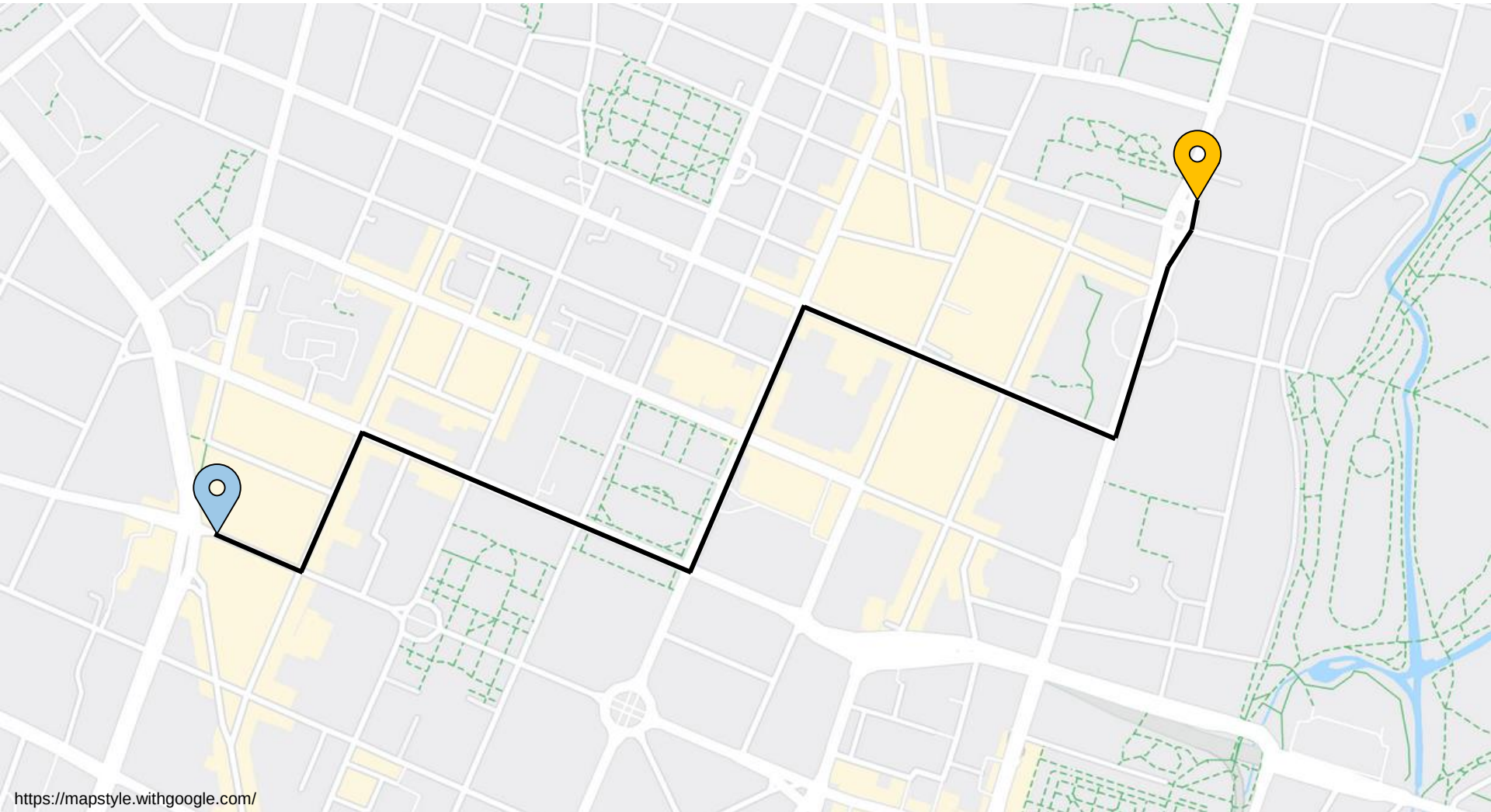
Introduction – Motivation

Map representation

https://mapstyle.withgoogle.com/

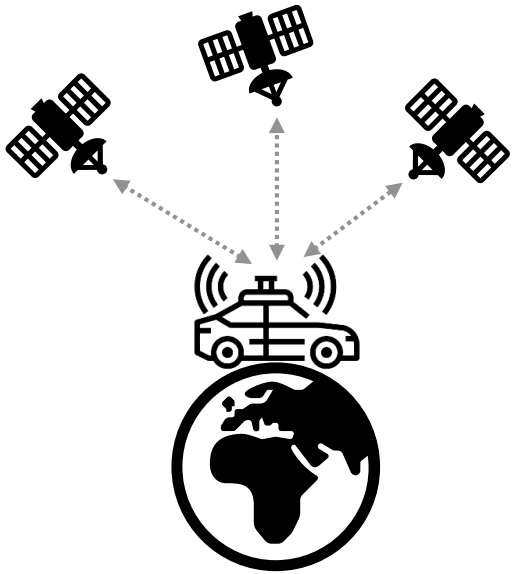
Introduction – Motivation

Map representation

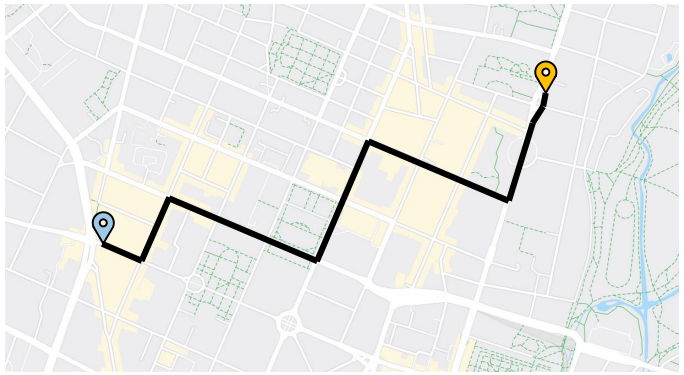


https://mapstyle.withgoogle.com/

Introduction – Motivation
Preconditions



Global Positioning System
Inertial Measurement Unit
Wheel Encoders



Detailed map information

https://mapstyle.withgoogle.com/

Additional Slides

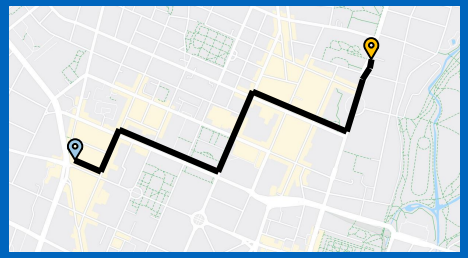
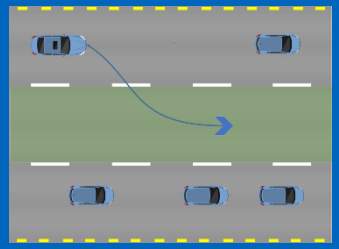
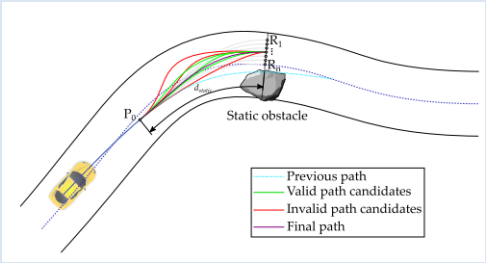
The human approach to finding routes on a street map is a very visual one. Feasible routes/paths are developed by visually tracing different street lines in the general direction of the target. Obstacles, dead ends and restrictions are intuitively taken into account, although – in typical routing situations – no intensive thinking or reflection is carried out. The human act of finding a route/path between a starting point and a destination can be considered as „intelligent“.

As a matter of fact, we do not understand the human approach of finding routes on maps in detail. Hence, we cannot model it directly. Instead, search algorithms have been evolved that solve the same problems by supposedly different means. Some popular ones of these shall be introduced in this lecture.

The question that remains is whether or not the presented algorithms can themselves be considered as „intelligent“ solely because they solve a problem that humans solve by means of human intelligence.

Introduction – Top-Down Consideration

Definition of terms

Global Planning	Uses map for planning without information of local environment. Focus: hours to minutes.	
Behavior Planning	High-level description of the vehicle motion. Focus: minutes to seconds.	
Local Planning	Consideration of local objects. Deals with reactive decisions. Focus: seconds to milliseconds.	

This Lecture

Next Lecture

https://mapstyle.withgoogle.com/
https://medium.com/behavior-trees-for-path-planning-autonomous/behavior-trees-for-path-planning-autonomous-driving-8db1575fec2c

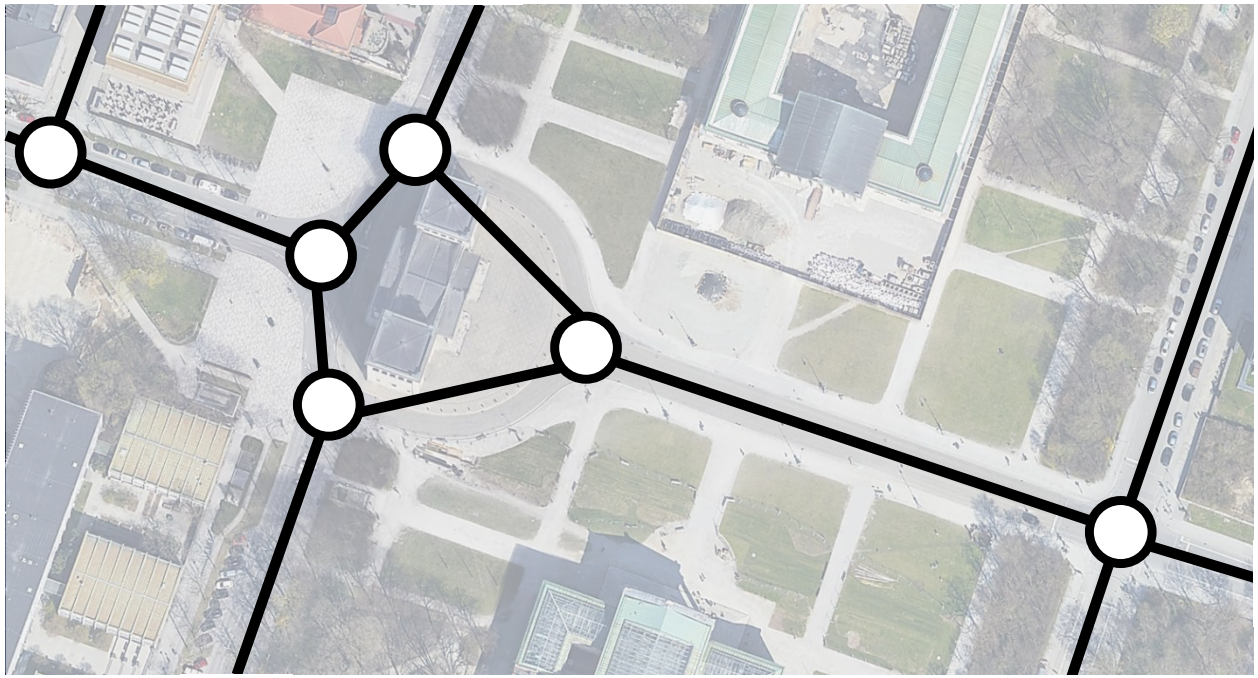
Introduction – Global to Local Planning
Differentiation

Global Planning	Local Planning
Map-based	Sensor-based (reactive planning)
Relatively slower response	Fast response
Known workspace	Suppose incomplete workspace
Generate path/route before moving	Planning and moving at the same time
No strict requirements on calculation time	Requirements on calculation time

Introduction – Route / Path / Trajectory Planning
Differentiation

<u>Route Planning</u>:

- Decision to take a route from source to destination
- Sequence of <u>discrete geometrical</u> nodes in map network

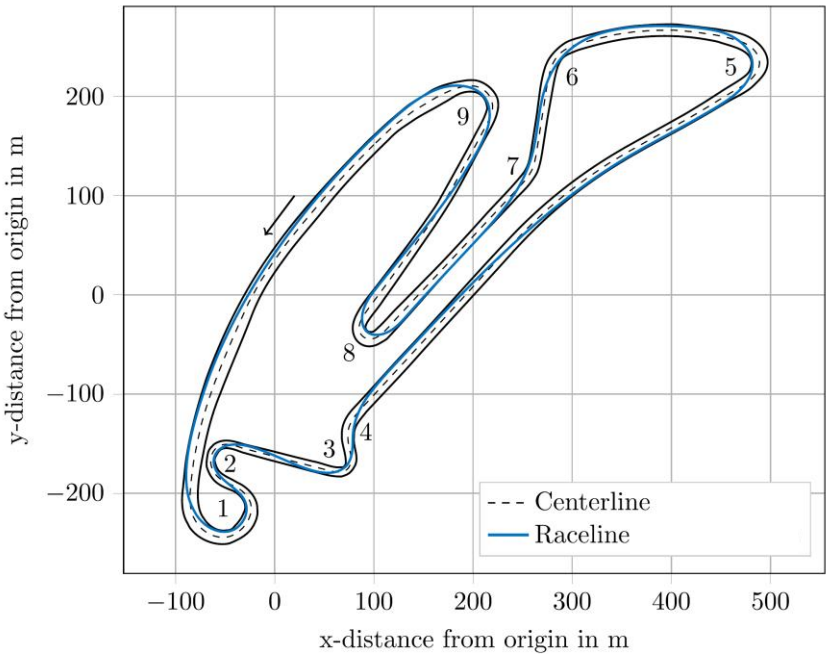


Source: Google Earth

Introduction – Route / Path / Trajectory Planning
Differentiation

Path Planning:

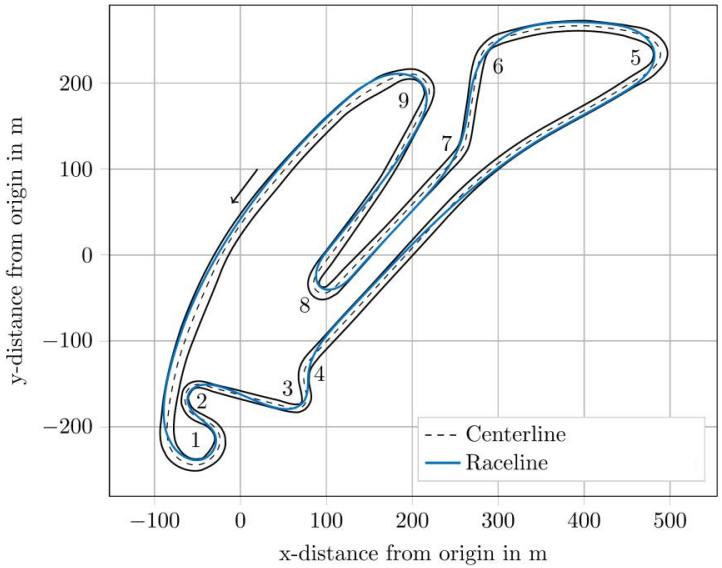
- Continuous curve in spatial domain
- Consideration of spatial / geometrical boundary conditions possible (e.g., track width, maximum curvature)



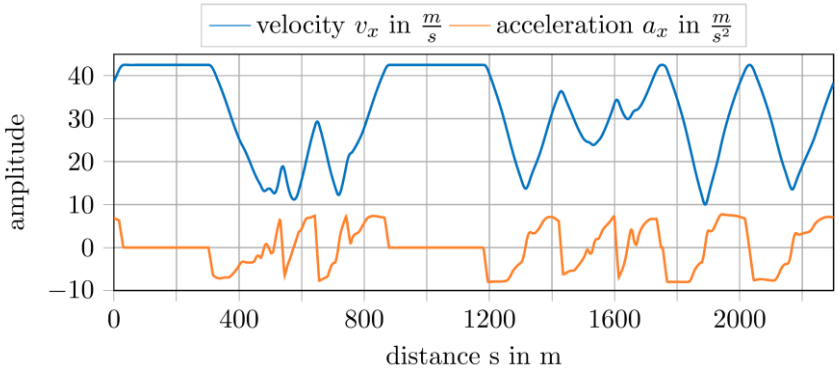
Introduction – Route / Path / Trajectory Planning
Differentiation

Trajectory Planning:

- Continuous curve in spatio-temporal domain
- Further conditions can be checked by temporal information (e.g., freedom from collisions, compliance with acceleration limits)



+



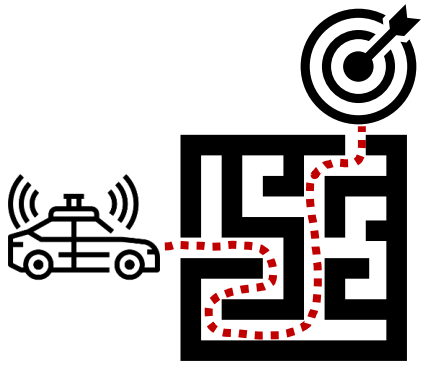
Additional Slides

A distinction must be made between global route planning and global path planning. Global route planning is generally used to generate a route from a source node to a destination node. Certain target variables, such as distance or time, can be minimized in this context. Exact geometric curve lines of the vehicle are not taken into account. Global path planning considers the continuous planning of the vehicle's curve. Spatial and geometrical boundary conditions must be observed. In global trajectory planning, the temporal component is also considered. For example, the vehicle should observe the maximum curvature velocity. The temporal component is required for the calculation of such target variables. Because the temporal component is particularly important in local planning, the term local trajectory planner is typically used. The local planner must be able to react to many objects and changes in the environment in the shortest possible time with the help of sensor data. The local planner uses sensor data to respond quickly to changes in the environment.

Introduction – Motivation
Objective

Find a route from start to destination

The global planner must be able to find a route from A to B.



Route has to be feasible

The calculated route must be feasible and comply with road traffic rules.

https://rp-online.de/leben/auto/fahrberichte/mercedes-g-klasse-im-gelaende-weiterhin-koenig_bid-22561309#1

Introduction – Motivation
Objective

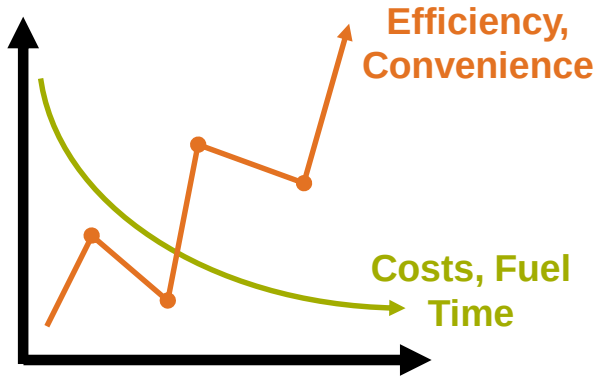
Route has to be effective

The calculated route must lead to the goal that is supposed to be reached at the beginning of the route planning process.

Route has to be efficient

In general, costs are incurred to achieve a desired destination. Possible target variables (costs) to be minimized (maximized) are:

time, fuel consumption, distance, costs, toll charges, convenience, travel experience…



https://onlinestreet.de/strassen/ortsschild/

Global Planning
Prof. Dr. Markus Lienkamp

Rainer Trauth, M. Sc.

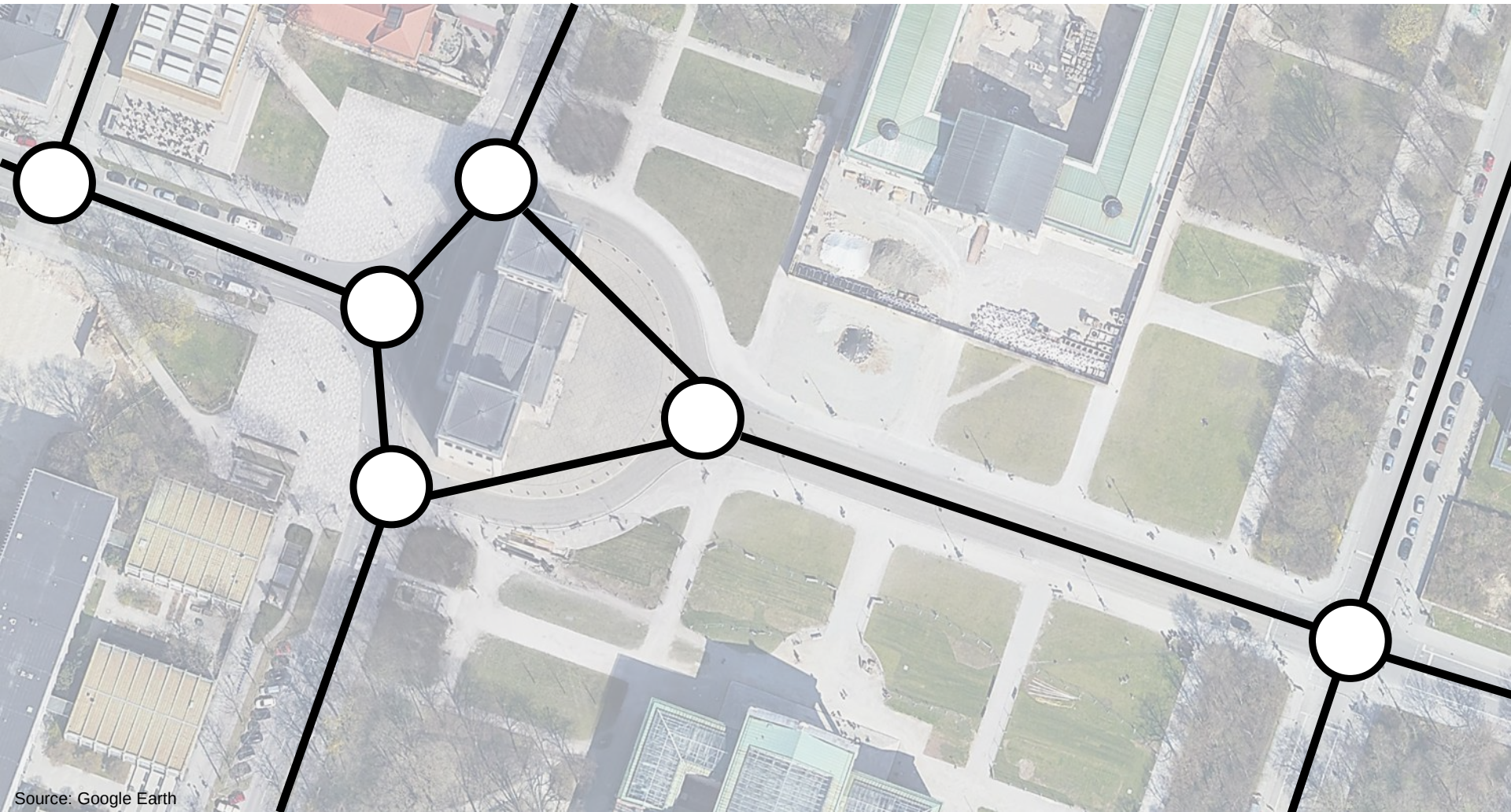
Agenda

1. Introduction
2. **Global Planning:**
    1. Dijkstra-Algorithm
    2. A*-Algorithm
3. Behavior Planning
4. Global Trajectory Planning on the Racetrack
5. Summary

Global Planning
Basic Elements

Source: Google Earth

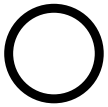
Global Planning
Basic Elements



Source: Google Earth

Global Planning
Basic Elements

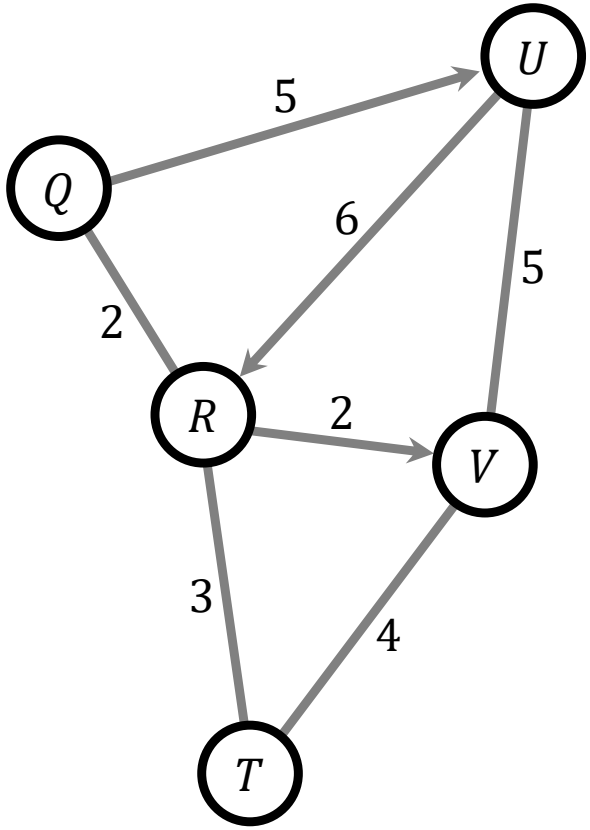
Node



(Weighted)
Directed Edge

(Weighted)
Undirected Edge

Edge weights are interpretable as **cost** (e.g. time, distance …)



Additional Slides

The data models for digital maps are structurally related to mathematical graphs. Mathematical graphs in turn can easily be formalized and used for automatic processing by pathfinding algorithms. Due to this reason, they constitute a link between mapping and pathfinding. If abstract mathematical graphs are used, the term “path planning” is used. Navigation engines are built upon mathematical graphs that are calculated from the digital map. The graph ultimately used for routing is restricted to the information indispensable for the pathfinding tasks.

In graphs, ways (streets) are cut into segments called edges. Therefore, the concept of a street is suspended, because streets – in the sense of collections of segments – are not directly relevant to the pathfinding tasks. Consequently, streets segments without crossroads or speed limit changes are oftentimes represented by a single edge in the routing graph. The reason for this is that the driver can only use or not use the street segments corresponding to the edge, but once he enters any of these segments he cannot do anything besides following the street until the next crossroads enables him to change direction. From a routing point of view, it is therefore redundant to sustain all street segments in the digital map when generating a routable graph. Hence, nodes in a routing graph correspond to crossroad positions or changes in road attributes that are relevant to the pathfinding process.

In graphs, there are two types of edges: directed and undirected ones. Directed edges can only be traversed in one direction, hence modeling one-way streets or individual lanes, whereas undirected edges work both ways.

Oftentimes, a graph is not used exclusively to find an arbitrary path through the street network, but to find the most cost efficient way to do it. Any pathfinding algorithm optimizing costs needs cost information stored in the graph. Storing cost information in a mathematical graph is possible using so-called edge weights. An edge weight is associated with an edge and denotes the cost of traversing that edge a single time.

In the context of navigation systems, costs can for example be units of time, street length or air pollution (CO_2 emission).

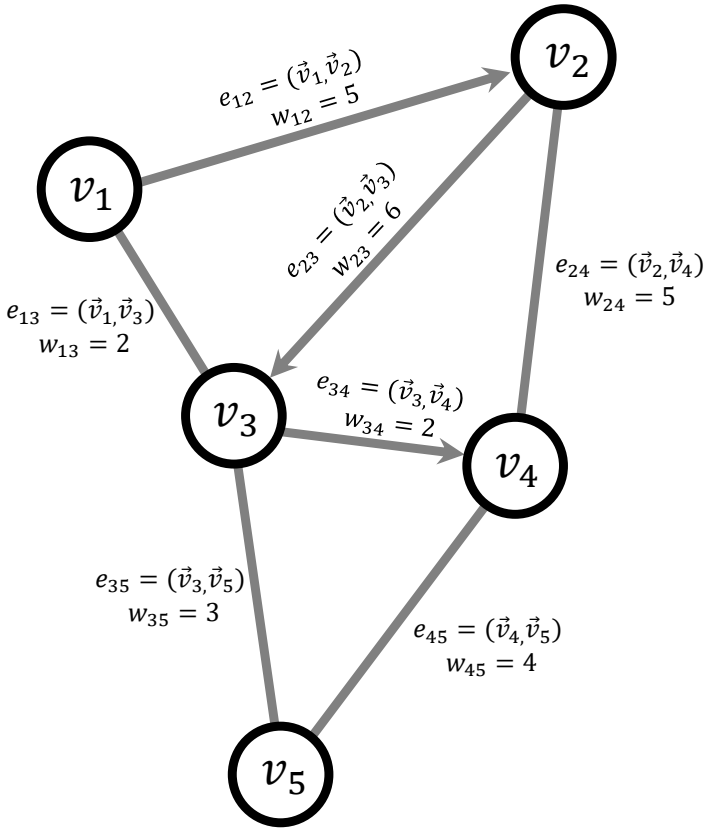
Global Planning
Basic Elements

$G:$	$Graph$
$V:$	$Nodes$
$\vec{v}:$	$Node$
$E:$	$Edges$
$e:$	$Edge$
$W:$	$Weights$
$w:$	$Weight$

$$G = (V, E, W)$$

$$e = (\vec{u}, \vec{v});\ \vec{u}, \vec{v} \in V$$

$$E = \{e_{12}, e_{13}, e_{23}, \dots\}$$
$$V = \{\vec{v}_1, \vec{v}_2, \vec{v}_3, \dots\}$$
$$W = \{w_{12}, w_{13}, w_{23}, \dots\}$$



Global Planning
Prof. Dr. Markus Lienkamp

Rainer Trauth, M. Sc.

Agenda

1. Introduction
2. **Global Planning:**
    1. **Dijkstra-Algorithm**
    2. A*-Algorithm
3. Behavior Planning
4. Global Trajectory Planning on the Racetrack
5. Summary

Global Planning
Dijkstra‘s Algorithm

- Invented by Edsger W. Dijkstra in 1956 (published 1959) [1]
- Finding optimal path from source node to destination node
- Graph-based algorithm
- Informed, complete and optimal path search algorithm
- Assumptions:
  - All edges are weighted by costs
  - No negative weighted costs
  - Path to start node has no costs
  - Starting node has no predecessor
  - For initialization, costs to all other nodes are infinite

[1] Dijkstra, EW 1959, 'A note on two problems in connexion with graphs', *Numerische Mathematik*, vol. 1, pp. 269-271. https://doi.org/10.1007/BF01386390

Additional Slides

A search algorithm is informed if it employs additional information about the problem that has the potential of guiding the search toward its goal. A search algorithm is complete if it is guaranteed to find an existing solution in a finite amount of time. A search algorithm is optimal if it is guaranteed to find an optimal solution in a finite amount of time.

Global Planning
Dijkstra‘s Algorithm

1. Initialization: Starting node=0, other nodes=∞
2. Distance of starting node is set to permanent, all other distances are temporarily
3. Setting starting node as active
4. Calculation of the temporary distances of the active node:
    1. Consideration of all neighbor nodes
    2. Summing up its distance with the weights of the edges
5. If calculated distance is smaller as the current one:
    1. Update the distance
    2. Set the current node as antecessor
6. Setting node with minimal temporary distance as active. Mark its distance as permanent.

Pseudocode:

```
1:  function Dijkstra(Graph, source):
2:      for each v in G:
3:          dist[v] ← ∞
4:          previous[v] ← Null
5:      dist[source] := 0
6:      while G is not empty:
7:          u := node in G with smallest dist[]
8:          remove u from G
9:          for each neighbor v of u:
10:             alt := dist[u] + dist_between(u, v)
11:             if alt < dist[v]:
12:                 dist[v] := alt
13:                 previous[v] := u
14:     return dist[], previous[]
```

Global Planning
Dijkstra‘s Algorithm

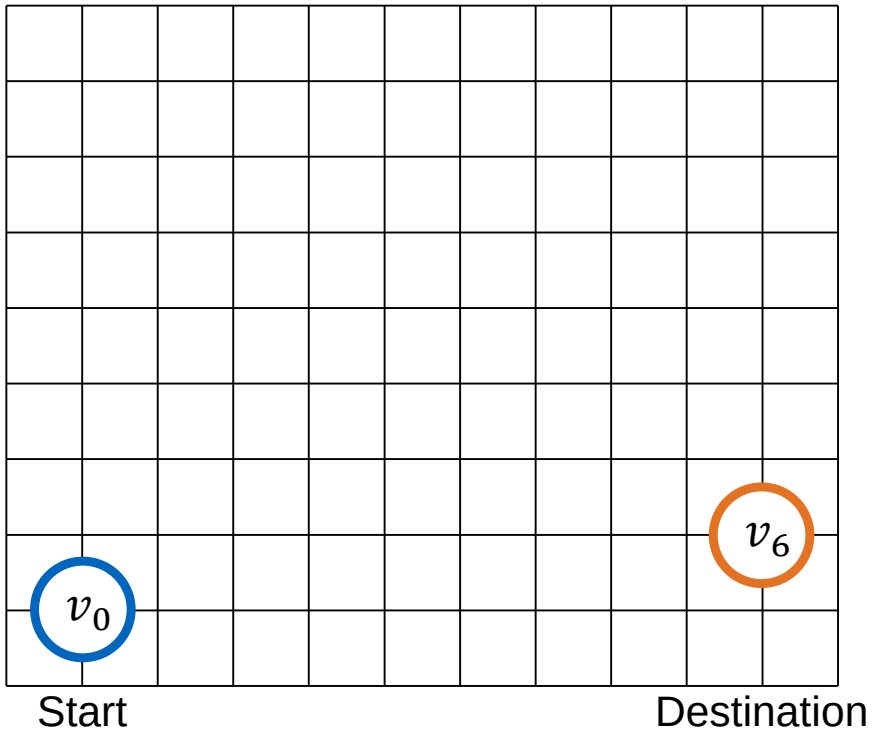
1. Initialization: Starting node=0, other nodes=∞
2. Distance of starting node is set to permanent, all other distances are temporarily
3. Setting starting node as active
4. Calculation of the temporary distances of the active node:
   1. Consideration of all ne
   2. Summing u
      of the edge
5. If calculated dista current one:
   1. Update the distance
   2. Set the current node as antecessor
6. Setting node with minimal temporary distance as active. Mark its distance as permanent.

Explore all shortest paths until destination is reached

Pseudocode:

```
1:  function Dijkstra(Graph, source):
2:      for each v in G:
3:                                  ← ∞
                                    [v] ← Null
                                    ] := 0
                                    ot empty:
                                    ode in G with smallest dist[]
            remove u from G
9:          for each neighbor v of u:
10:             alt := dist[u] + dist_between(u, v)
11:             if alt < dist[v]:
12:                 dist[v] := alt
13:                 previous[v] := u
14:     return dist[], previous[]
```

Global Planning
Dijkstra‘s Algorithm



d=distance
p=predecessor
c=considered

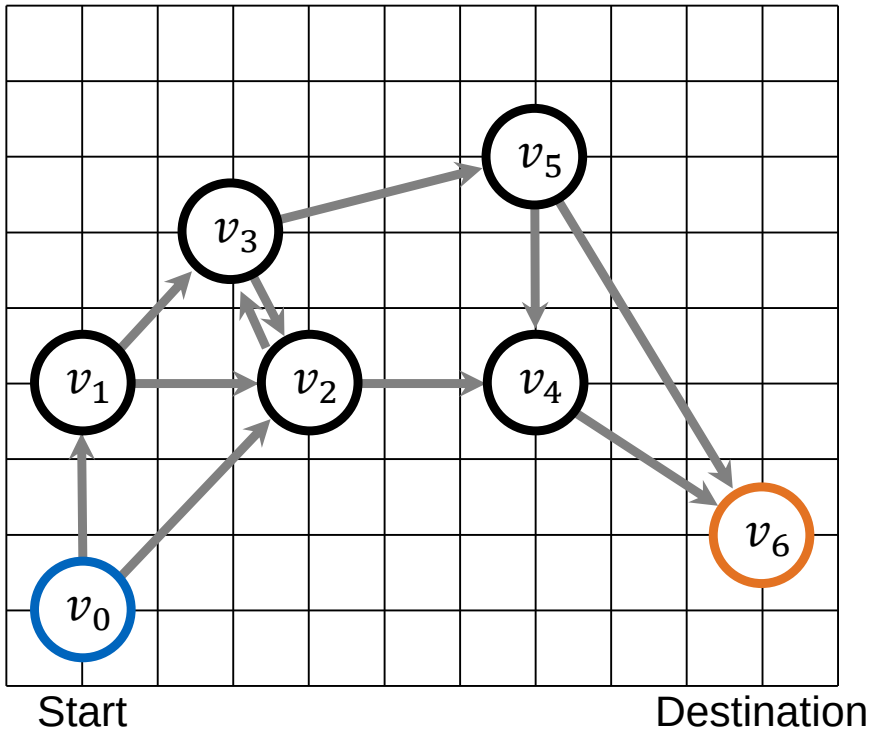
Step 0:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
d[v]	0	∞	∞	∞	∞	∞	∞
p[v]	-	-	-	-	-	-	-
c[v]	-	-	-	-	-	-	-

Priority Queue:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
d[v]	0	∞	∞	∞	∞	∞	∞

Global Planning
Dijkstra‘s Algorithm



d=distance
p=predecessor
c=considered

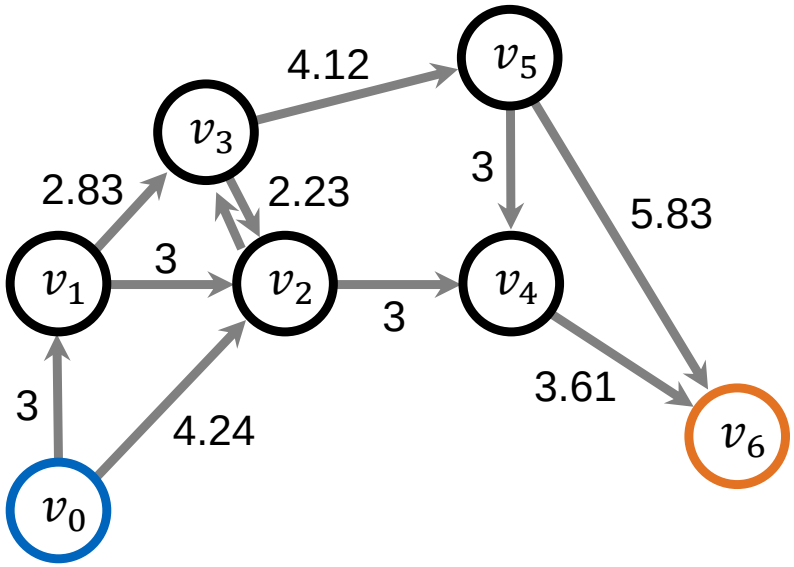
Step 0:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
d[v]	0	∞	∞	∞	∞	∞	∞
p[v]	-	-	-	-	-	-	-
c[v]	-	-	-	-	-	-	-

Priority Queue:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
d[v]	0	∞	∞	∞	∞	∞	∞

Global Planning
Dijkstra‘s Algorithm



d=distance
p=predecessor
c=considered

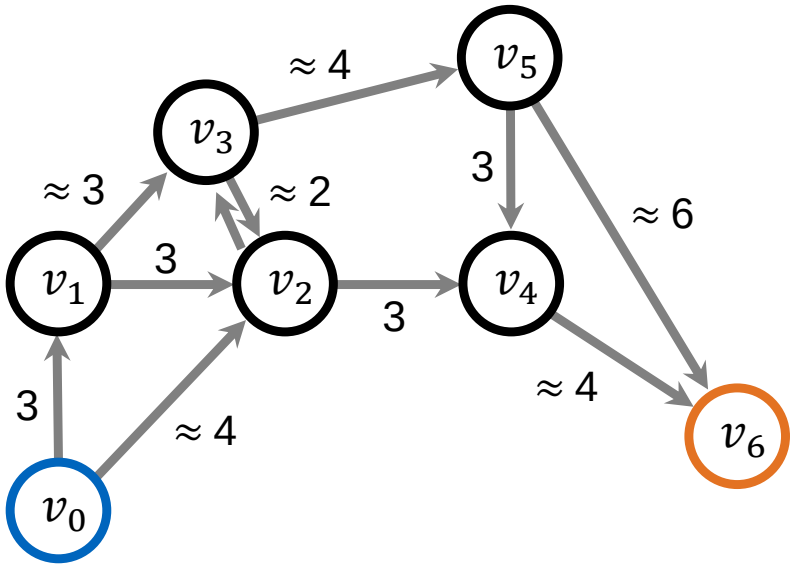
Step 0:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
d[v]	0	∞	∞	∞	∞	∞	∞
p[v]	-	-	-	-	-	-	-
c[v]	-	-	-	-	-	-	-

Priority Queue:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
d[v]	0	∞	∞	∞	∞	∞	∞

Global Planning
Dijkstra‘s Algorithm



d=distance
p=predecessor
c=considered

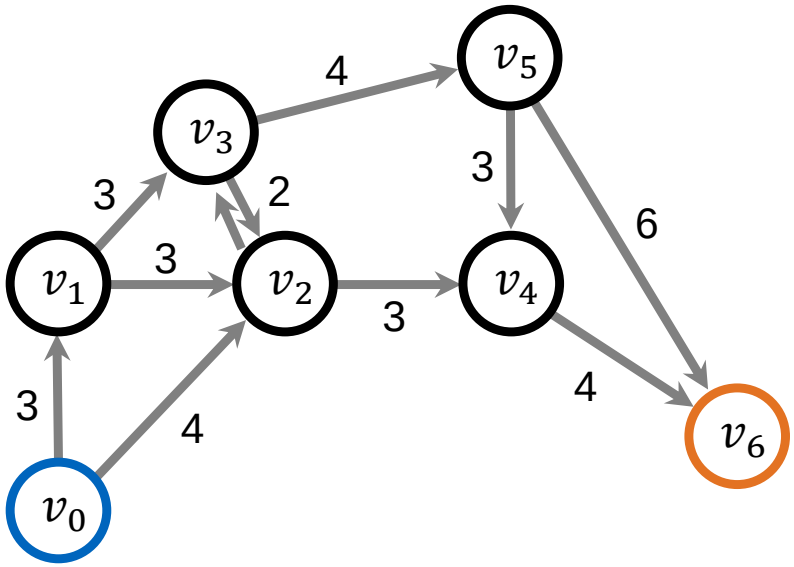
Step 0:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
d[v]	0	∞	∞	∞	∞	∞	∞
p[v]	-	-	-	-	-	-	-
c[v]	-	-	-	-	-	-	-

Priority Queue:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
d[v]	0	∞	∞	∞	∞	∞	∞

Global Planning
Dijkstra‘s Algorithm



d=distance
p=predecessor
c=considered

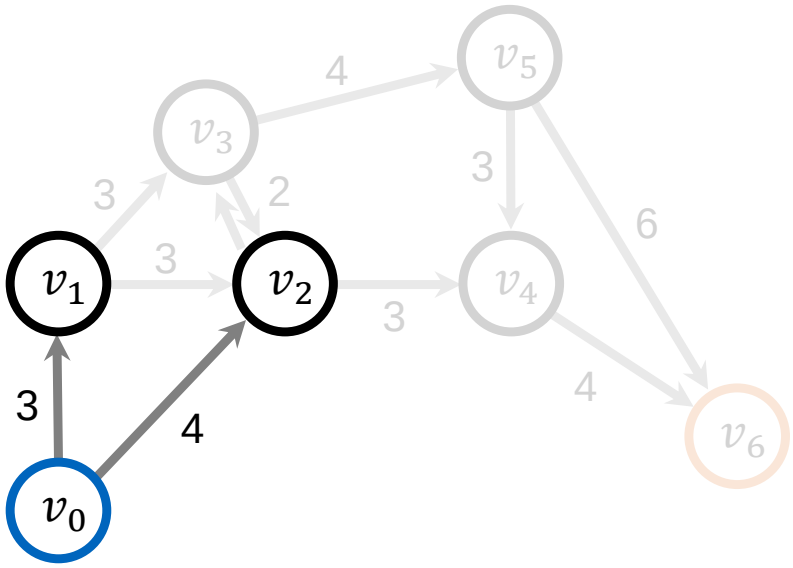
Step 0:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
d[v]	0	∞	∞	∞	∞	∞	∞
p[v]	-	-	-	-	-	-	-
c[v]	-	-	-	-	-	-	-

Priority Queue:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
d[v]	0	∞	∞	∞	∞	∞	∞

Global Planning
Dijkstra‘s Algorithm



d=distance
p=predecessor
c=considered

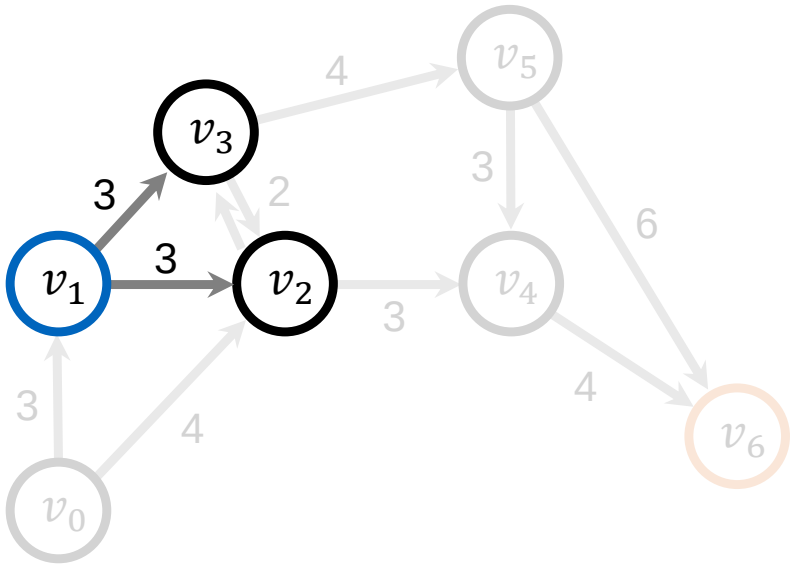
Step 1:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
d[v]	0	3	4	∞	∞	∞	∞
p[v]	-	v_0	v_0	-	-	-	-
c[v]	✓	-	-	-	-	-	-

Priority Queue:

v	v_1	v_2	v_3	v_4	v_5	v_6
d[v]	3	4	∞	∞	∞	∞

Global Planning
Dijkstra‘s Algorithm



d=distance
p=predecessor
c=considered

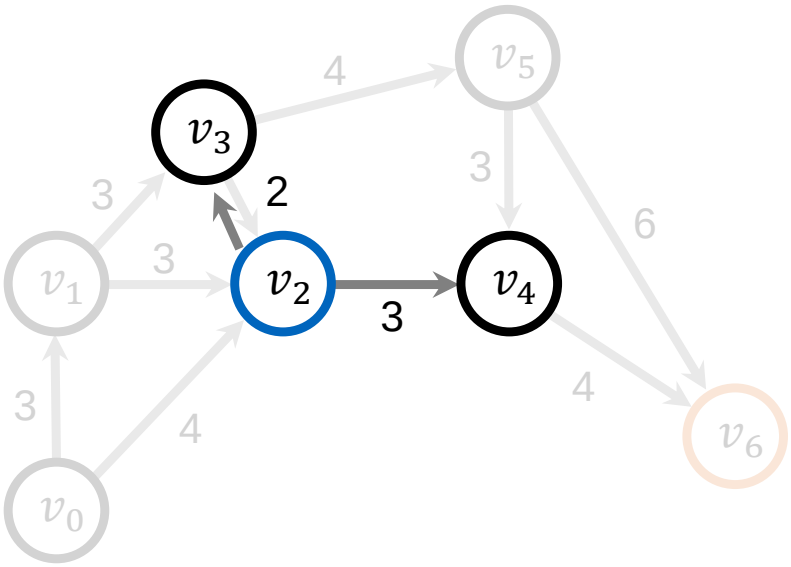
Step 2:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
d[v]	0	3	4	6	∞	∞	∞
p[v]	-	v_0	v_0	v_1	-	-	-
c[v]	✓	✓	-	-	-	-	-

Priority Queue:

v	v_2	v_3	v_4	v_5	v_6
d[v]	4	6	∞	∞	∞

Global Planning
Dijkstra‘s Algorithm



d=distance
p=predecessor
c=considered

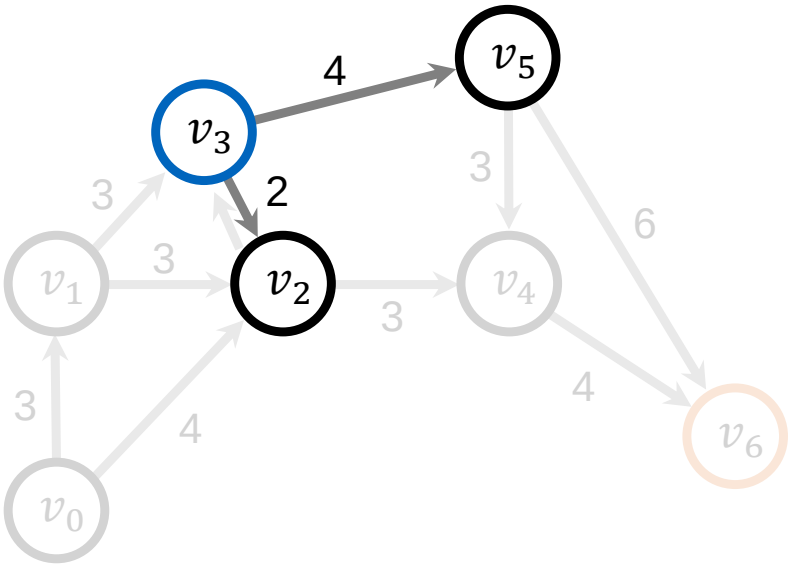
Step 3:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
d[v]	0	3	4	6	7	∞	∞
p[v]	-	v_0	v_0	v_1	v_2	-	-
c[v]	✓	✓	✓	-	-	-	-

Priority Queue:

v	v_3	v_4	v_5	v_6
d[v]	6	7	∞	∞

Global Planning
Dijkstra‘s Algorithm



d=distance
p=predecessor
c=considered

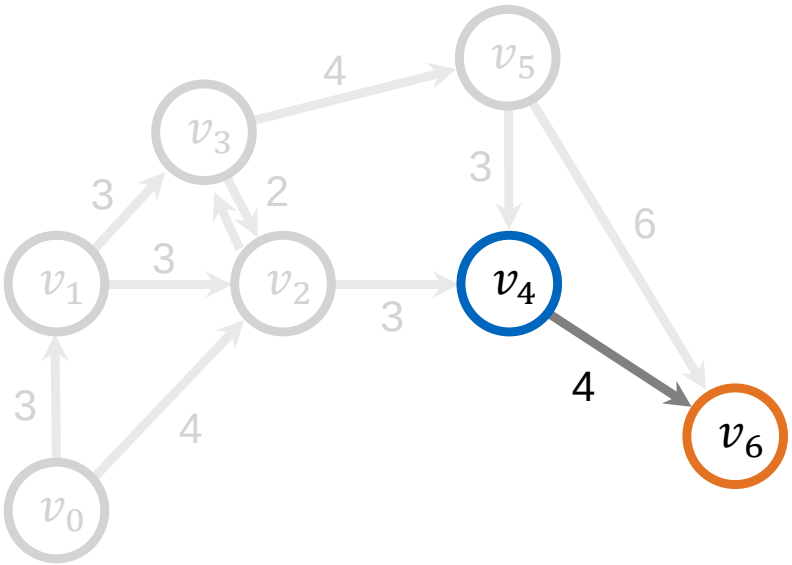
Step 4:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
d[v]	0	3	4	6	7	10	∞
p[v]	-	v_0	v_0	v_1	v_2	v_3	-
c[v]	✓	✓	✓	✓	-	-	-

Priority Queue:

v	v_4	v_5	v_6
d[v]	7	10	∞

Global Planning
Dijkstra‘s Algorithm



d=distance
p=predecessor
c=considered

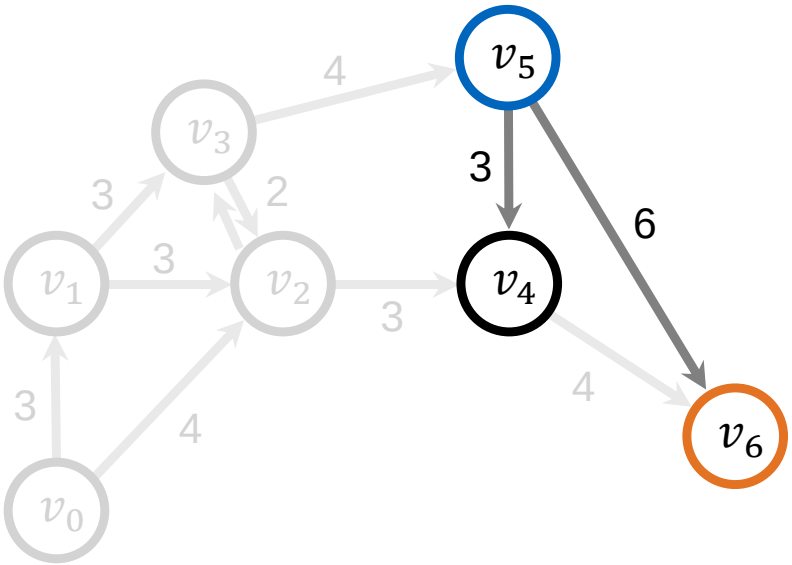
Step 5:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
d[v]	0	3	4	6	7	10	11
p[v]	-	v_0	v_0	v_1	v_2	v_3	v_4
c[v]	✓	✓	✓	✓	✓	-	-

Priority Queue:

v	v_5	v_6
d[v]	10	11

Global Planning
Dijkstra‘s Algorithm



d=distance
p=predecessor
c=considered

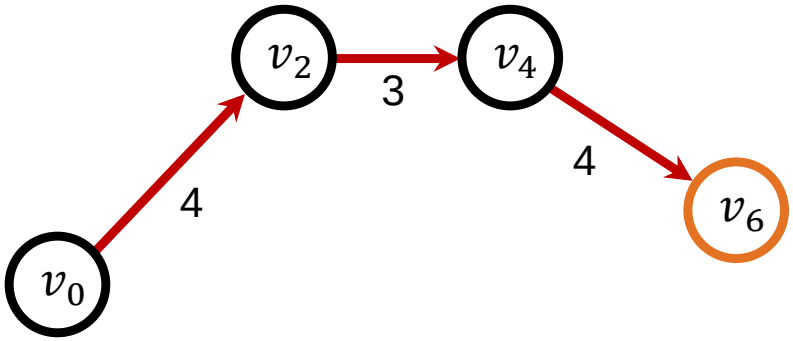
Step 6:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
d[v]	0	3	4	6	7	10	11
p[v]	-	v_0	v_0	v_1	v_2	v_3	v_4
c[v]	✓	✓	✓	✓	✓	✓	-

Priority Queue:

v	v_6
d[v]	11

Global Planning
Dijkstra‘s Algorithm



d=distance
p=predecessor
c=considered

Step 7:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
d[v]	0	3	4	6	7	10	11
p[v]	-	v_0	v_0	v_1	v_2	v_3	v_4
c[v]	✓	✓	✓	✓	✓	✓	✓

Priority Queue:

v	-
d[v]	-

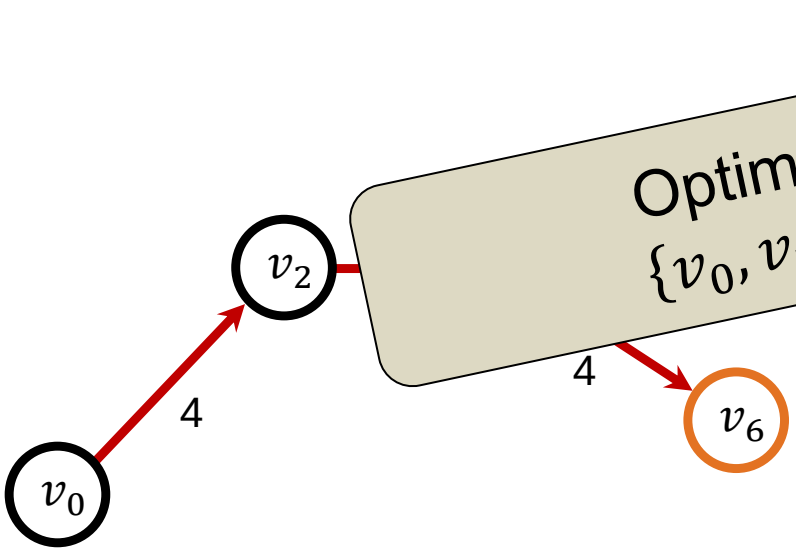
$Q = \emptyset$

Global Planning
Dijkstra‘s Algorithm

d=distance
p=predecessor
c=considered

Step 7:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
d[v]	0	3	4	6	7	10	11
	v_0	v_1	v_2	v_3	v_4		
	✓	✓	✓	✓	✓		

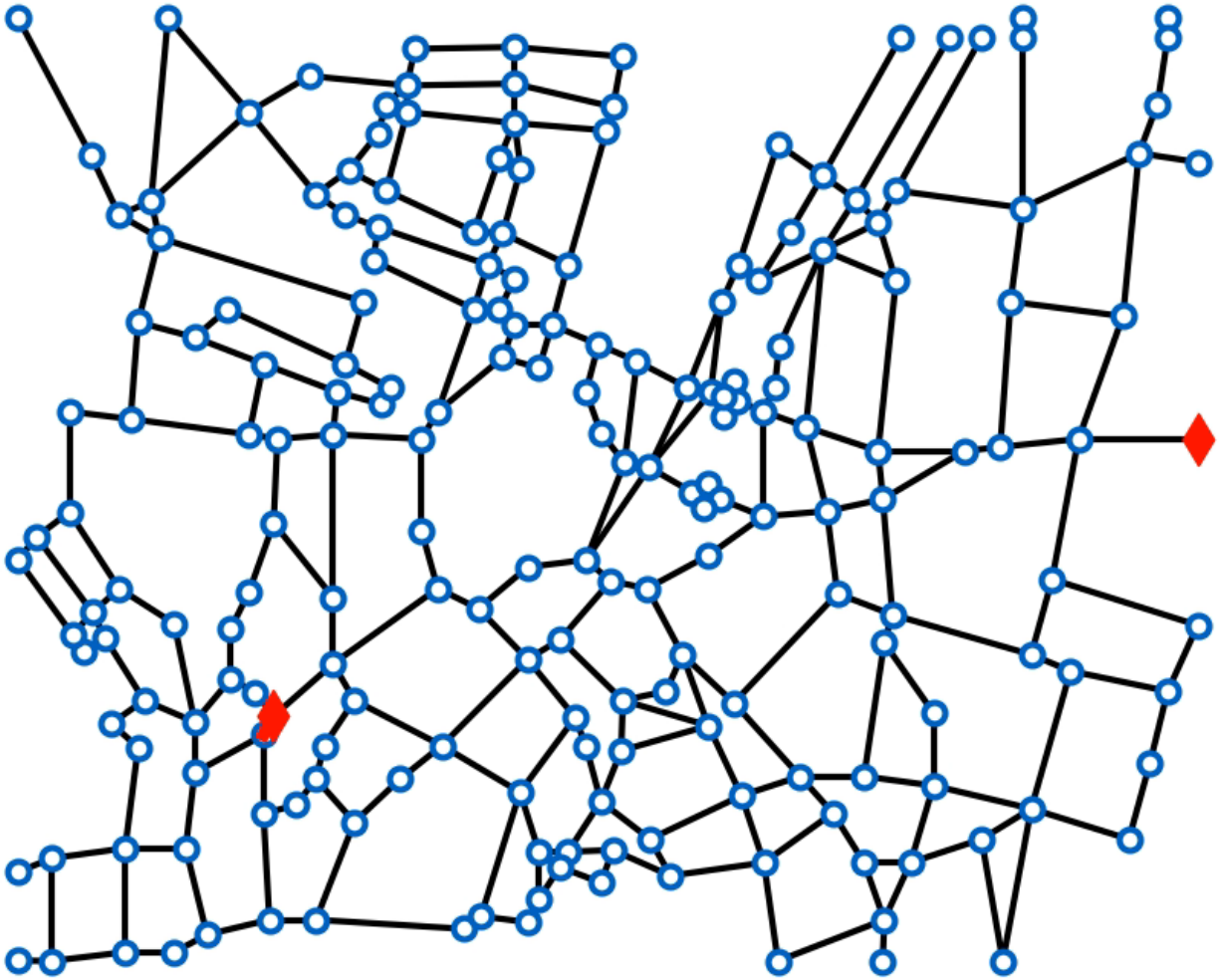


Priority Queue:

v	-
d[v]	-

$Q = \emptyset$

Global Planning
Dijkstra‘s Algorithm



Additional Slides

The Dijkstra algorithm explores the street network in a circular fashion, generating the most cost efficient paths until one of them terminates in the goal node. Therefore, it is not only guaranteed to find a feasible path, but also to find a cost optimal path. However, it does not employ any information about the distance to the goal. Hence, all paths that are shorter than or equally short to the optimal path have to be examined regardless of their potential to reach the destination with optimum cost. I.e. a partial path that starts off in the wrong cardinal direction will be considered prior to a path starting off into the general direction of the destination if the latter is longer than the first, even though a human would directly recognize the error in this approach.

Global Planning
Prof. Dr. Markus Lienkamp

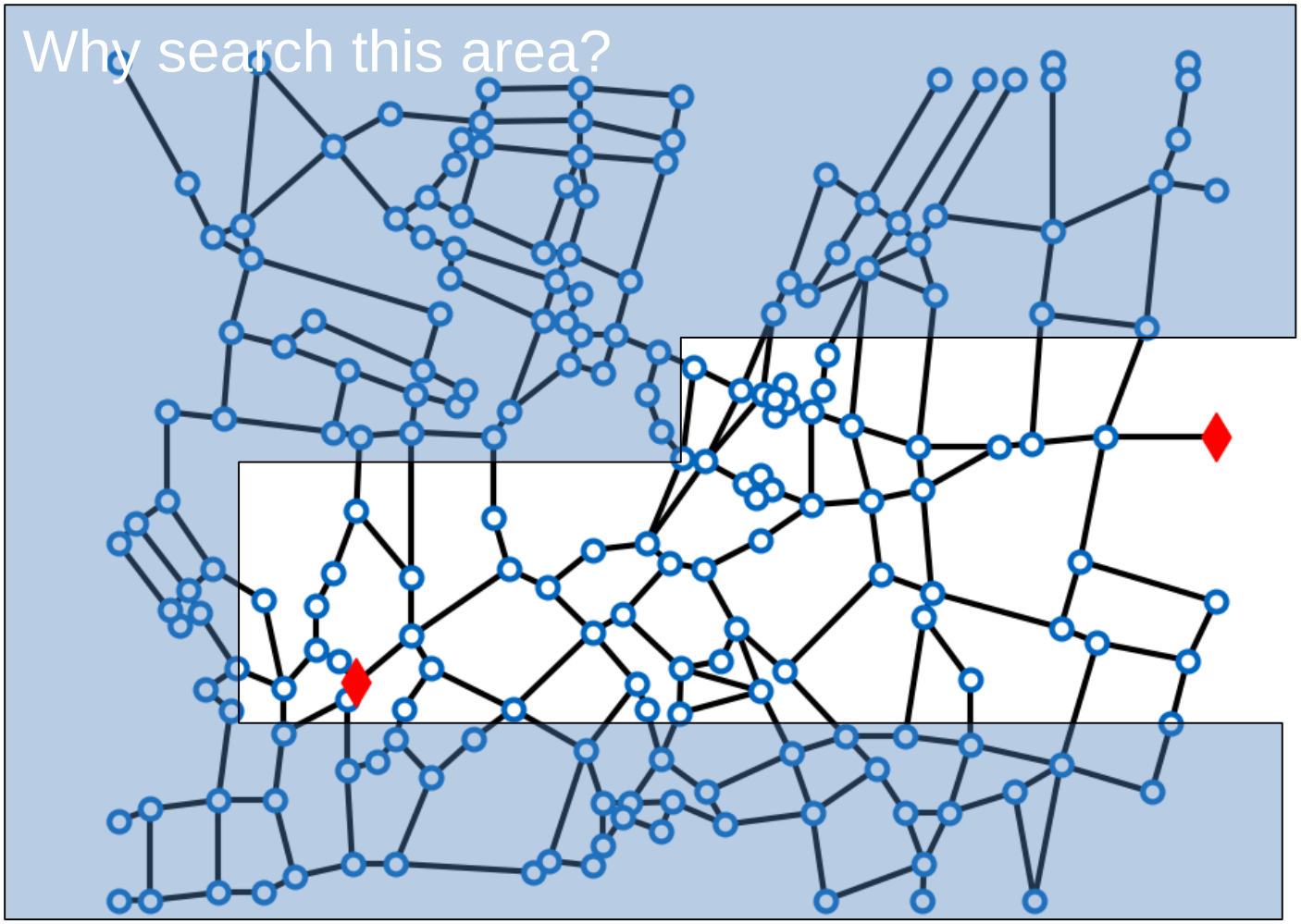
Rainer Trauth, M. Sc.

Agenda

1. Introduction
2. **Global Planning:**
    1. Dijkstra-Algorithm
    2. **A*-Algorithm**
3. Behavior Planning
4. Global Trajectory Planning on the Racetrack
5. Summary

Global Planning

A*-Algorithm



Global Planning
A*-Algorithm

- Invented by Peter Hart, Nils Nilsson and Bertram Raphael in 1968 [1]
- In literature frequently described as extension of Dijkstra's algorithm
- Informed, complete and optimal path search algorithm
- Differences to Dijkstra's algorithm:
  - Informed heuristic search algorithm
  - An estimation function accelerates the search process
  - Node costs = distance to the start node plus the estimated distance to the destination node. Node with lowest overall costs has priority
- Estimation function:
  - Free choice of the estimation function
  - Has to be feasible. Estimator must never overestimate the cost of a route

[1] P. E. Hart, N. J. Nilsson and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," in *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100-107, July 1968, doi: 10.1109/TSSC.1968.300136.

Global Planning

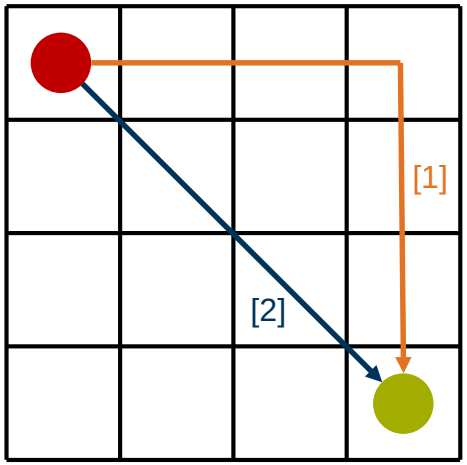
A*-Algorithm – Cost estimation function

$$f(v) = g(v) + h(v)$$

- $f(v) = total\ estimated\ cost\ of\ path\ through\ node\ v$
- $g(v) = cost\ so\ far\ to\ reach\ v$
- $h(v) = estimated\ cost\ from\ v\ to\ destination$

Cost estimation heuristics:

- Exact heuristics → time consuming
- Approximation heuristics:
  - The Manhattan Distance Heuristic [1]
  - Euclidean Distance [2]



Global Planning
A*-Algorithm

Create open-list and closed list

Add starting node to open-list

Repeat algorithm until destination is found

1. Calculation of the temporary distances of the active node:
    1. Consideration of all neighbor nodes
    2. Summing up its distance with the weights of the edges plus estimated distance to destination
2. If calculated distance is smaller than current:
    1. Update the distance
    2. Set the current node as antecessor
    3. Add node to open-list
3. Add current node to closed-list
4. Proceed with minimal temporary distance node as active node

Pseudocode, Main Part:

```
1:  make open-list with starting node
2:  make empty closed-list
3:  while destination not reached:
4:      consider node with min f[v]
5:      for each child u of current node v :
6:          set child costs f[u]
7:          if child u is in open-list:
8:              if g[u] < g[v]:
9:                  continue to line 17
10:         else if child u is in closed-list:
11:             if g[u] < g[v] :
12:                 add u to open-list
13:         else:
14:             add u to open-list
15:         Set g[u] = successor current cost
16:         Set prev[u] = v
17:     add v to closed-list
```

Global Planning
A*-Algorithm

Create open-list and closed list

Add starting node to open-list

Repeat algorithm until destination is found

1. Calculation of the temporary distances of the active node:
   1. Consideration of all neighbor nodes
   2. Summing up its distan… of the edge… destination…
2. If calculated dista… …rrent:
   1. Update the …
   2. Set the current node as antecessor
   3. Add node to open-list
3. Add current node to closed-list
4. Proceed with minimal temporary distance node as active node

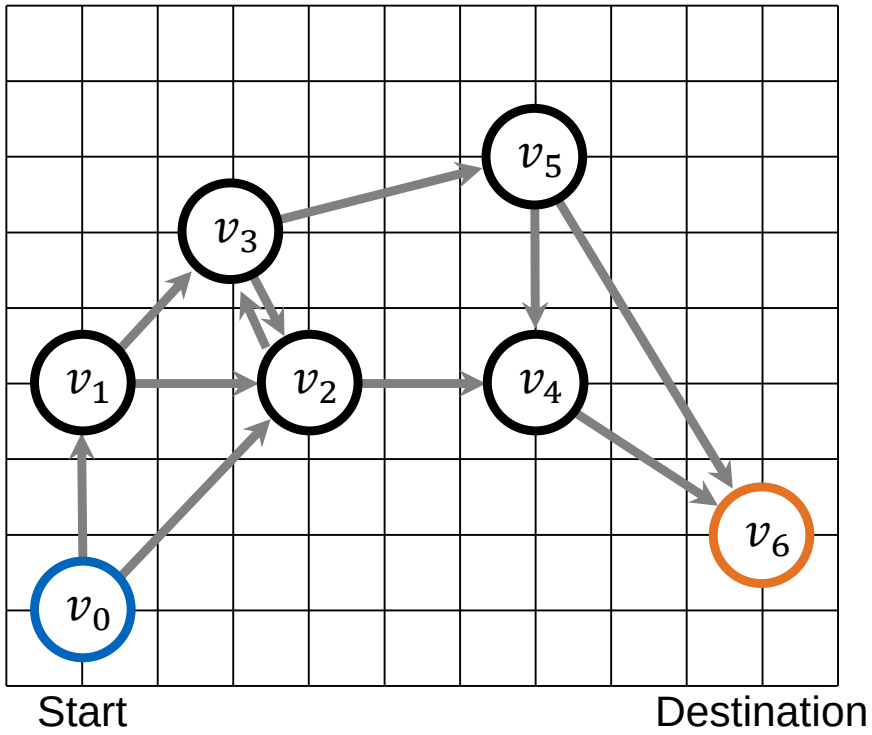
Compare Lower Bound Estimate: Airline Distance

Pseudocode, Main Part:

```
1:  make open-list with starting node
2:  make empty closed-list
3:  while destination not reached:
4:      consider… with min f[v]
5:      …u of current node v :
6:      …sts f[u]
7:      … open-list:
8:      …g[v]:
9:          continue to line 17
10:     else if child u is in closed-list:
11:         if g[u] < g[v] :
12:             add u to open-list
13:     else:
14:         add u to open-list
15:     Set g[u] = successor current cost
16:     Set prev[u] = v
17: add v to closed-list
```

Global Planning
A*-Algorithm

g=distance so far
h=v to destination
f=total estimated cost
p=predecessor
c=considered



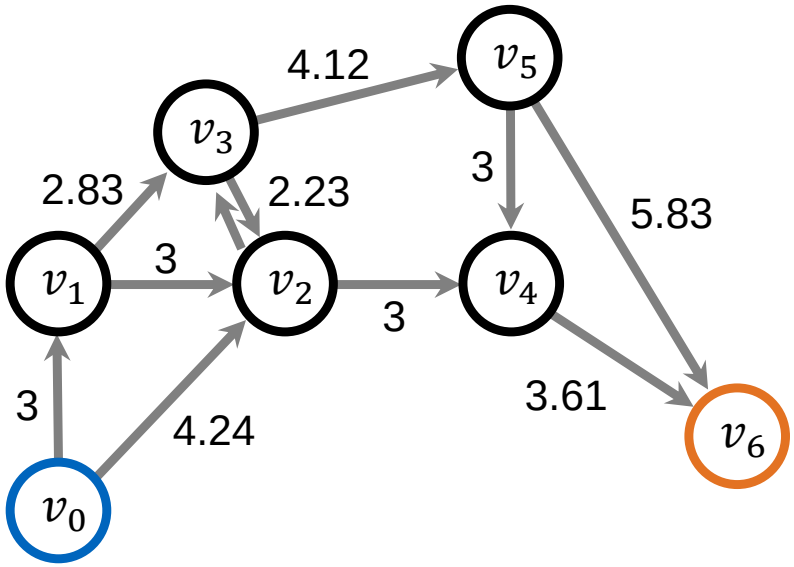
Step 0:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
g[v]	0	∞	∞	∞	∞	∞	∞
h[v]	-	-	-	-	-	-	-
f[v]	-	-	-	-	-	-	-
p[v]	-	-	-	-	-	-	-
c[v]							

Priority Queue:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
f[v]	-	∞	∞	∞	∞	∞	∞

Global Planning
A*-Algorithm



g=distance so far
h=v to destination
f=total estimated cost
p=predecessor
c=considered

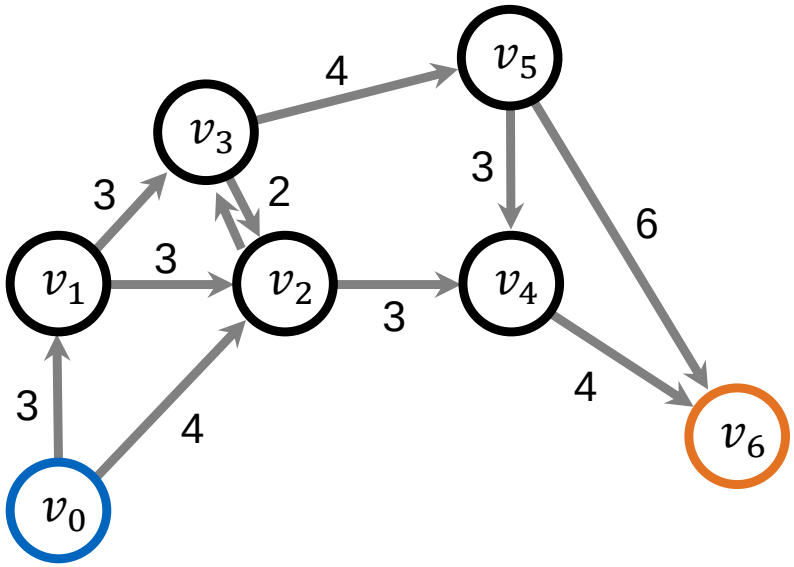
Step 0:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
g[v]	0	∞	∞	∞	∞	∞	∞
h[v]	-	-	-	-	-	-	-
f[v]	-	-	-	-	-	-	-
p[v]	-	-	-	-	-	-	-
c[v]							

Priority Queue:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
f[v]	-	∞	∞	∞	∞	∞	∞

Global Planning
A*-Algorithm



g=distance so far
h=v to destination
f=total estimated cost
p=predecessor
c=considered

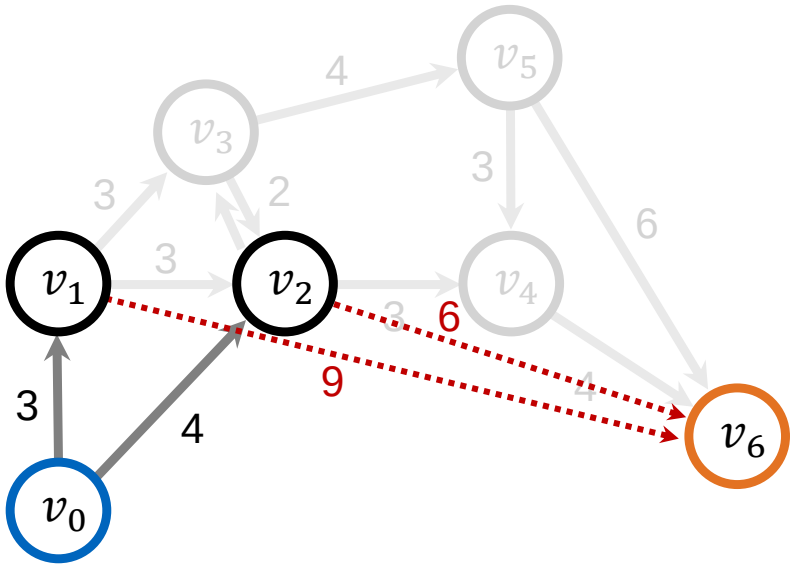
Step 0:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
g[v]	0	∞	∞	∞	∞	∞	∞
h[v]	-	-	-	-	-	-	-
f[v]	-	-	-	-	-	-	-
p[v]	-	-	-	-	-	-	-
c[v]							

Priority Queue:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
f[v]	-	∞	∞	∞	∞	∞	∞

Global Planning
A*-Algorithm



g=distance so far
h=v to destination
f=total estimated cost
p=predecessor
c=considered

Step 1:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
g[v]	0	3	4	∞	∞	∞	∞
h[v]	9	9	6	-	-	-	-
f[v]	9	12	10	-	-	-	-
p[v]	-	v_0	v_0	-	-	-	-
c[v]	✓						

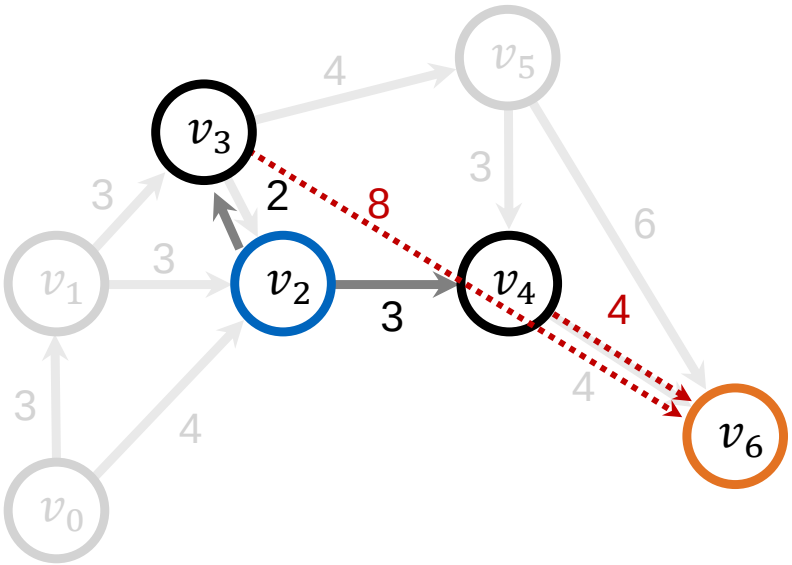
Priority Queue:

v	v_1	v_2	v_3	v_4	v_5	v_6
f[v]	12	10	∞	∞	∞	∞

Euclidean Airline Distance

Global Planning
A*-Algorithm

g=distance so far
h=v to destination
f=total estimated cost
p=predecessor
c=considered



Step 2:

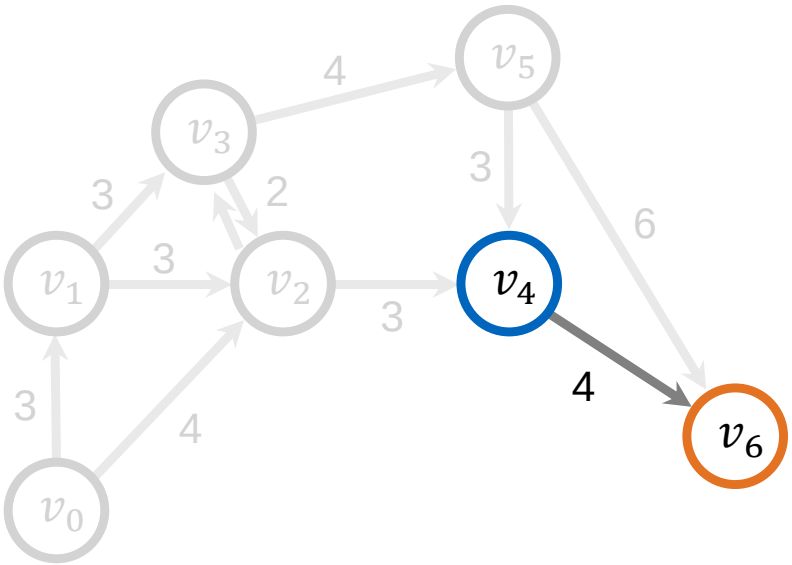
v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
g[v]	0	3	4	6	7	∞	∞
h[v]	9	9	6	8	4	-	-
f[v]	9	12	10	14	11	-	-
p[v]	-	v_0	v_0	v_2	v_2	-	-
c[v]	✓		✓				

Priority Queue:

v	v_1	v_3	v_4	v_5	v_6
f[v]	12	14	11	∞	∞

Euclidean Airline Distance

Global Planning
A*-Algorithm



g=distance so far
h=v to destination
f=total estimated cost
p=predecessor
c=considered

Step 3:

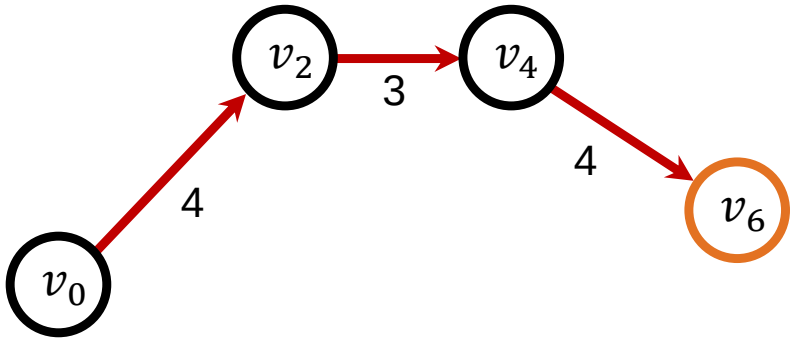
v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
g[v]	0	3	4	6	7	∞	11
h[v]	9	9	6	8	4	-	0
f[v]	9	12	10	14	11	-	11
p[v]	-	v_0	v_0	v_2	v_2	-	v_4
c[v]	✓		✓		✓		

Priority Queue:

v	v_1	v_3	v_5	v_6
f[v]	12	14	∞	11

Global Planning
A*-Algorithm

g=distance so far
h=v to destination
f=total estimated cost
p=predecessor
c=considered



Step 3:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
g[v]	0	3	4	6	7	∞	11
h[v]	9	9	6	8	4	-	0
f[v]	9	12	10	14	11	-	11
p[v]	-	v_0	v_0	v_2	v_2	-	v_4
c[v]	✓		✓		✓		

Priority Queue:

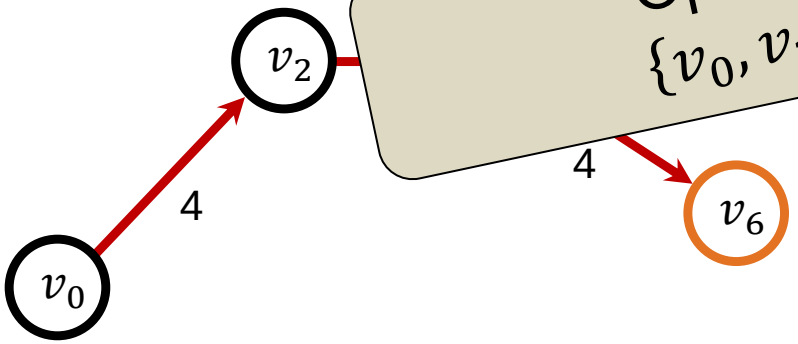
v	v_1	v_3	v_5	v_6
f[v]	12	14	∞	11

Global Planning
A*-Algorithm

g=distance so far
h=v to destination
f=total estimated cost
p=predecessor
c=considered

Step 3:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
g[v]	0	3	4	6	7	∞	11
				8	4	-	0
				14	11	-	11
	-	v_0	v_0	v_2	v_2	-	v_4
c[v]	✓		✓		✓		

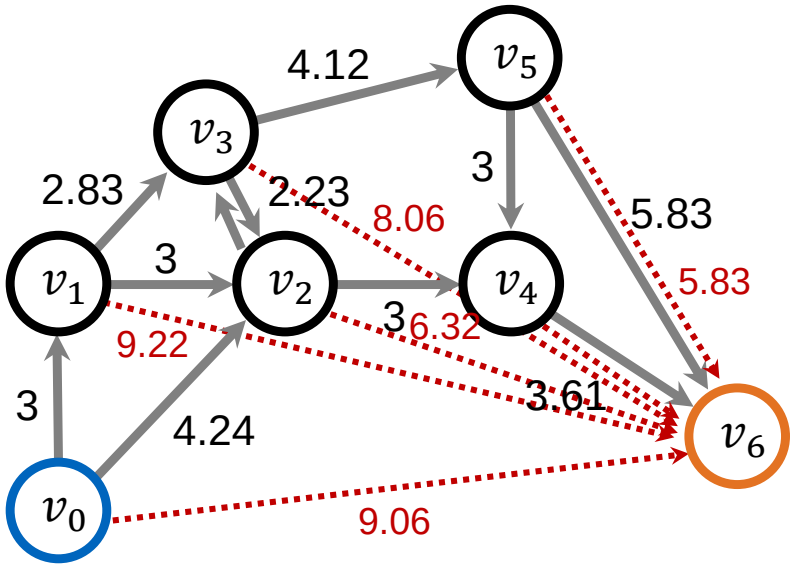


Optimal Path
$\{v_0, v_2, v_4, v_6\}$

Priority Queue:

In order to always obtain an optimal path, all nodes must be visited or heuristic must not overestimate the cost to reach goal

Global Planning
A*-Algorithm



Euclidean Airline Distance

g=distance so far
h=v to destination
f=total estimated cost
p=predecessor
c=considered

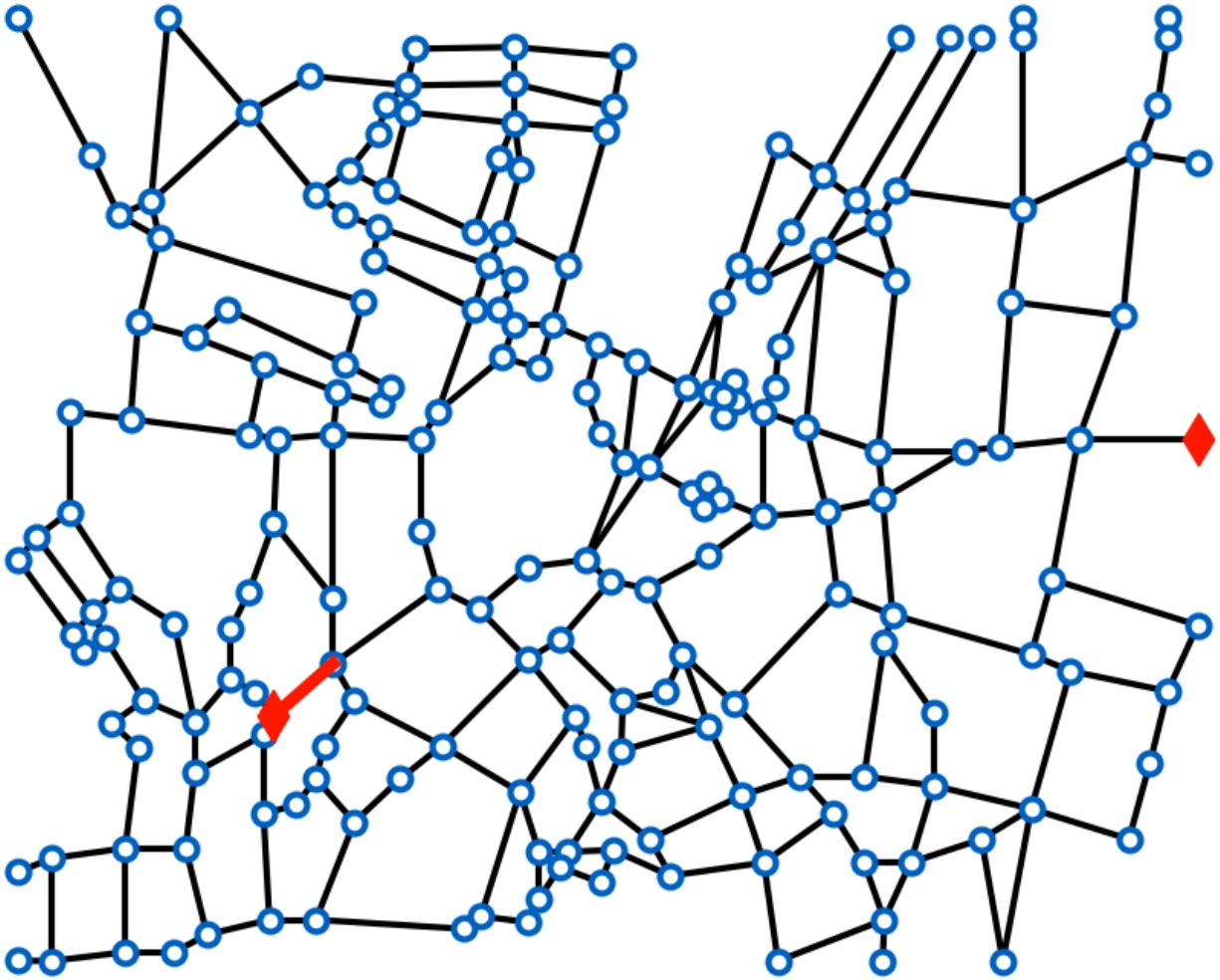
Step 0:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
g[v]	0	∞	∞	∞	∞	∞	∞
h[v]	9	6	5	4	5	2	-
f[v]	9	-	-	-	-	-	-
p[v]	-	-	-	-	-	-	-
c[v]							

Priority Queue:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
f[v]	9	∞	∞	∞	∞	∞	∞

Global Planning
A*-Algorithm



Additional Slides

The animation shows that A* converges much faster toward the destination than all of the previous algorithms. A* also favors paths in the general direction of the destination over paths that run in different directions. By doing so, the number of iterations is reduced, especially when working with large and highly connected networks in which the general direction is mostly the best one to follow (which holds for street networks).

A* explores all partial paths in the order of their potential to reach the destination with a minimum amount of steps. To guarantee an optimal solution, A* continues to explore the graph even if a feasible path has been found, because – unlike Dijkstra – it can not be sure that the first feasible path is an optimal path. In A*, the search process can only be terminated if there is no partial path left that could potentially reach the destination with less cost than the path already found.

Therefore, the estimate used in the algorithm has to be a lower bound estimate to not rule out partial paths that can potentially be shorter than a path already found. The airline distance is such a lower bound, because no set of street segments can be shorter than the airline distance between its start and the destination.

In real-life applications, A* or Dijkstra algorithms are used in navigation systems.

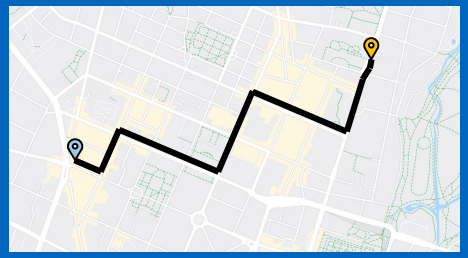
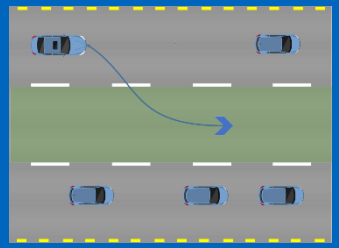
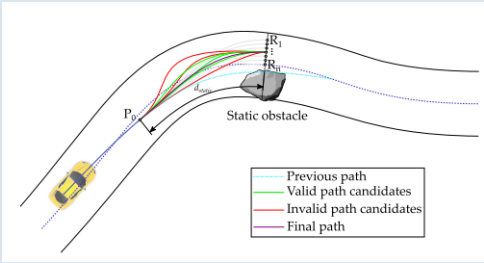
Global Planning
Prof. Dr. Markus Lienkamp

Rainer Trauth, M. Sc.

Agenda

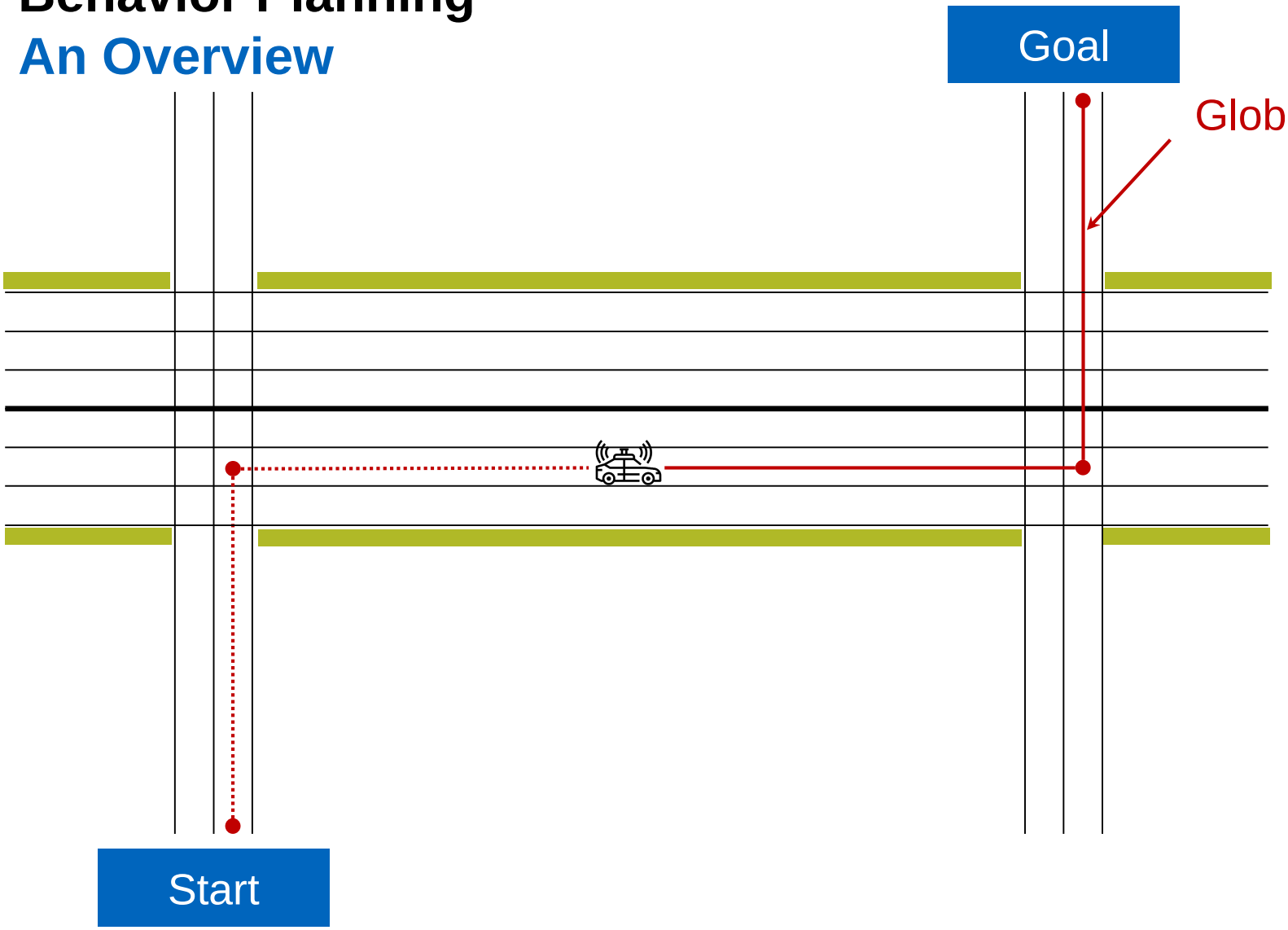
1. Introduction
2. Global Planning:
    1. Dijkstra-Algorithm
    2. A*-Algorithm
3. **Behavior Planning**
4. Global Trajectory Planning on the Racetrack
5. Summary

Behavior Planning
Definition of terms

Global Planning	Uses map for planning without information of local environment. Focus: hours to minutes.	
Behavior Planning	High-level description of the vehicle motion. Focus: minutes to seconds.	
Local Planning	Consideration of local objects. Deals with reactive decisions. Focus: seconds to milliseconds.	

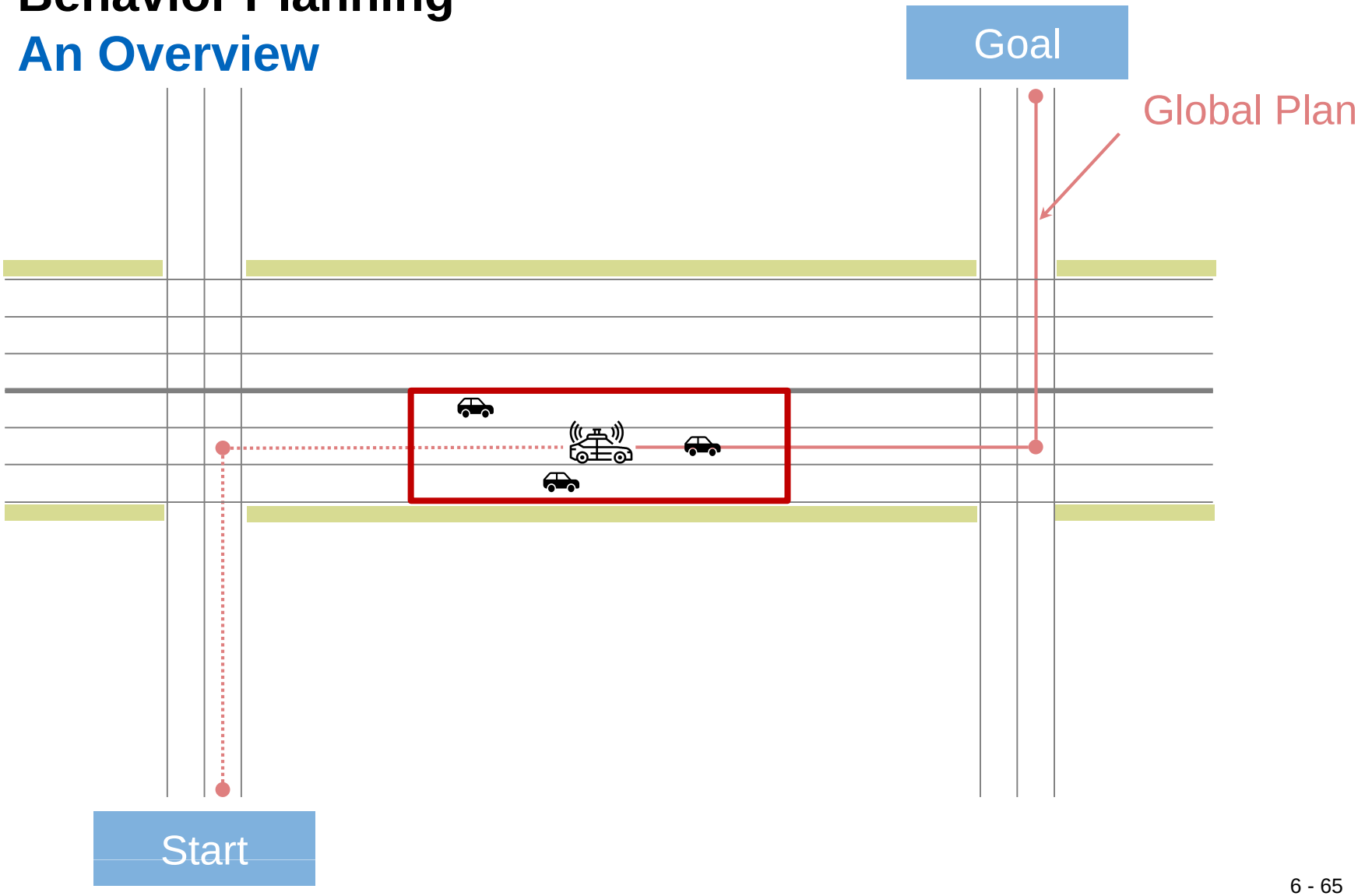
https://mapstyle.withgoogle.com/
https://medium.com/behavior-trees-for-path-planning-autonomous/behavior-trees-for-path-planning-autonomous-driving-8db1575fec2c

Behavior Planning
An Overview

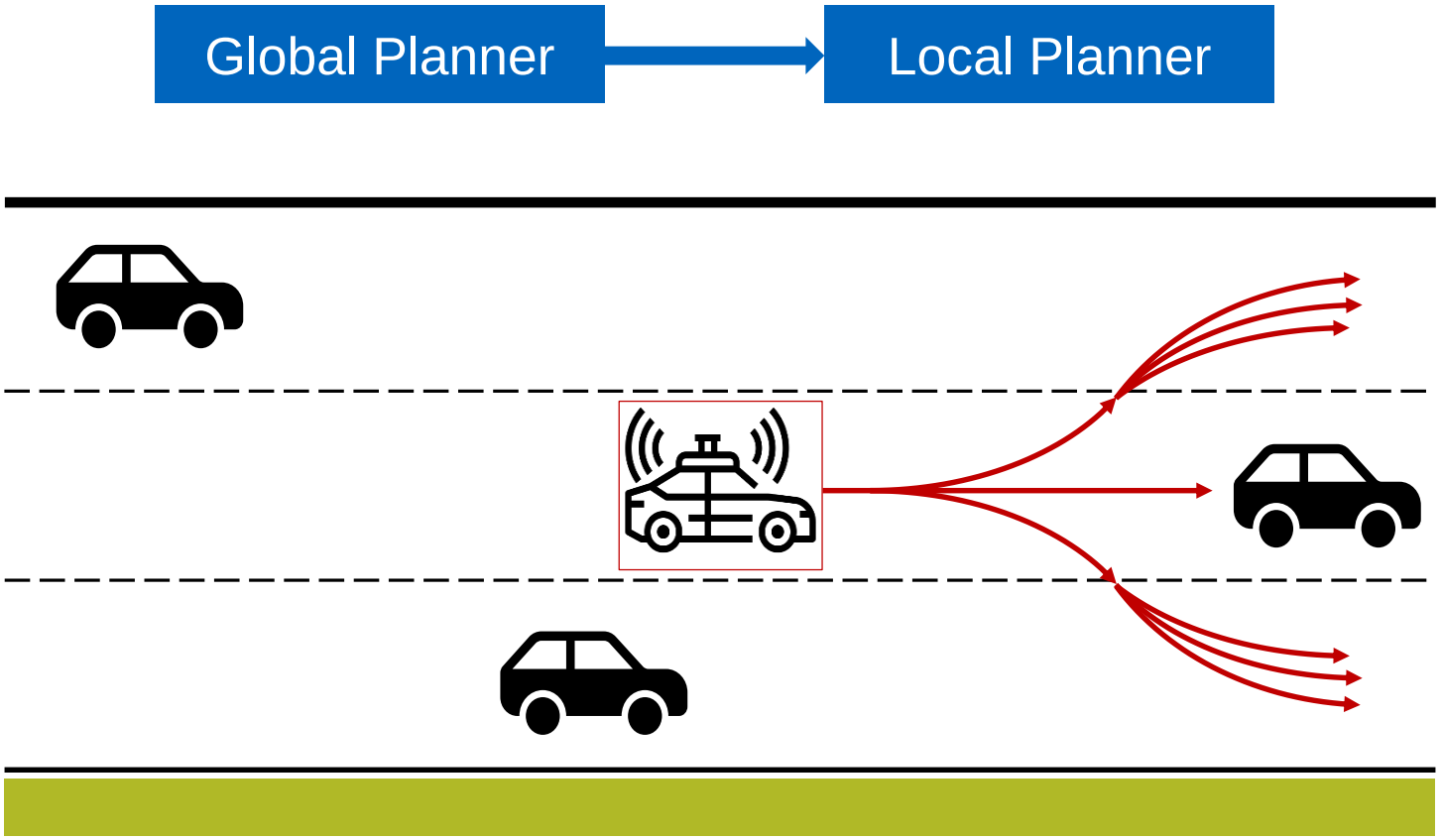


Behavior Planning

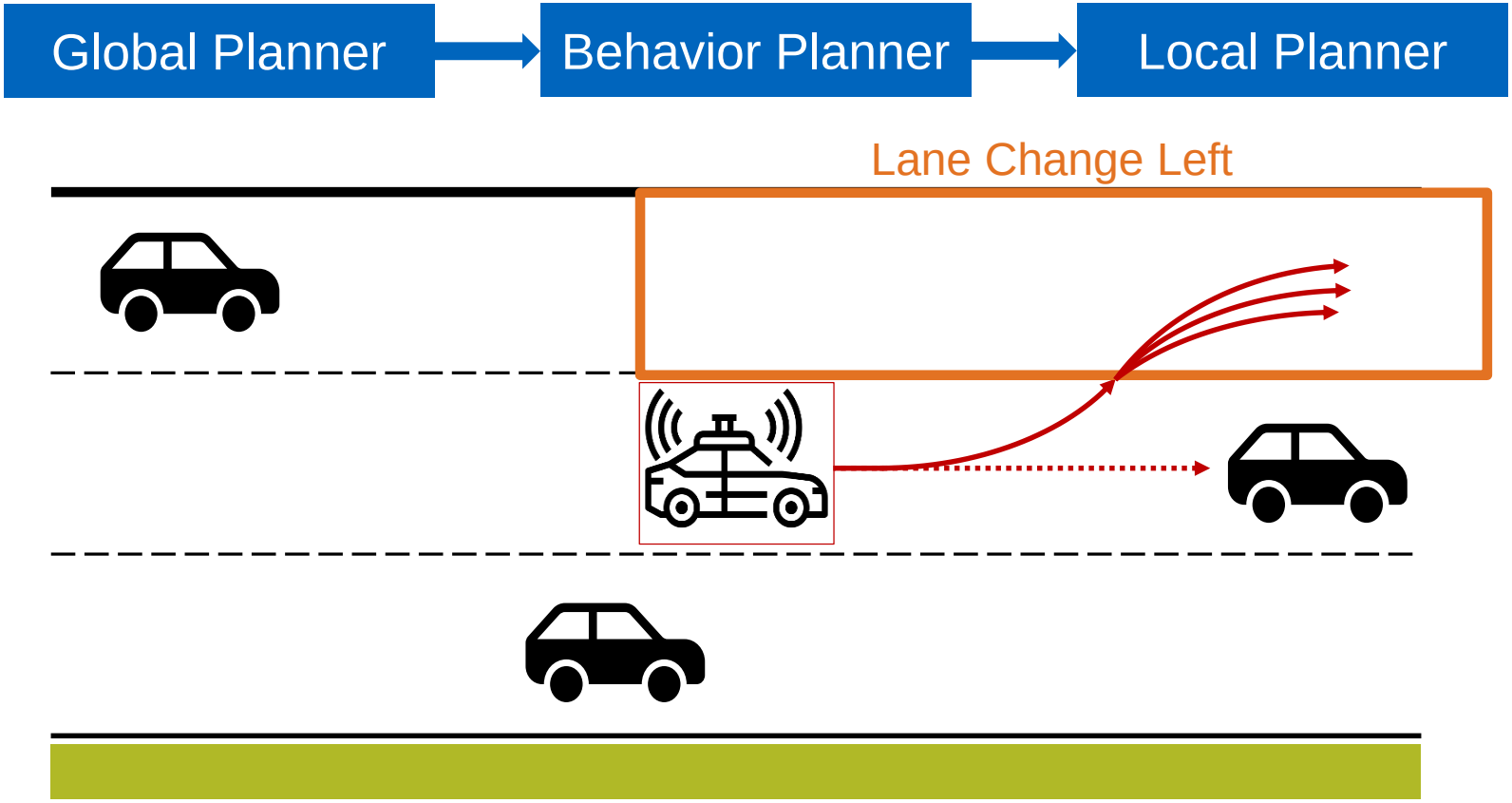
An Overview



Behavior Planning
An Overview



Behavior Planning
An Overview



Behavior Planning
An Overview

Plans **high-level** driving actions to safely achieve the driving mission under various driving situations

- Fills technical gap between Global Planner and Local Planner
- Considers rules of the road and static / dynamic objects around the vehicle

Decision of behavior can be determined by costs, e.g.:
→ Feasibility costs
→ Security costs
→ Legality costs
→ Comfort costs
→ Speed costs
→ …

Possible constraints:
→ Global objective
→ Road speed limit
→ Road lane boundaries
→ Stop locations
→ Set of interest vehicles
→ …

Behavior Planning
Finite-state Machine (FSM)

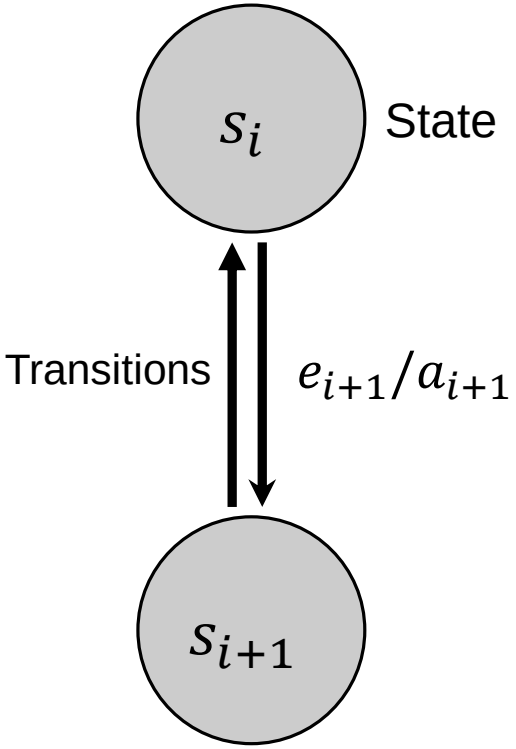
Behavior of the vehicle can be modeled by **Finite-state Machines**

- Finite-state Machine (FSM):
  - Mathematical model of computation
  - Consists of finite number of states
  - Behavior of the vehicle can be modeled by transitions between states.
  - Transitions based on current state and given input
  - Deterministic behavior
- Preconditions:
  - There is exactly one state per timestep
  - The time delay at the state transition is irrelevant

Behavior Planning
Finite-state Machine (FSM)

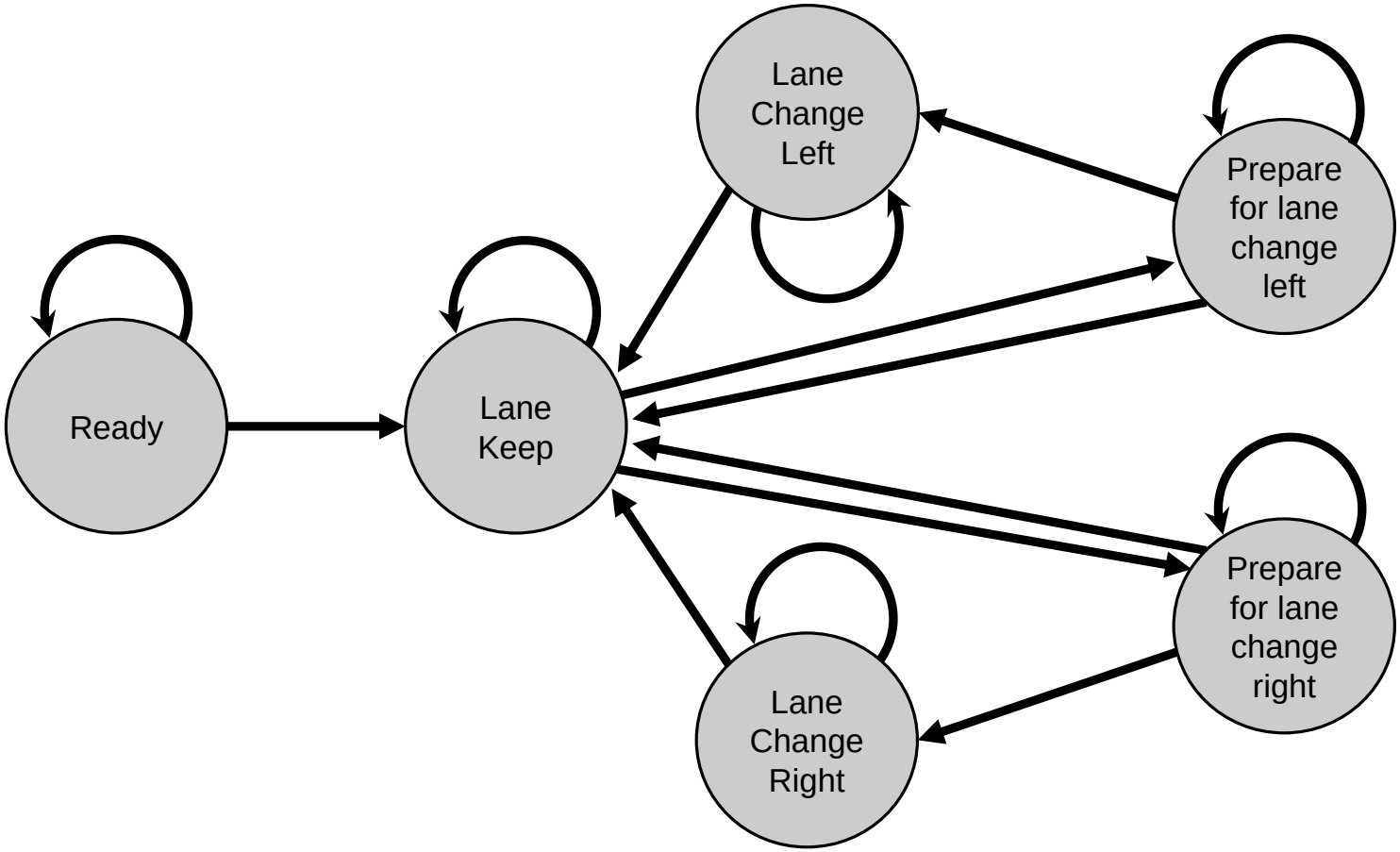
Mathematical model:

- Description by a tuple (S, E, A, F, δ)
- $s_1, s_2, s_3, \ldots \in S$: Set of states
- $e_1, e_2, e_3 \ldots \in E$: Set of inputs
- $a_1, a_2, a_3 \ldots \in A$: Set of actions
- $s_0 \in S$: Initial state
- $F \in S$: Set of final states
- $\delta: S \times E \rightarrow S \times A$: Transition function



Behavior Planning

Finite-state Machine (FSM) – An Example



https://towardsdatascience.com/

Behavior Planning
Finite-state Machine (FSM)

Advantages of Finite-state Machines:

- Limiting number of rule checks
- Clear in structure
- Easy to calibrate / optimize
- Simple implementation
- High flexibility
- Easy determination of reachability of a state

Disadvantages of Finite-state Machines:

- High knowledge of system design required
- Large FSMs hard to visualize
- Difficult transferability to other projects

Additional Slides

A finite state machine (FSM) is the realization of a control concept, which is based on an abstracted machine, which has a number of states, by which its operation is defined. This machine works by passing from one state to another state and by executing certain actions at such state transitions and in the persistence of states. The subsequent state results from the current state and an external event, e.g. a keystroke. In this process, the machine is sensitive to specific events in different states.

The FSM is intended to ensure that only a certain number of states are possible. The vehicle can therefore only act according to certain patterns.

Highway driving is a particularly good way of demonstrating how FSM works. Only a few actions are possible in this context. The vehicle can change speed, change lanes or leave the highway. The short-term reactions of the vehicle to changes in the environment are handled by the local planner. The local planner takes over the planning of the path on the basis of the information available for the respective time step. The Behavior Planner is responsible for the medium-term driving strategy of the vehicle. All driving maneuvers can be calculated by reducing the respective costs of a path.

Global Planning
Prof. Dr. Markus Lienkamp

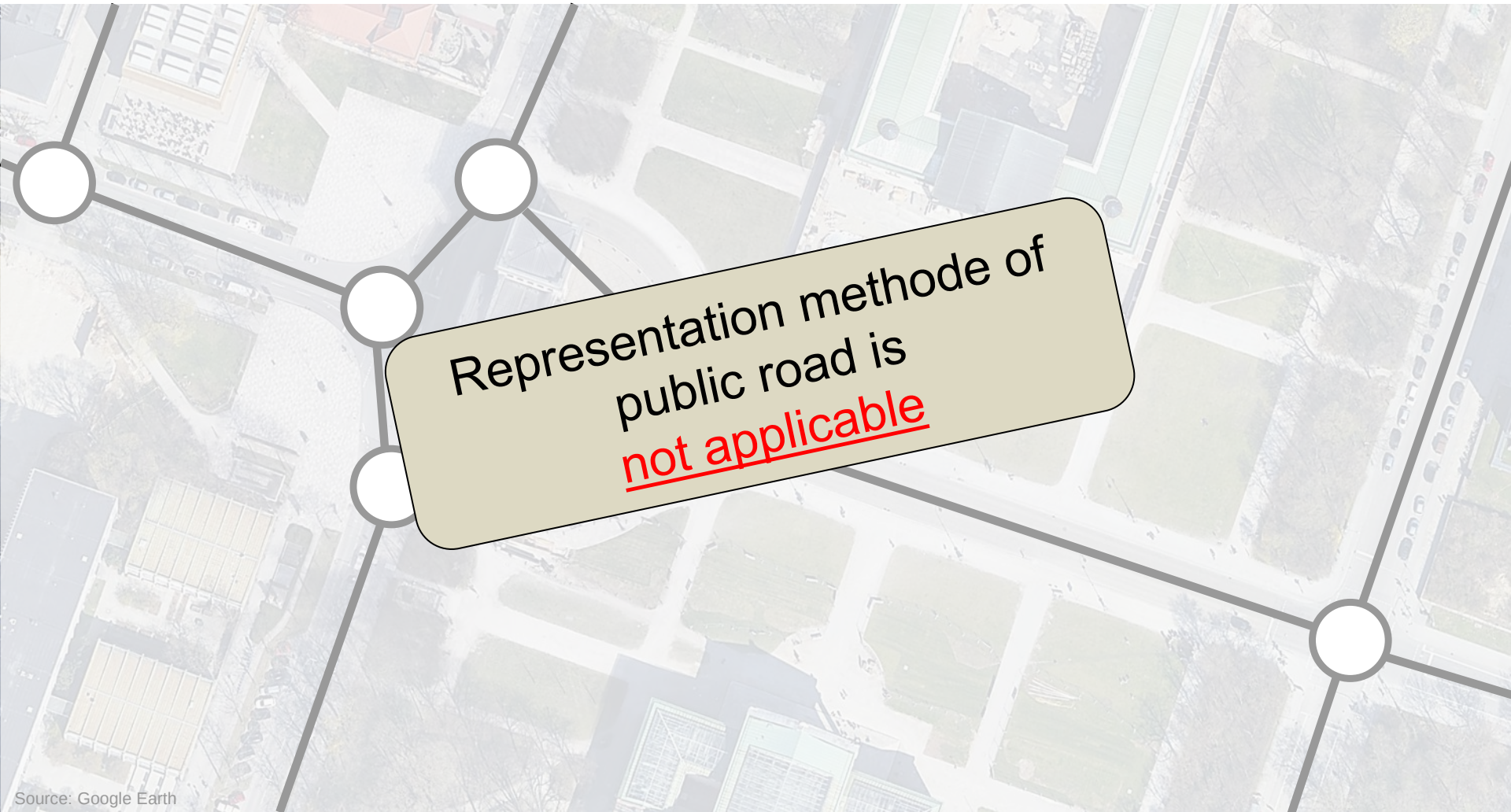
Rainer Trauth, M. Sc.

Agenda

1. Introduction
2. Global Planning:
    1. Dijkstra-Algorithm
    2. A*-Algorithm
3. Behavior Planning
4. **Global Trajectory Planning on the Racetrack**
5. Summary

Global Trajectory Planning on the Racetrack

Digital Representation of a Race Track



Source: Google Earth

Global Trajectory Planning on the Racetrack
Possibilities for Obtaining the Track Representation

1. Using GPS points (e.g., from Open Street Maps) to get centerline, process satellite images and use edge detection to obtain the track widths
2. Recording sensor data while driving along the track with subsequent processing of the data (next slide)

1

2

Sources: Google Maps (left), semester thesis by Andressa de Paula Suiti (right)

Global Trajectory Planning on the Racetrack

Process from LiDAR Data to Centerline

Several steps required to obtain the race track representation from a 2D LiDAR scan:

a) Preprocessing of raw data

b) Smoothing and filtering

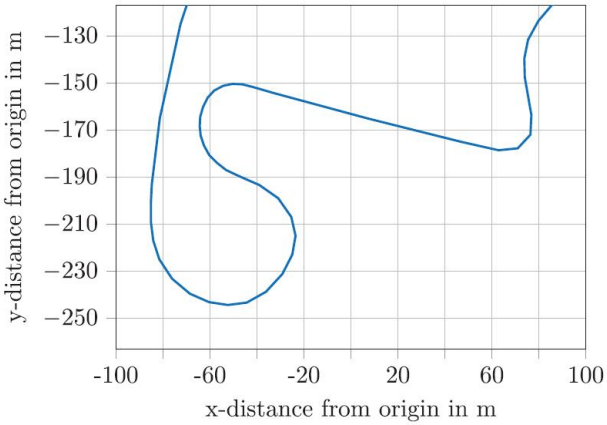
c) Applying Euclidian distance transform

d) Smoothing

(a)

(b)

(c)

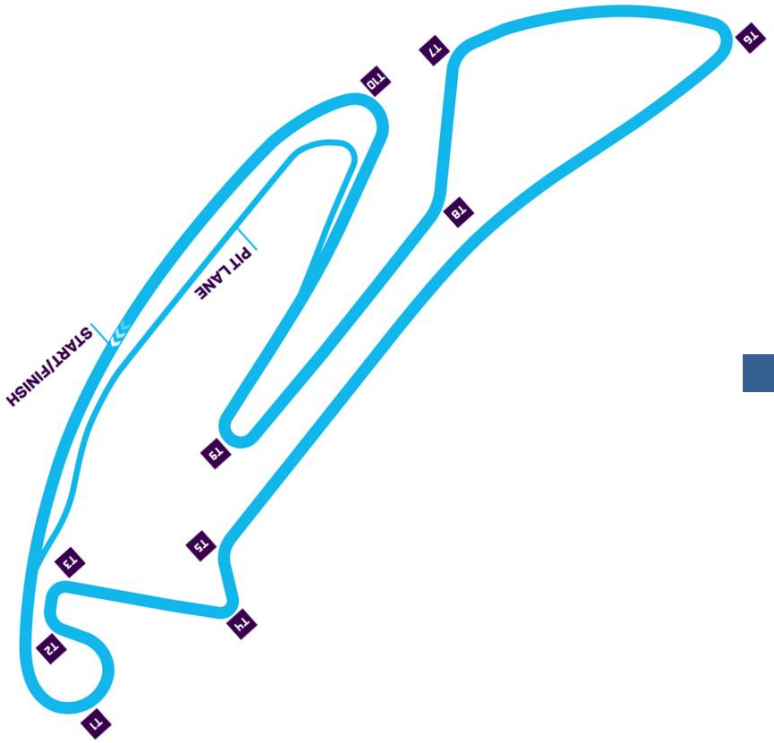


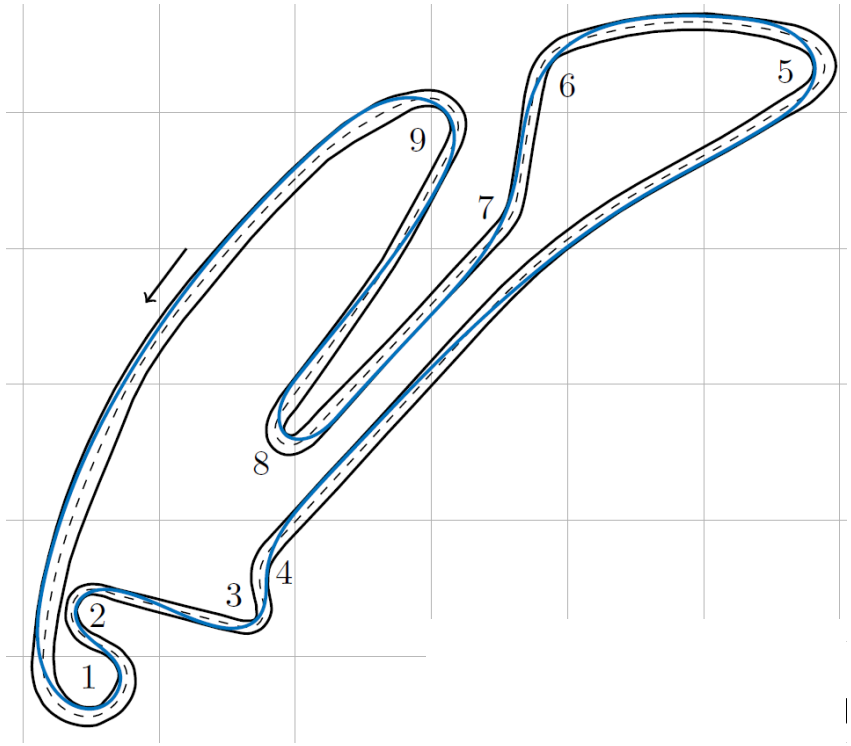
(d)

Source: Minimum curvature trajectory planning and control for an autonomous race car, DOI: 10.1080/00423114.2019.1631455

Global Trajectory Planning on the Racetrack

Defining the Goal for a Global Planner





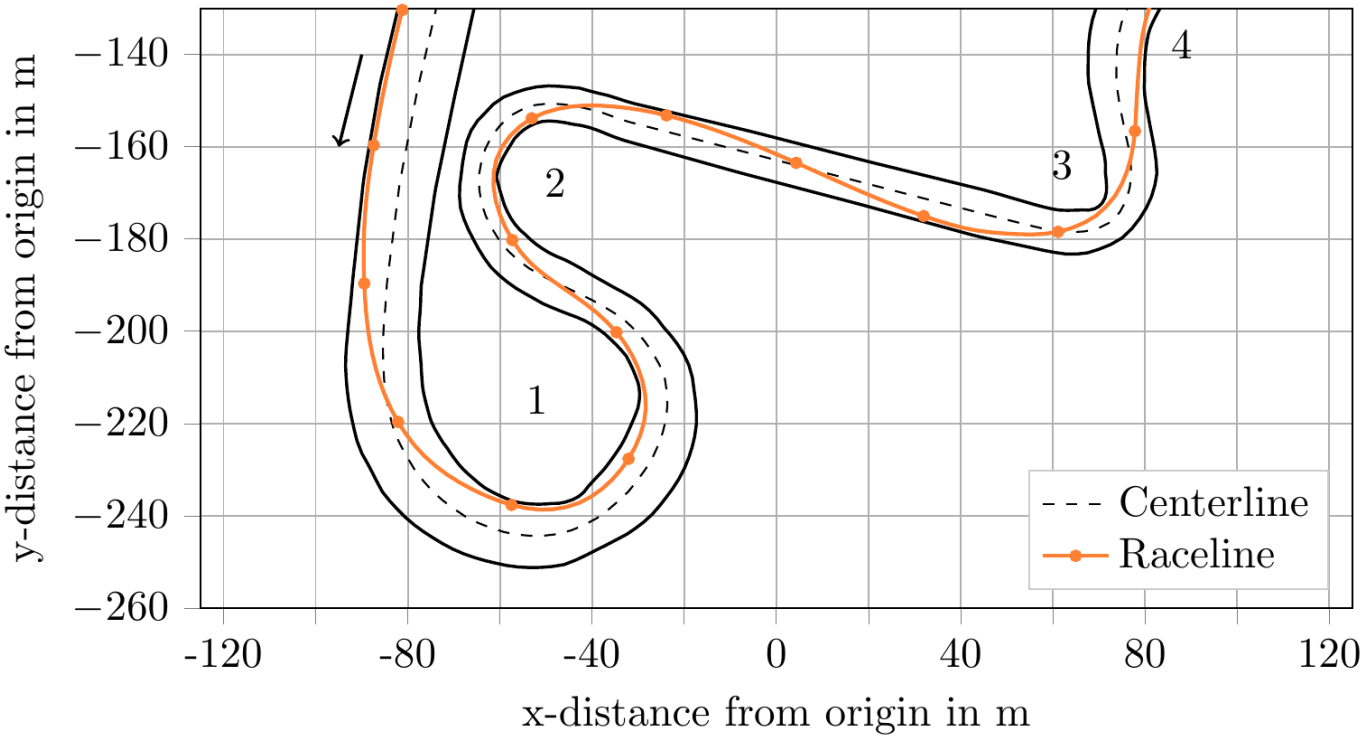
What we have:
Race track representation

What we want:
Racing line on the track

Sources: https://www.fiaformulae.com/en/championship/19-20-race-calendar/2019-2020/berlin-1/berlin-eprix (left)
Minimum curvature trajectory planning and control for an autonomous race car, DOI: 10.1080/00423114.2019.1631455 (right)

Global Trajectory Planning on the Racetrack
Differentiation of Path- and Trajectory Planning

Path planning: $[x, y]$:

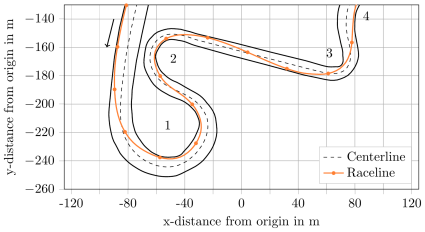


Source: Minimum curvature trajectory planning and control for an autonomous race car, DOI: 10.1080/00423114.2019.1631455

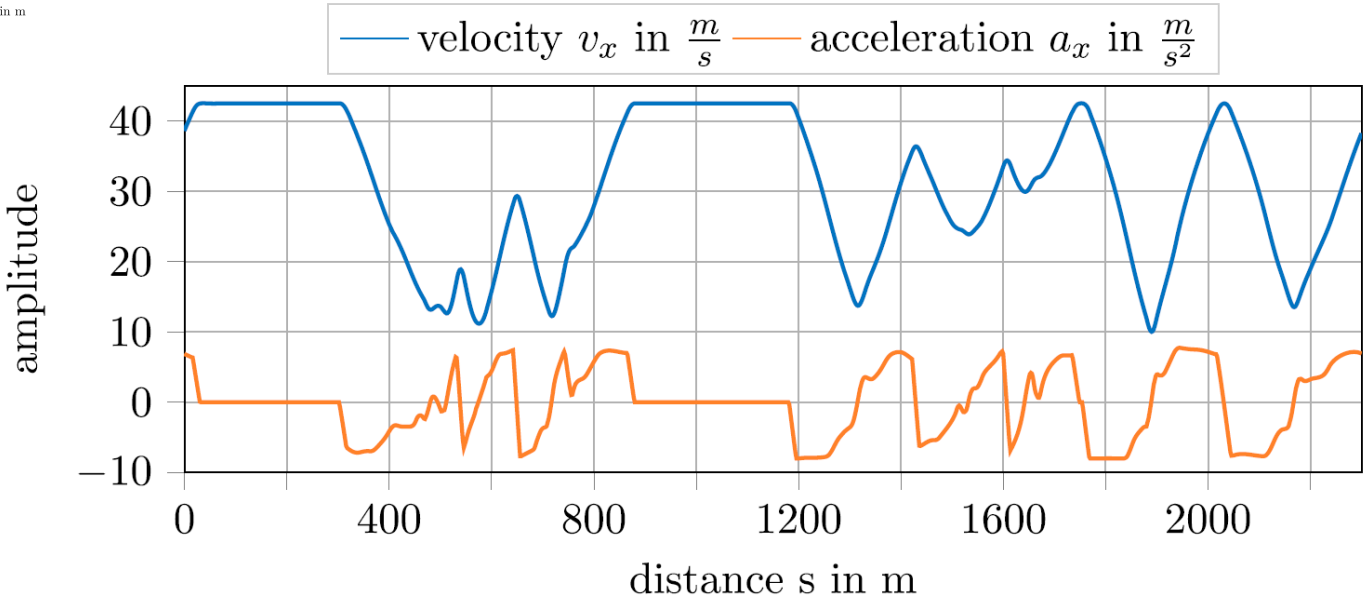
Global Trajectory Planning on the Racetrack

Differentiation of Path- and Trajectory Planning

Trajectory planning: $[x, y, v]$:



+



Source: Minimum curvature trajectory planning and control for an autonomous race car, DOI: 10.1080/00423114.2019.1631455

Global Trajectory Planning on the Racetrack
Differentiation of Path- and Trajectory Planning

Route Planning:
- Decision to take a route from source to destination
- Sequence of discrete geometrical nodes under minimization of a target variable

Path Planning:
- Spatial domain
- Continuou
- Consid geometrical boundary conditions possible (e.g., tr , maximum curvature)

One can't do much with the racing line alone
→ We mostly talk about trajectory planning

Trajectory Planning:
- Spatio-temporal domain
- Further conditions can be checked by temporal information (e.g., freedom from collisions, compliance with acceleration limits)

Global Trajectory Planning on the Racetrack
In Context of Circuit Racing

Tasks of the global trajectory planner:

- Planning of an optimal trajectory → According to any criterion
- Consideration of vehicle dynamics limits

No tasks of the global trajectory planner are:

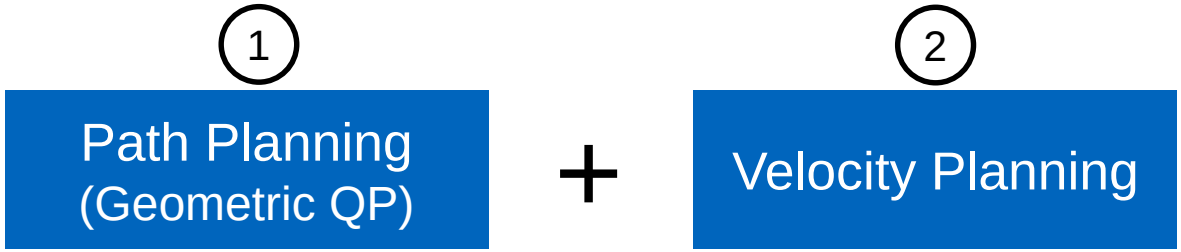
- (High-level) navigation / route planning
- Reaction to the dynamic environment (task of the local trajectory planner)

Global Trajectory Planning on the Racetrack

Several Approaches for Trajectory Planning

Creation of the trajectory by means of optimization:

Solving **two** subsequent sub-problems:



→ Shortest line (Geometric QP) + Velocity Planning (1)

→ Minimum curvature line (Geometric QP) + Velocity Planning (2)

Advantages of Geometric QP problems:

- Few parameters required
- Fast calculation times

QP = Quadratic Programming

Sources:
(1) Braghin et al., Race driver model, DOI: 10.1016/j.compstruc.2007.04.028
(2) Heilmeier et al., Minimum curvature trajectory planning and control for an autonomous race car, DOI: 10.1080/00423114.2019.1631455

Additional Slides

There are also other methods to get an optimal trajectory on a race track. Unlike the slide before, the problem can also be solved in a single optimization problem.

Solving **one** single problem: simultaneous Path + Velocity Profile Planning

→ Minimum time trajectory planning (Optimal Control) (1)

For more information about the method please refer to the attached paper.

Advantages of <u>minimum time optimal control</u> problems:

- Objective equals the goal that is actually pursued on the race track
- Depending on the model, considerably more boundary conditions can be considered, e.g., energy limitations

(1) Christ et al., Time-optimal trajectory planning for a race car considering variable tyre-road friction coefficients, DOI: 10.1080/00423114.2019.1704804

Global Trajectory Planning on the Racetrack

Quadratic Programming

- Quadratic programming is a process to solve mathematical optimization problems involving quadratic functions
- The method minimizes or maximizes a target value
- Certain boundary conditions must be maintained during optimization

$$\min f(x) = \frac{1}{2} x^T Q x + c^T x$$

$$subject\ to: \quad Ax \leq b$$

$$x \geq 0,\ c, x \in \mathbb{R}^n, A_{m \times n}, Q_{m \times n}$$

Global Trajectory Planning on the Racetrack

Digital Representation of a Race Track

- Representation by centerline points and according track widths
- Discretization step size dependent on use case, e.g. $5m$

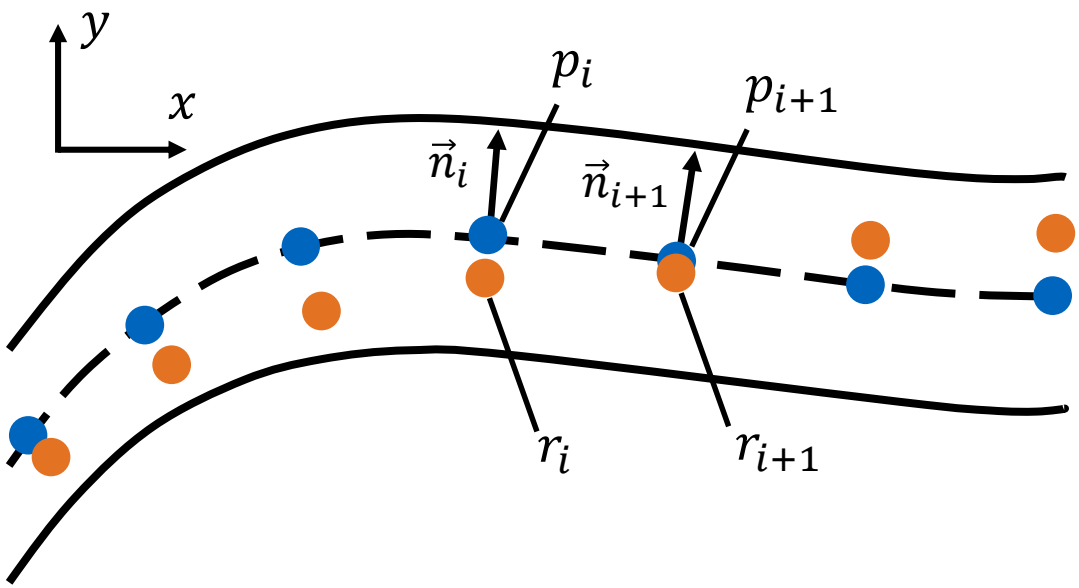
Source: Google Maps

Global Trajectory Planning on the Racetrack

Idea of Geometrical Optimization Problems

Shifting each point along its normal vector within the track widths (variable α in our case)

$$\vec{r}_i = \vec{p}_i + \alpha_i \vec{n}_i$$



Minimization of a given objective function:

- E.g.: Minimizing the sum of the curvature values for all discretization points
- Consideration of the vehicle width (+ safety distance)

$$\alpha_i \in \left[-w_{\mathrm{tr,left},i} + \frac{w_{\mathrm{veh}}}{2}, w_{\mathrm{tr,right},i} - \frac{w_{\mathrm{veh}}}{2} \right]$$

i: discretization step
w_{veh}: vehicle width
w_{tr}: track width

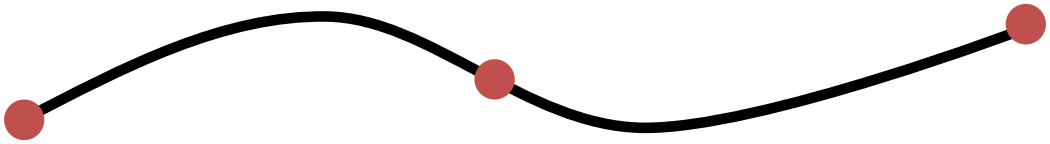
Source: Minimum curvature trajectory planning and control for an autonomous race car, DOI: 10.1080/00423114.2019.1631455

Global Trajectory Planning on the Racetrack
(Curvature Continuous Cubic) Splines

- Many approaches in the field of path planning are based on splines
- Third order polynomials → continuous description
- Possible to assure heading and curvature continuity
- Analytical calculation of the exact derivatives for…
  - tangent vectors,
  - normal vectors,
  - heading,
  - curvature

  …possible.

→ This is the prerequisite for our approaches.



Global Trajectory Planning on the Racetrack
(Curvature Continuous Cubic) Splines

- Mostly, we rely on separated polynomials to represent x- and y-coordinates of the i-th segment / spline (in the further process only the x-component is shown):

$$x_i(t) = a_i + b_i t + c_i t^2 + d_i t^3$$

$$x_i'(t) = b_i + 2c_i t + 3d_i t^2$$

$$x_i''(t) = 2c_i + 6d_i t$$

- The equations set up in dependence of the normalized curvilinear parameter t, which is defined as follows (s is the curvilinear distance along the centerline):

$$t_i(s) = \frac{s - s_{i0}}{\Delta s_i}$$

Additional Slides

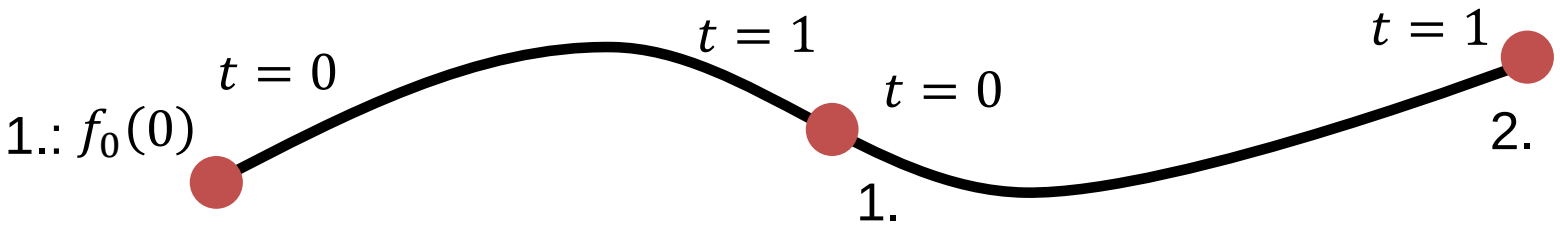
t is the normalized curvilinear parameter along one spline segment starting at the distance s_{i0}. Therefore, $0 \leq t_i \leq 1$ holds. The spline interpolation requires that consecutive splines share their respective beginning and endpoint as well as their first and second derivative at these points. The latter fact ensures smooth curvature along the interpolation. Complying with these constraints, the spline parameters a_i, b_i, c_i and d_i are obtained from the solution of a linear equation system of the form $Az = b$. The matrix A and the vector b formalize the above constraints.

Global Trajectory Planning on the Racetrack

Determination of the Spline Parameters a, b, c, d

We have several boundary conditions:

1. Start of spline should be placed on current discretization point
2. End of spline should be placed on subsequent discretization point
3. Heading continuity, i.e. heading at the end of the spline should be equal to the heading at the beginning of the next spline
4. Curvature continuity, i.e. curvature at the end of the spline should be equal to the curvature at the beginning of the next spline



2.: $f_0(1) = f_1(0)$

3.: $\dot{f}_0(1) = \dot{f}_1(0)$

4.: $\ddot{f}_0(1) = \ddot{f}_1(0)$

Global Trajectory Planning on the Racetrack
Setting Up the Linear Equation System (LES)

- Converting the boundary conditions into equations:

1. $t = 0$: $a_i = x_i$
2. $t = 1$: $a_i + b_i + c_i + d_i = x_{i+1}$
3. $t = 1$ and $t = 0$: $b_i + 2 \cdot c_i + 3 \cdot d_i - b_{i+1} = 0$
4. $t = 1$ and $t = 0$: $2 \cdot c_i + 6 \cdot d_i - 2 \cdot c_{i+1} = 0$

- The equations can be split up into a coefficient matrix M and the parameter matrix A (b contains the right hand side):

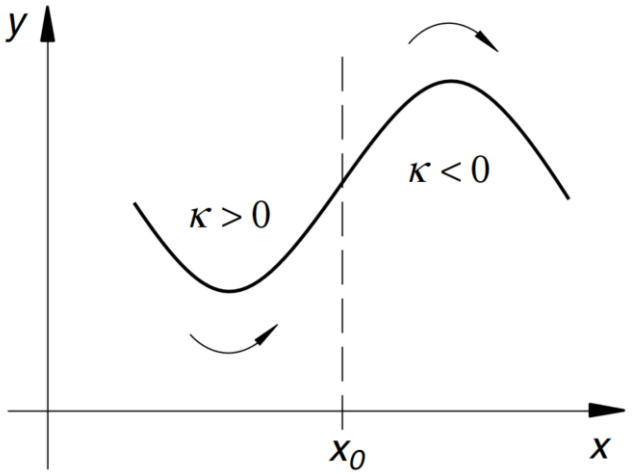
$$M \cdot A = b$$

Global Trajectory Planning on the Racetrack
Usage of the Determined Parameter Matrix A

- Having the spline parameters available, we can determine heading and curvature in the 2-D space as follows:
  - Heading: $\psi = \text{atan2}\left(\frac{y'}{x'}\right)$
  - Curvature: $\kappa = \frac{x'y'' - y'x''}{\left(x'^2 + y'^2\right)^{\frac{3}{2}}}$

$\kappa > 0 \leftrightarrow Left\ curvature$

$\kappa < 0 \leftrightarrow Right\ curvature$



Papula L. Mathematische formelsammlung: für ingenieure und naturwissenschaftler. Braunschweig/Wiesbaden: Springer FachmedienWiesbaden; 2017.

GitHub: https://github.com/TUMFTM/trajectory_planning_helpers/blob/master/trajectory_planning_helpers/calc_head_curv_an.py

Global Trajectory Planning on the Racetrack
Shortest Path QP

- Optimization problem:

$$\text{minimize } [\alpha_i \dots \alpha_\text{N}] \sum_{i=1}^{N} d_i^2(t)$$

$$\text{subject to} \qquad \alpha_i \in \left[\alpha_{i,\min}, \alpha_{i,\max}\right] \quad \forall\, 1 \leq i \leq N$$

- Evaluation solely at the discretization points: $t = 0$

- Calculation of the squared distances:

normal vectors

$$\Delta P_{x,i} = x_{i+1} - x_i + \alpha_{i+1}\Delta x_{i+1} - \alpha_i \Delta x_i$$

$$d_i^2 = \Delta P_{x,i}^T \Delta P_{x,i} + \Delta \mathrm{P}_{y,i}^\mathrm{T} \Delta P_{y,i}$$

- Several matrix reshaping operations lead to an implementable formulation of the problem (see code on GitHub)

Source: Race driver model, DOI: 10.1016/j.compstruc.2007.04.028

Global Trajectory Planning on the Racetrack
Minimum Curvature QP

- Optimization problem:

$$\text{minimize } [\alpha_i \dots \alpha_N] \sum_{i=1}^{N} \kappa_i^2(t)$$

$$\text{subject to} \quad \alpha_i \in [\alpha_{i,\min}, \alpha_{i,\max}] \quad \forall\, 1 \leq i \leq N$$

- Evaluation solely at the discretization points: $t = 0$

- Calculation of the squared curvature values

$$\kappa_i = \frac{x_i' y_i'' - y_i' x_i''}{\left(x_i'^2 + y_i'^2\right)^{\frac{3}{2}}} \quad \Rightarrow \quad \kappa_i^2 = \frac{x_i'^2 y_i''^2 - 2 x_i' x_i'' y_i' y_i'' + y_i'^2 x_i''^2}{\left(x_i'^2 + y_i'^2\right)^3}$$

$$a_y = v^2 \cdot \kappa$$

- Several matrix reshaping operations lead to an implementable formulation of the problem (see code on GitHub)

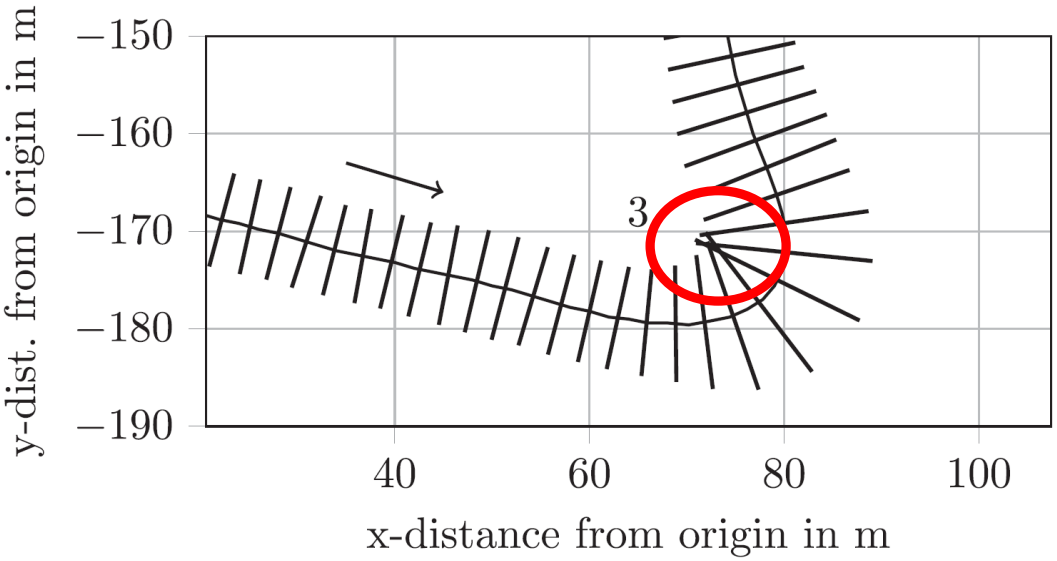
Source: Minimum curvature trajectory planning and control for an autonomous race car, DOI: 10.1080/00423114.2019.1631455
Papula L. Mathematische Formelsammlung: für Ingenieure und Naturwissenschaftler: Springer Fachmedien Wiesbaden; 2017.

Additional Slides

The L2 norm is chosen because of its desirable properties in terms of numerical optimization and elegant formulation of the resulting optimization problem. From the spline representation it follows that it is most convenient and efficient to evaluate the splines at $t = 0$.

Global Trajectory Planning on the Racetrack
Problems when Using Normal Vectors in the Approach

- Possible intersections of the normal vectors:



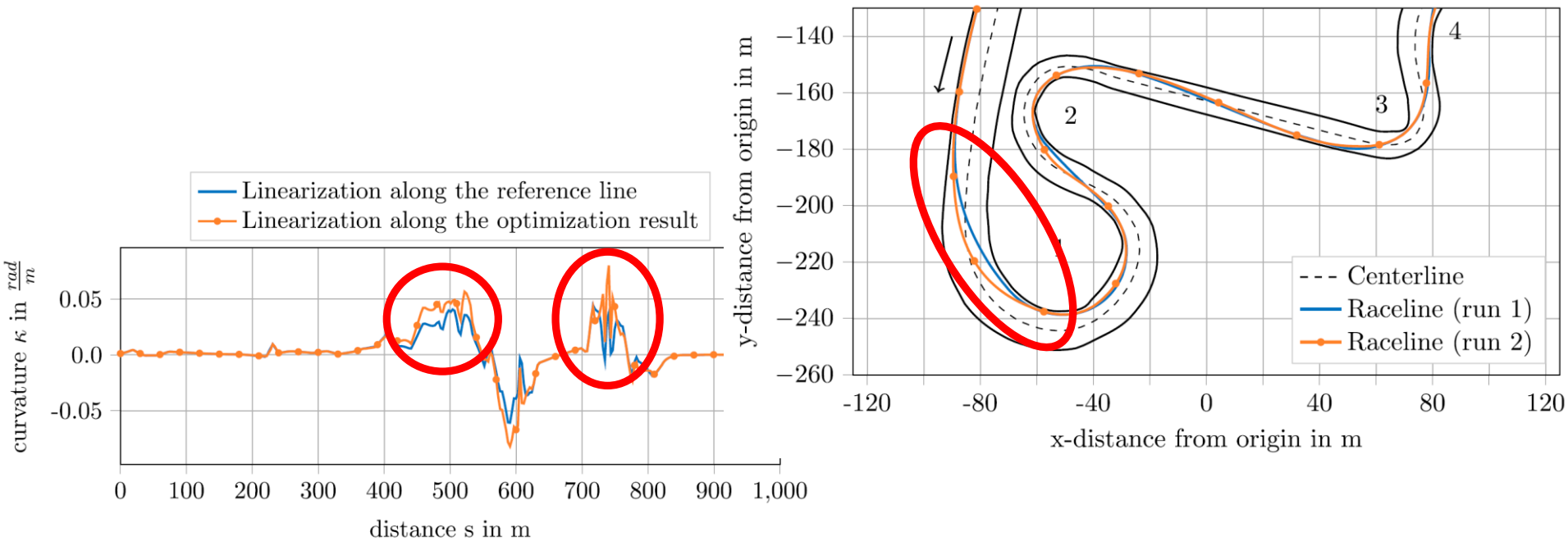
- Can be solved by smoothing the centerline (draws the line towards the center of the corner → reference line instead of centerline) or by increasing the discretization step size

Source: Minimum curvature trajectory planning and control for an autonomous race car, DOI: 10.1080/00423114.2019.1631455

Global Trajectory Planning on the Racetrack
Problems of the Minimum Curvature Approach

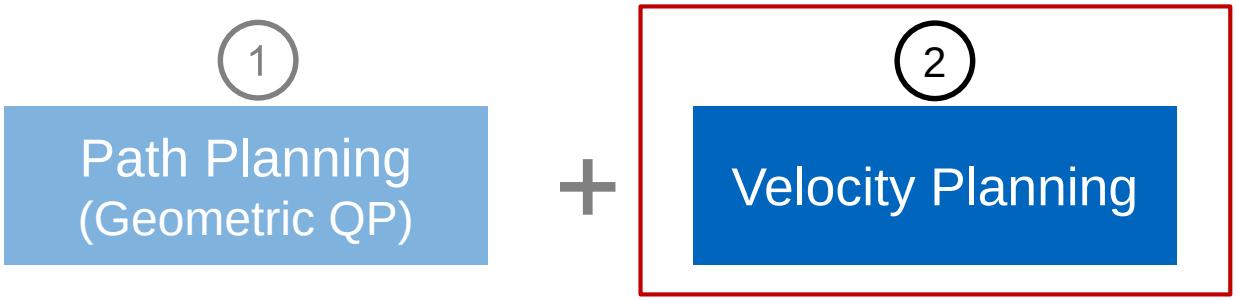
- Linearization errors lead to suboptimal results (and non-compliance with the boundary conditions)

- Can be solved by an iterative invocation of the optimization problem



Source: Minimum curvature trajectory planning and control for an autonomous race car, DOI: 10.1080/00423114.2019.1631455

Global Trajectory Planning on the Racetrack
Planning of a Velocity Profile



Typical solver types for planning a velocity profile on the race track

- **Quasi-steady state solvers** → Forward-Backward (next slides)

→ Subsequent points are solved independently of each other

→ Advantages: fast, less complicated

Sources:
(1) See lecture „Rennsporttechnik“ for the working principle
(2) Heilmeier et al., A Quasi-Steady-State Lap Time Simulation for Electrified Race Cars, DOI: 10.1109/EVER.2019.8813646

Global Trajectory Planning on the Racetrack
Planning of a Velocity Profile

Assumption: The race car is constantly driven at the vehicle dynamics limits

- Longitudinal acceleration limits result from engine limits (positive) and tires/brakes (negative)
- Lateral acceleration limits result from maximum transmittable tire forces

Important equations:

→ Acting lateral acceleration (dependent on velocity v and curvature κ)

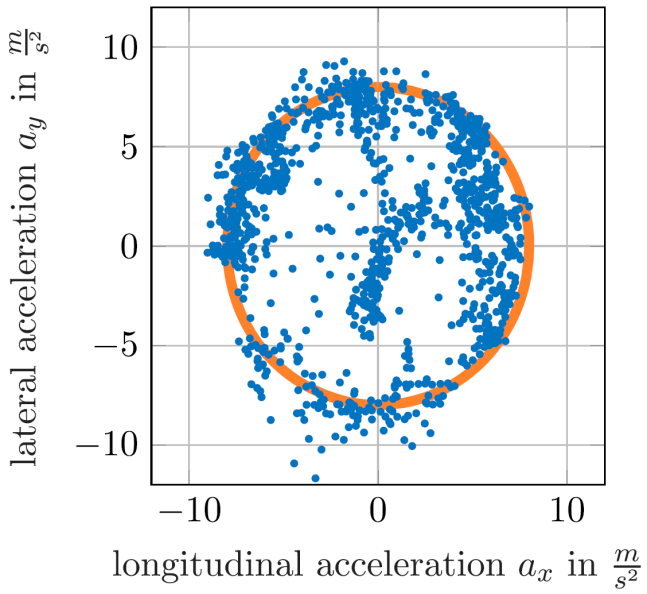
$$a_y = \frac{v^2}{R} = v^2 \cdot \kappa$$

→ Possible longitudinal acceleration (dependent on drivetrain force F_x)

$$a_x = \frac{F_x}{m}$$

Global Trajectory Planning on the Racetrack
The g-g Diagram

- Simple approaches are usually based on a g-g diagram, which connects the possible longitudinal and lateral acceleration (simplified vehicle dynamics)



- A g-g-v diagram is an extension that takes the velocity into account as a third dimension (relevant if we want to consider downforce and drag)

Source: Minimum curvature trajectory planning and control for an autonomous race car, DOI: 10.1080/00423114.2019.1631455

Additional Slides

Typical solver types for planning a velocity profile on the race track are Quasi-steady state solvers. The subsequent points are solved independently of each other. Possible implementations are the Forward-Backward implementation, the Pure Forward (1) implementation and a Mixture (2) of those implementations. Another method to solve the problem are transient solvers. These solvers consider time-dependent effects such as delayed tire forces or actor constraints (1). Transient solvers are a bit more accurate than Quasi-steady state solvers, but less time-efficient.

Sources:
(1) See lecture „Rennsporttechnik“ for the working principle
(2) Heilmeier et al., A Quasi-Steady-State Lap Time Simulation for Electrified Race Cars, DOI: 10.1109/EVER.2019.8813646

Global Trajectory Planning on the Racetrack

The Working Principle of a Forward-Backward Solver (I)

1. Get first velocity profile estimate based on the maximum possible lateral acceleration for the curvature at every discretization point

2. Cut the profile where the velocity is above the maximum velocity

3. Loop through the discretization points and apply possible positive longitudinal acceleration → Forward part

4. Loop through the discretization points and apply possible negative longitudinal acceleration → Backward part

Global Trajectory Planning on the Racetrack

The Working Principle of a Forward-Backward Solver (II)

- In steps 3. and 4. we derive the velocity in the next discretization point as follows (simplified version):
  1. Calculate acting lateral acceleration:

$$a_{y,i} = v_i^2 \cdot \kappa_i$$

  2. Determine remaining potential for longitudinal acceleration:

$$a_{x,i} = a_{x,max} \sqrt{1 - \left(\frac{a_{y,i}}{a_{y,max}} \right)^2}$$

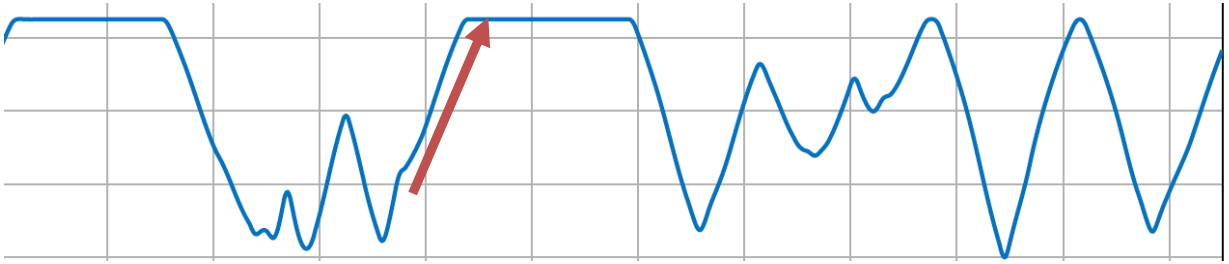
  3. Calculate velocity in the next discretization point assuming a constant longitudinal acceleration over element length l_i:

$$v_{i+1} = \sqrt{v_i^2 + 2 \cdot a_{x,i} \cdot l_i}$$

Global Trajectory Planning on the Racetrack

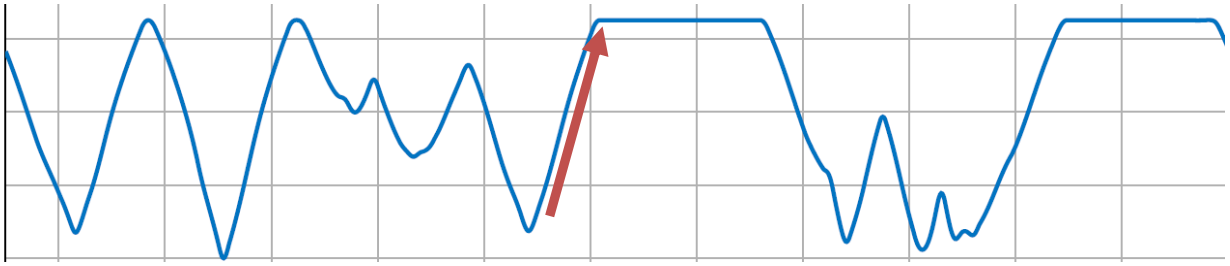
The Working Principle of a Forward-Backward Solver (III)

- Inversion of the velocity profile (and track) allows us to consider the negative longitudinal acceleration (during braking) as if it was a positive acceleration
- Normal profile and track for the forward calculation:



Acceleration

- Inversed profile and track for the backward calculation:



Deceleration (in reality)

Global Trajectory Planning on the Racetrack

Ball Park Figures for the Calculation times

	Shortest Path Optimization	Minimum Curvature Optimization	Minimum Time Optimization
Calculation time in seconds	4 s	18 s	151 s

Conditions:

- Berlin Formula E track
- Discretization step size $3m$
- Intel Core i7 6820HQ

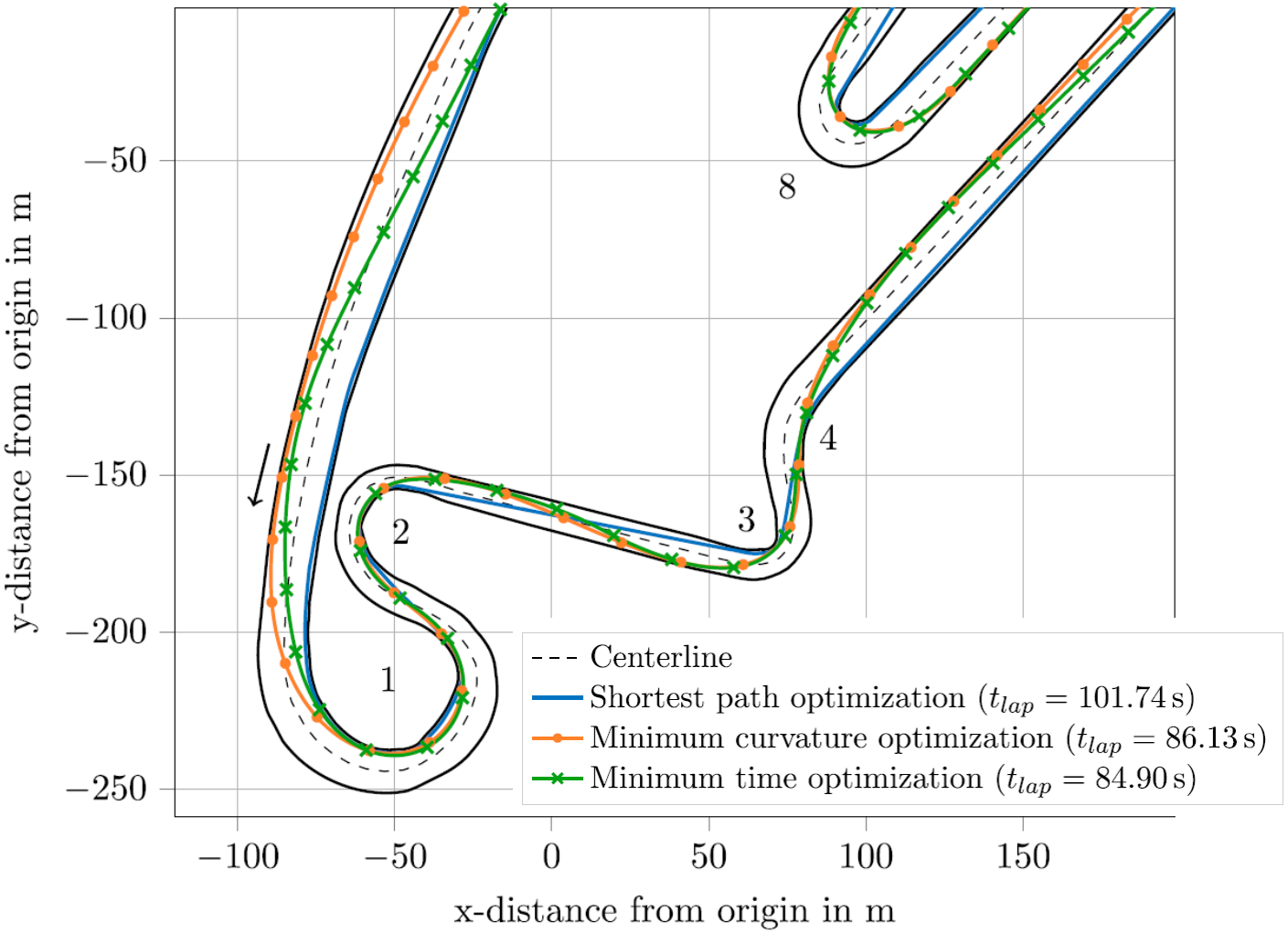
General remark:

- Computing time is not as critical in global planning as it is in local planning, but still not negligible (e.g., for an adjustment during the race)

Source: Minimum curvature trajectory planning and control for an autonomous race car, DOI: 10.1080/00423114.2019.1631455

Global Trajectory Planning on the Racetrack

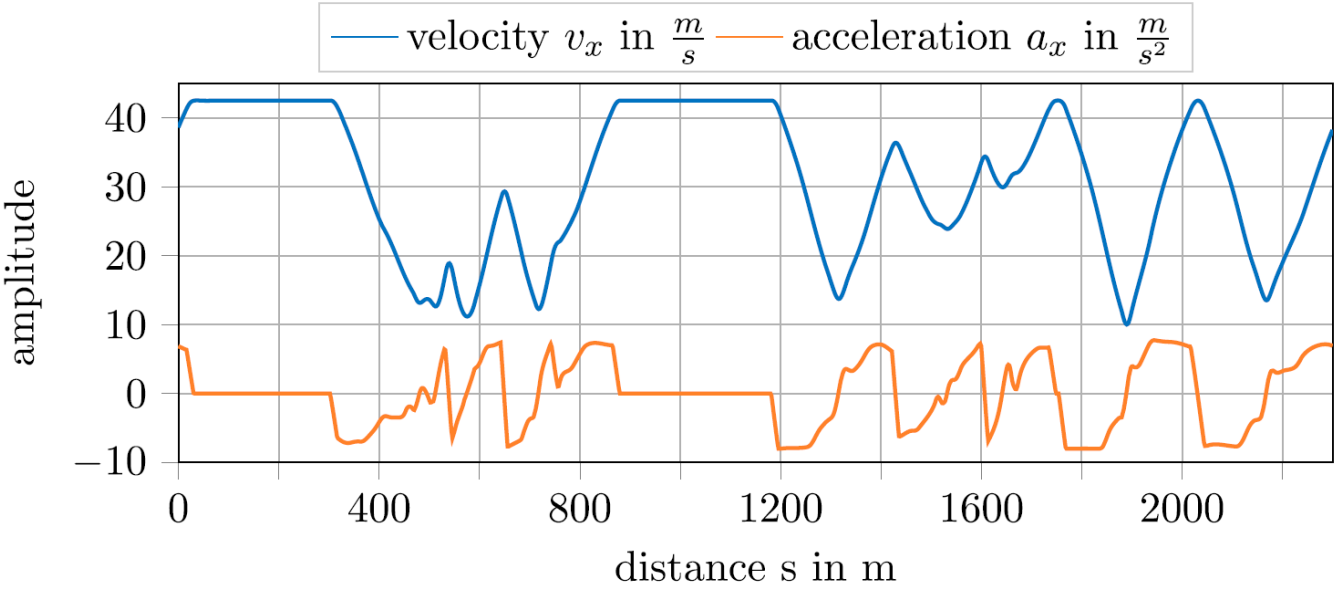
Comparison of the Results for the Three Objectives



Source: Minimum curvature trajectory planning and control for an autonomous race car, DOI: 10.1080/00423114.2019.1631455

Global Trajectory Planning on the Racetrack
The Velocity Profile Berlin Map

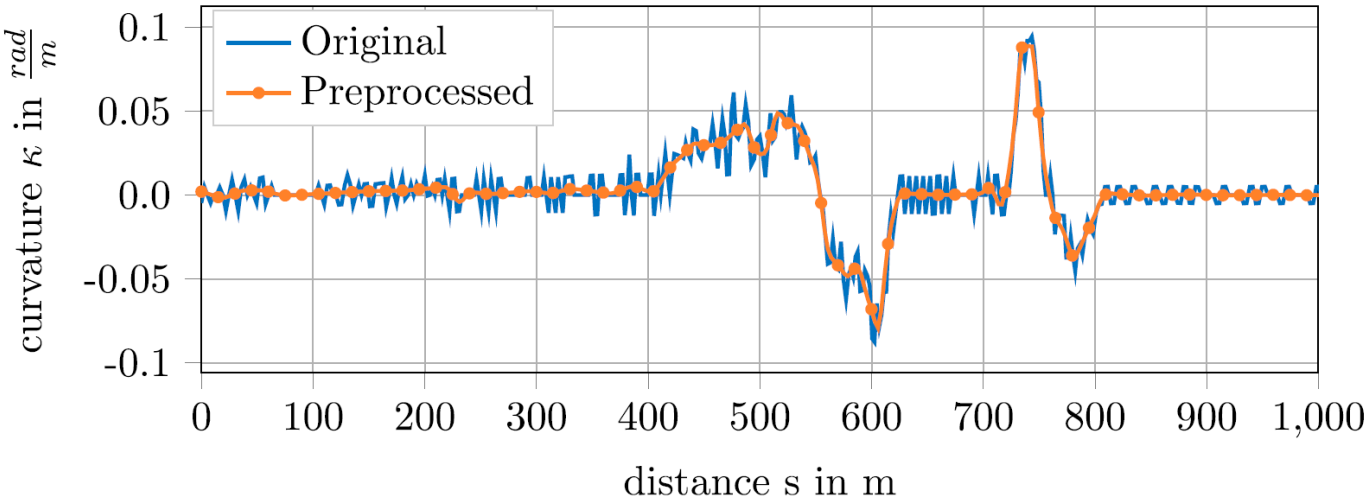
→ Maximum velocity was 150 kph (approx. 42m/s)



Source: Minimum curvature trajectory planning and control for an autonomous race car, DOI: 10.1080/00423114.2019.1631455

Global Trajectory Planning on the Racetrack
An Important Aspect for Planning a Velocity Profile

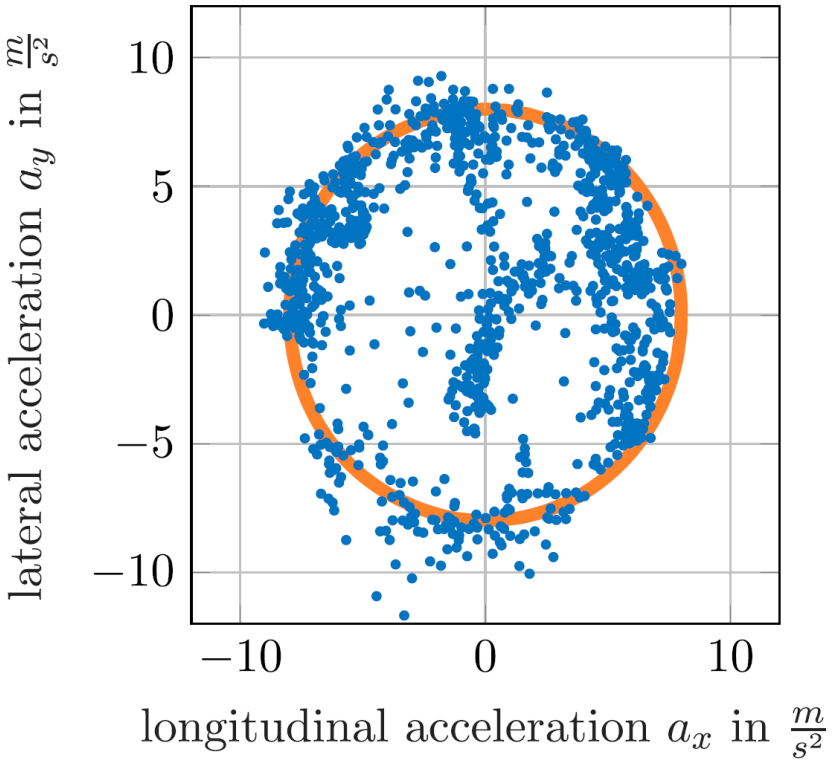
→ Smoothness of the curvature profile (used as an input) is extremely important to obtain a valid velocity profile → often it must be filtered before using it



Source: Minimum curvature trajectory planning and control for an autonomous race car, DOI: 10.1080/00423114.2019.1631455

Global Trajectory Planning on the Racetrack
Realization in Practice (II)

→ Comparison between planned (orange) and measured (blue) accelerations



Source: Minimum curvature trajectory planning and control for an autonomous race car, DOI: 10.1080/00423114.2019.1631455

Global Trajectory Planning on the Racetrack
Software Available on GitHub

- Global trajectory planner: https://github.com/TUMFTM/global_racetrajectory_optimization
- Trajectory planning helper functions: https://github.com/TUMFTM/trajectory_planning_helpers
- Lap time simulation: https://github.com/TUMFTM/laptime-simulation
- Race track database: https://github.com/TUMFTM/racetrack-database

Global Planning
Prof. Dr. Markus Lienkamp

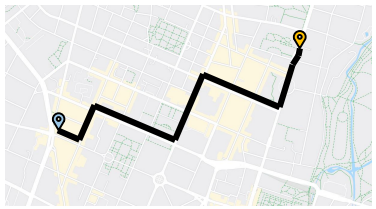
Rainer Trauth, M. Sc.

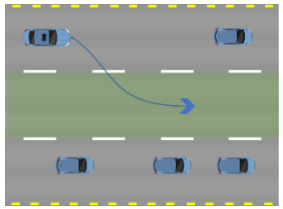
Agenda

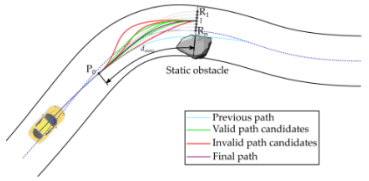
1. Introduction
2. Global Planning:
    1. Dijkstra-Algorithm
    2. A*-Algorithm
3. Behavior Planning
4. Global Trajectory Planning on the Racetrack
5. **Summary**

Summary
What did we learn today

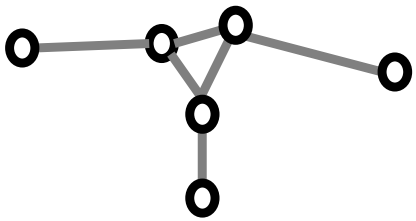
- Distinction between:
  - Global Planning
  - Behavior Planning
  - Local Planning

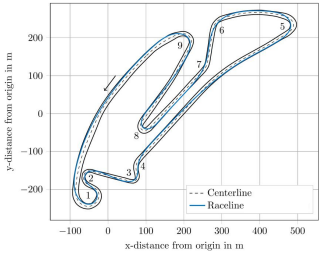


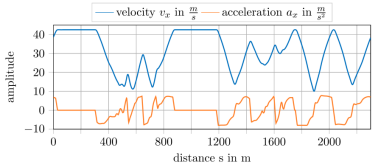




- Distinction between:
  - Route Planning
  - Path Planning
  - Trajectory Planning



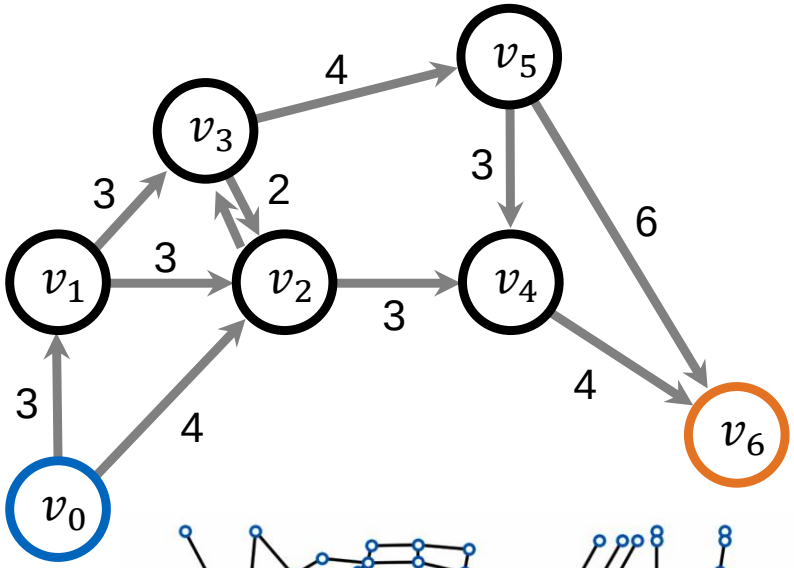


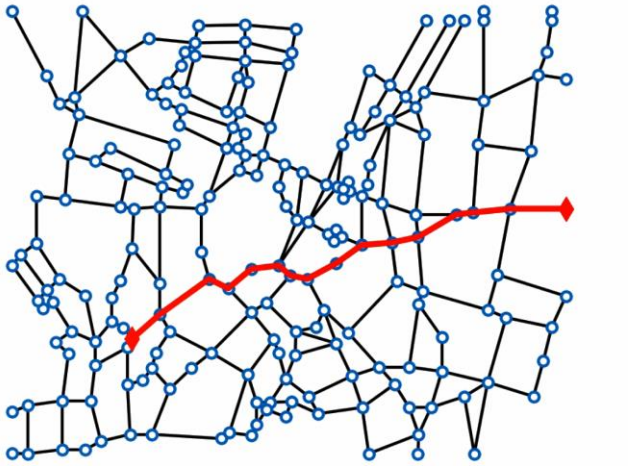


- Goals of Route Planning

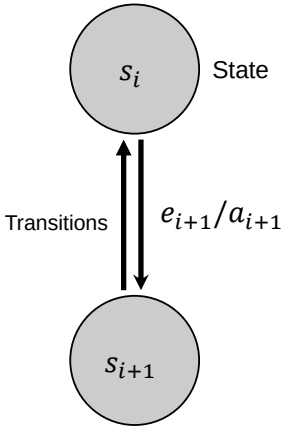
Summary
What did we learn today

- Formal representation of Graphs
- How Dijkstra's Algorithm works
- How A*-Algorithm works
- Differences between the algorithms



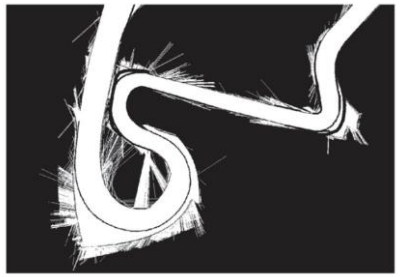


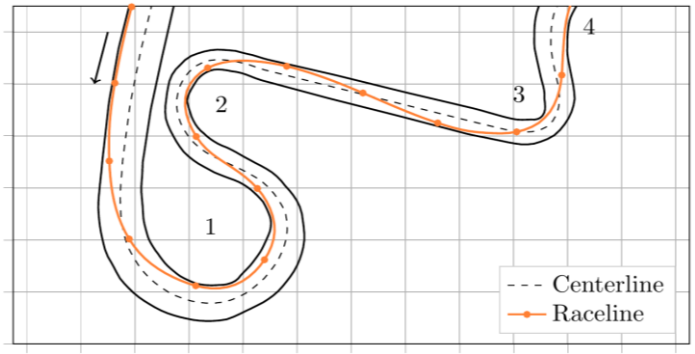
Summary
What did we learn today



- Behavior Planner:
  - Can fill the technical gap between Local + Global Planner
  - Can be represented with Finite-state Machines

- Race track trajectory:
  - Mathematical description of race track
  - Global Trajectory Planning on a Race track
  - Calculate Splines and Velocity Profile
  - Differences of possible algorithms





Summary
What did we learn today

- The goal of Global Route Planning is to find a **feasible**, **effective** and **efficient** way from A to B
- The presented Global Planning Algorithm works with **complete information**
- Unlike the Local Planner, no **real-time condition** is necessary for the Global Planner
- **Graph-based** methods can be applied to find a suitable path
- **Dijkstra** and **A*** always find an **optimal** path if available
- **A*** uses a **heuristic estimation** to reduce the number of iteration steps
- **Finite-State-Machines** can be used to model the **behavior** of the vehicle
- The **methods** to **calculate** the global raceline on the race track are **different** the methods on public roads

Summary
What did we learn today

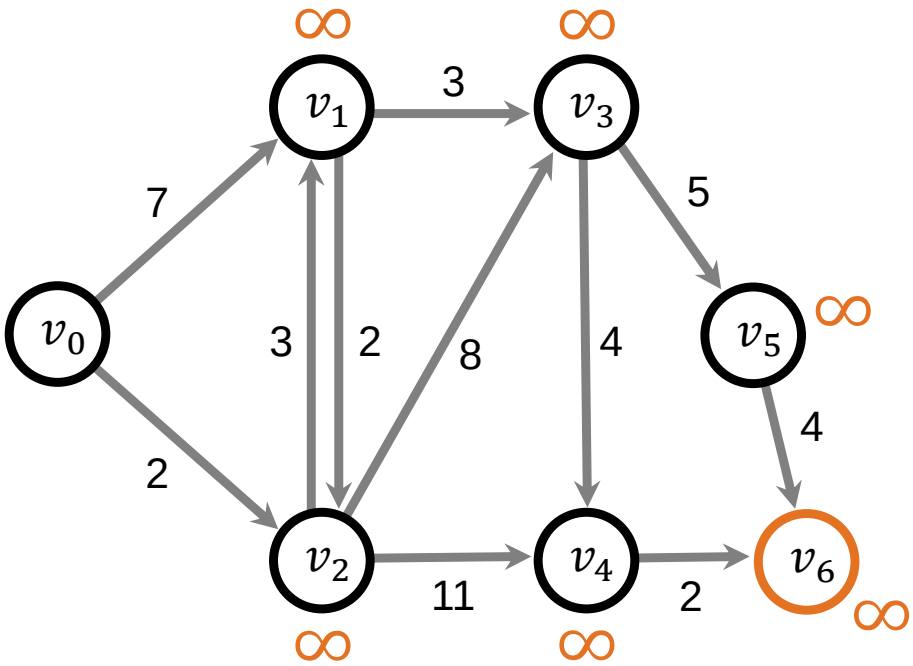
- **Trajectory** Planning includes the **spatio-temporal** domain
- There are **several** approaches to calculate the **optimal** race path
- The **computation** time is a **decisive factor** when choosing the method
- **Smoothing** the center line can **solve possible problems** within the normal vector approach
- **Quasi-steady state solvers** are a **fast** method for planning the velocity profile
- The **g-g-diagram** can be used to **visualize** the accelerations in lateral and longitudinal direction

Summary
References

- https://www.ri.cmu.edu/pub_files/2014/6/IV2014-Junqing-Final.pdf
- https://www.coursera.org/lecture/motion-planning-self-driving-cars/lesson-1-behaviour-planning-tPdVH
- https://www.researchgate.net/publication/321065254_A_Finite_State_Machine_Based_Automated_Driving_Controller_and_its_Stochastic_Optimization
- https://www.researchgate.net/publication/273264449_Understanding_Dijkstra_Algorithm
- https://www-m9.ma.tum.de/graph-algorithms/spp-dijkstra/index_de.html
- http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html
- https://www-m9.ma.tum.de/graph-algorithms/spp-a-star/index_de.html

Additional slides for better understanding

Global Planning
Dijkstra‘s Algorithm



d=distance
p=predecessor
c=considered

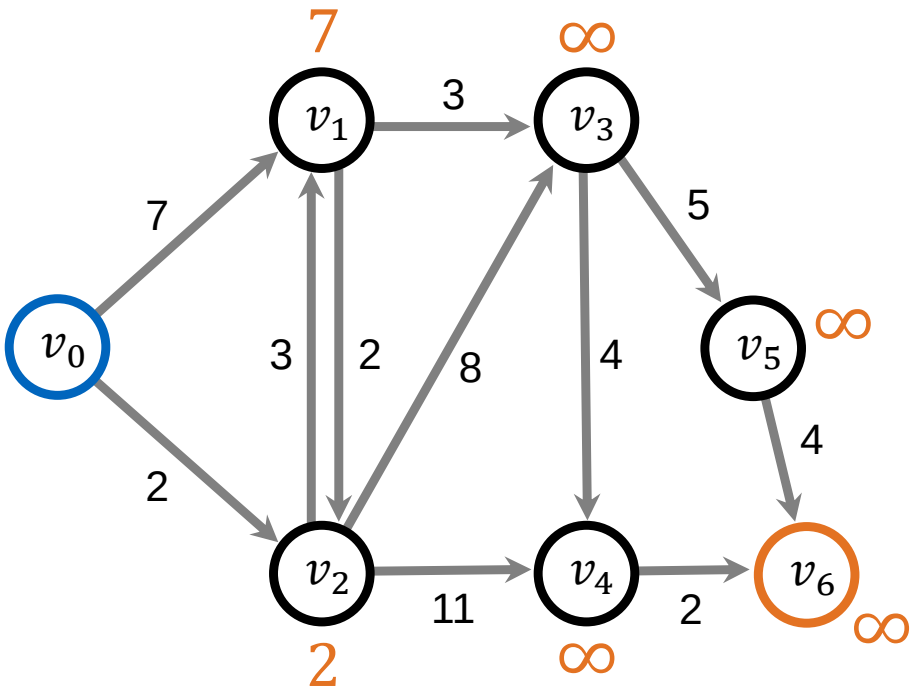
Step 0:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
d[v]	0	∞	∞	∞	∞	∞	∞
p[v]	-	-	-	-	-	-	-
c[v]	-	-	-	-	-	-	-

Priority Queue:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
d[v]	0	∞	∞	∞	∞	∞	∞

Global Planning
Dijkstra‘s Algorithm



d=distance
p=predecessor
c=considered

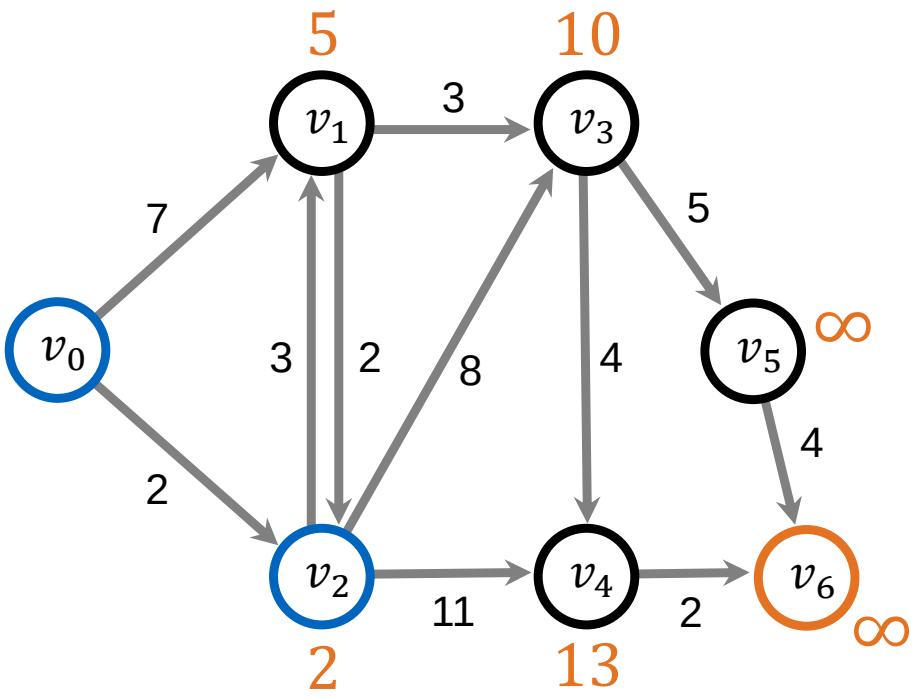
Step 1:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
d[v]	0	7	2	∞	∞	∞	∞
p[v]	-	v_0	v_0	-	-	-	-
c[v]	✓	-	-	-	-	-	-

Priority Queue:

v	v_1	v_2	v_3	v_4	v_5	v_6
d[v]	7	2	∞	∞	∞	∞

Global Planning
Dijkstra‘s Algorithm



d=distance
p=predecessor
c=considered

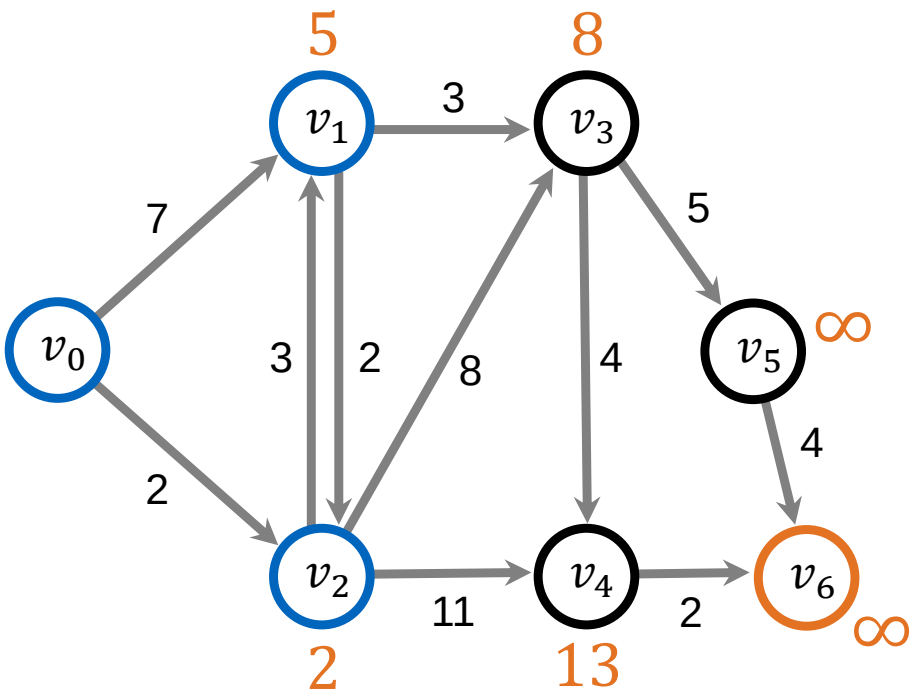
Step 2:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
d[v]	0	5	2	10	13	∞	∞
p[v]	-	v_2	v_0	v_2	v_2	-	-
c[v]	✓	-	✓	-	-	-	-

Priority Queue:

v	v_1	v_3	v_4	v_5	v_6
d[v]	5	10	13	∞	∞

Global Planning
Dijkstra‘s Algorithm



d=distance
p=predecessor
c=considered

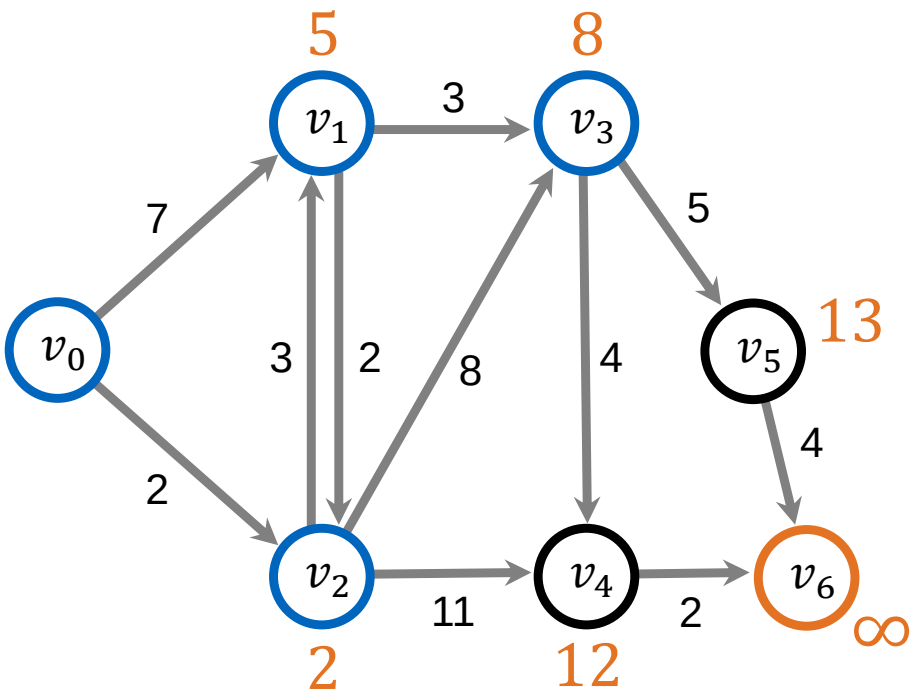
Step 3:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
d[v]	0	5	2	8	13	∞	∞
p[v]	-	v_2	v_0	v_1	v_2	-	-
c[v]	✓	✓	✓	-	-	-	-

Priority Queue:

v	v_3	v_4	v_5	v_6
d[v]	8	13	∞	∞

Global Planning
Dijkstra‘s Algorithm



d=distance
p=predecessor
c=considered

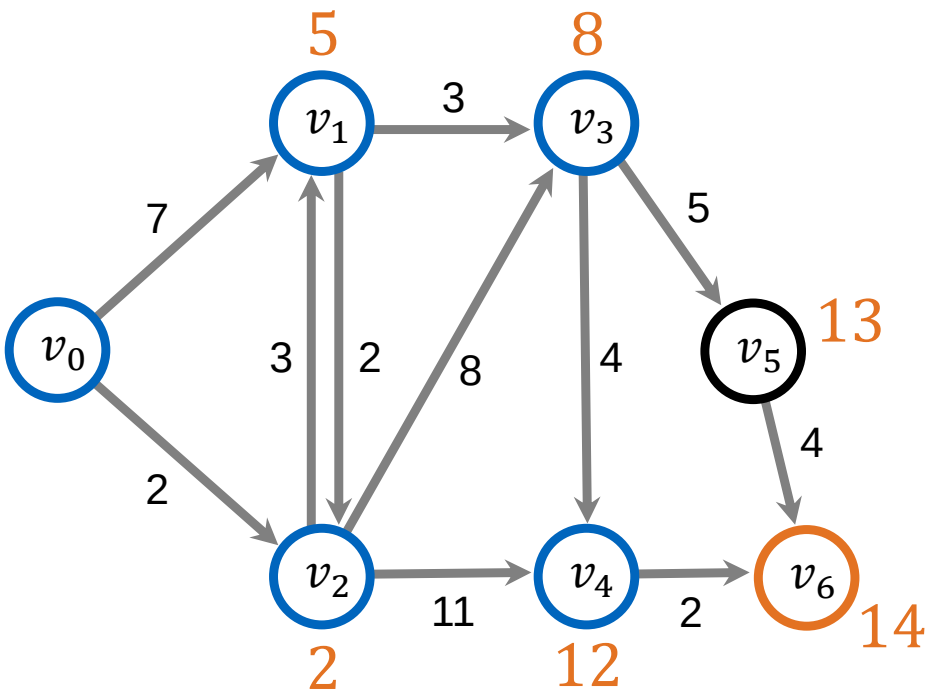
Step 4:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
d[v]	0	5	2	8	12	13	∞
p[v]	-	v_2	v_0	v_1	v_3	v_3	-
c[v]	✓	✓	✓	✓	-	-	-

Priority Queue:

v	v_4	v_5	v_6
d[v]	12	13	∞

Global Planning
Dijkstra‘s Algorithm



d=distance
p=predecessor
c=considered

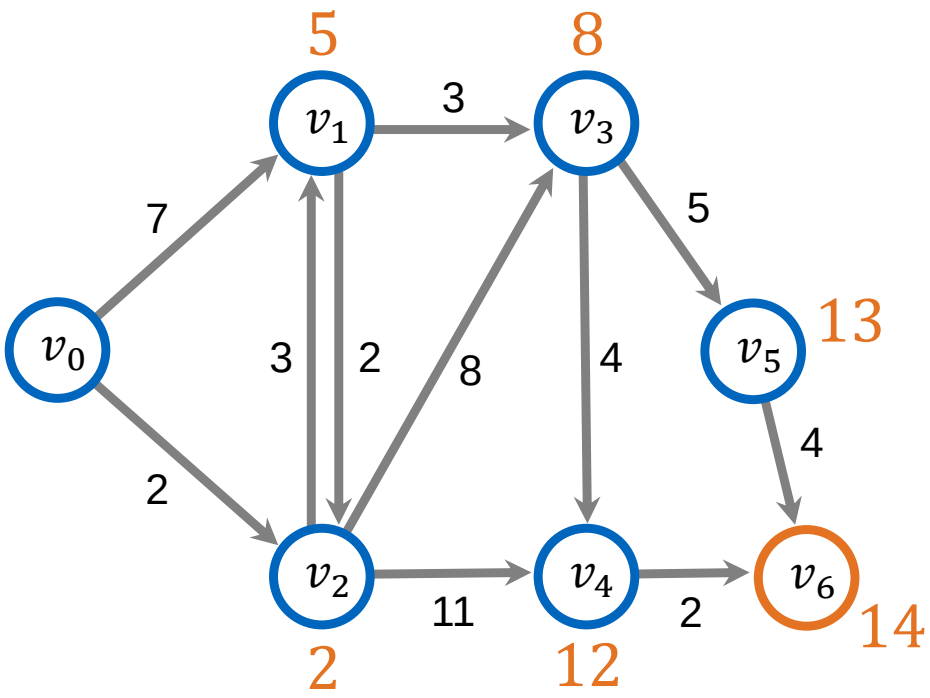
Step 5:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
d[v]	0	5	2	8	12	13	14
p[v]	-	v_2	v_0	v_1	v_3	v_3	v_4
c[v]	✓	✓	✓	✓	✓	-	-

Priority Queue:

v	v_5	v_6
d[v]	13	14

Global Planning
Dijkstra‘s Algorithm



d=distance
p=predecessor
c=considered

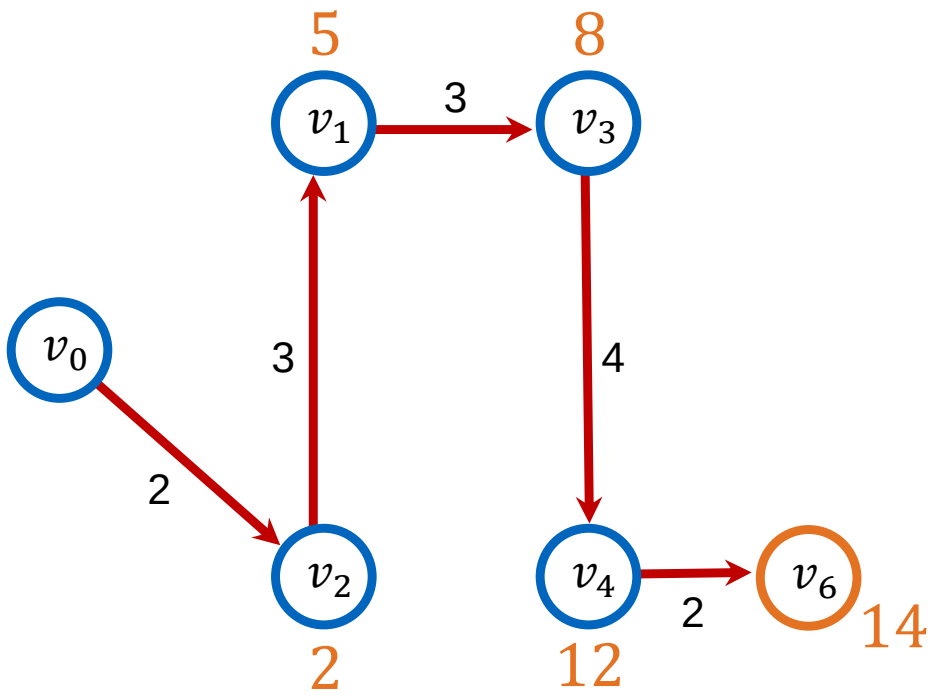
Step 6:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
d[v]	0	5	2	8	12	13	14
p[v]	-	v_2	v_0	v_1	v_3	v_3	v_4
c[v]	✓	✓	✓	✓	✓	✓	-

Priority Queue:

v	v_6
d[v]	14

Global Planning
Dijkstra‘s Algorithm



d=distance
p=predecessor
c=considered

Step 7:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
d[v]	0	5	2	8	12	13	14
p[v]	-	v_2	v_0	v_1	v_3	v_3	v_4
c[v]	✓	✓	✓	✓	✓	✓	✓

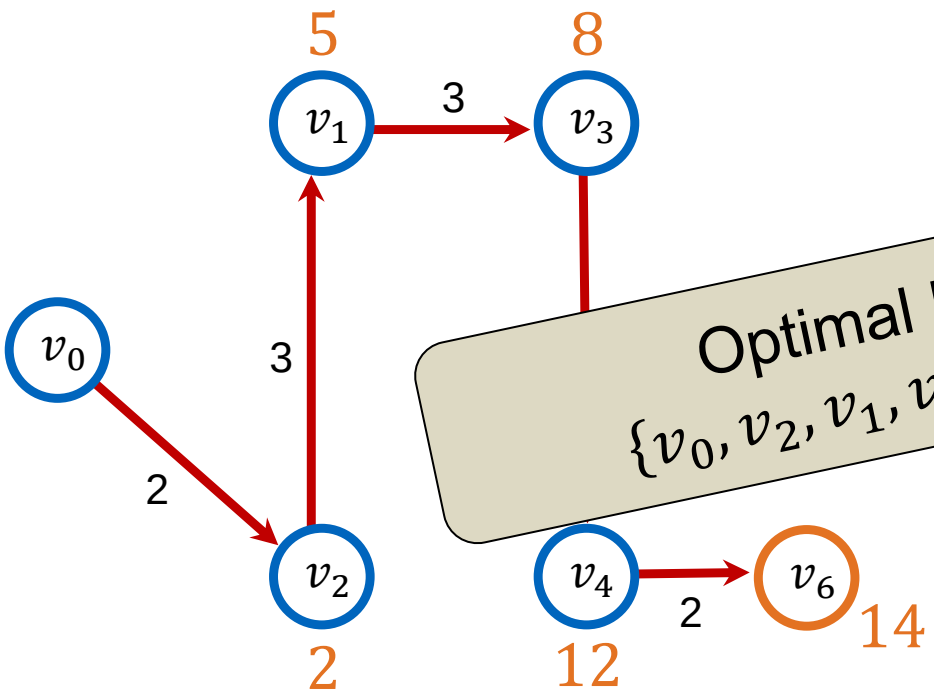
Priority Queue:

v	–
d[v]	–

$Q = \emptyset$

Global Planning
Dijkstra's Algorithm

d=distance
p=predecessor
c=considered



Step 7:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
d[v]	0	5	2	8	12	13	14
	v_0	v_1	v_3	v_3	v_4		
	✓	✓	✓	✓	✓		

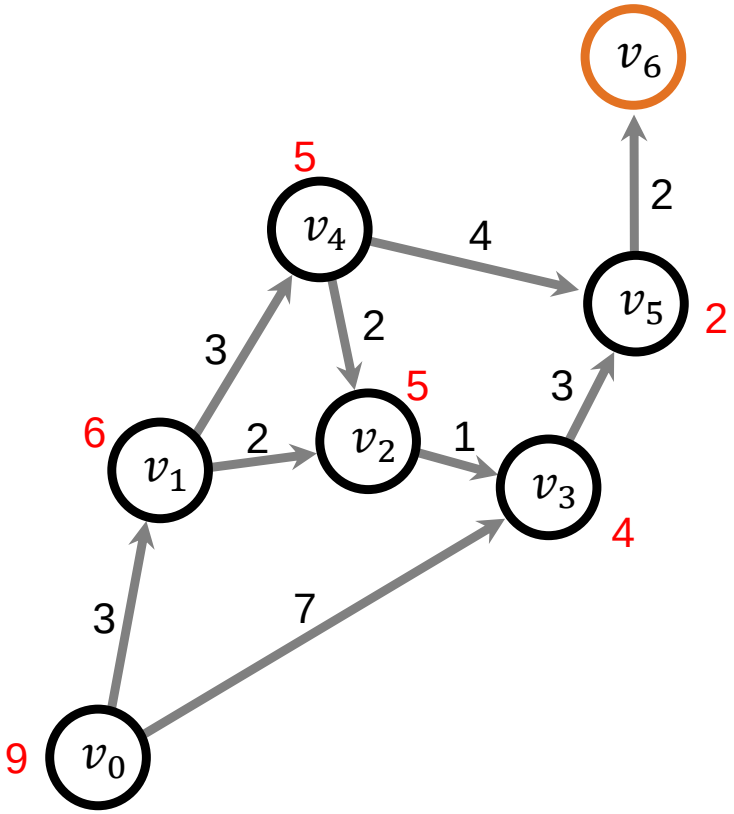
Optimal Path
$\{v_0, v_2, v_1, v_3, v_4, v_6\}$

Priority Queue:

v	–
d[v]	–

$Q = \emptyset$

Global Planning
A*-Algorithm



g=distance so far
h=v to destination
f=total estimated cost
p=predecessor
c=considered

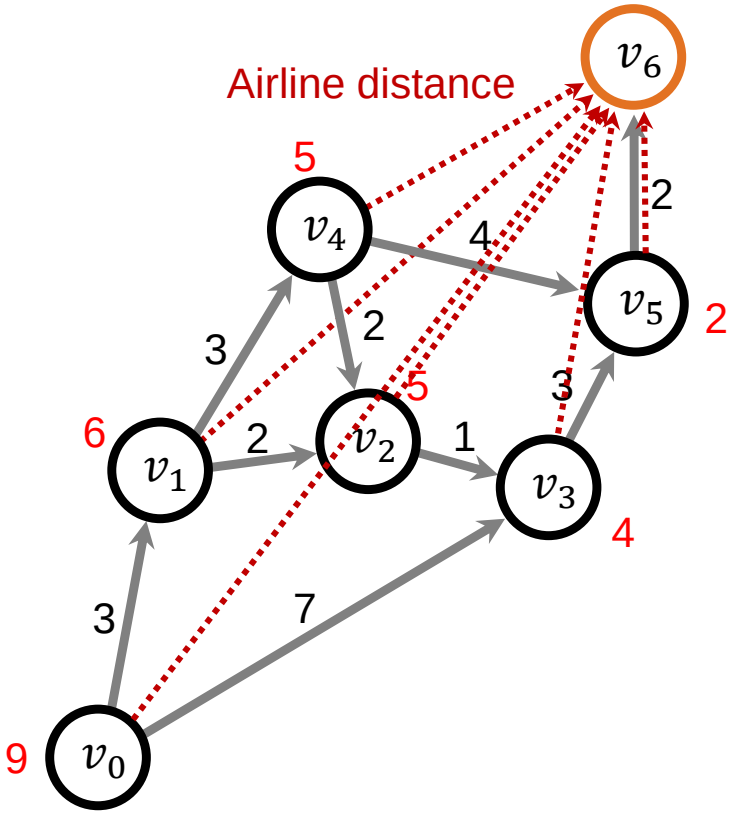
Step 0:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
g[v]	0	∞	∞	∞	∞	∞	∞
h[v]	9	6	5	4	5	2	-
f[v]	9	-	-	-	-	-	-
p[v]	-	-	-	-	-	-	-
c[v]							

Priority Queue:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
f[v]	9	∞	∞	∞	∞	∞	∞

Global Planning
A*-Algorithm



g=distance so far
h=v to destination
f=total estimated cost
p=predecessor
c=considered

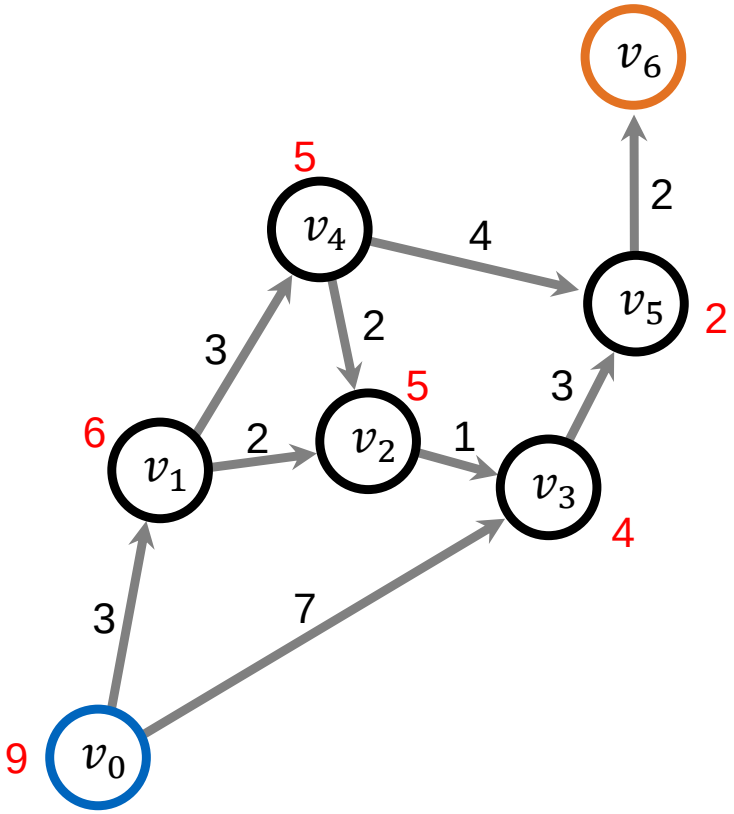
Step 0:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
g[v]	0	∞	∞	∞	∞	∞	∞
h[v]	9	6	5	4	5	2	-
f[v]	9	-	-	-	-	-	-
p[v]	-	-	-	-	-	-	-
c[v]							

Priority Queue:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
f[v]	9	∞	∞	∞	∞	∞	∞

Global Planning
A*-Algorithm



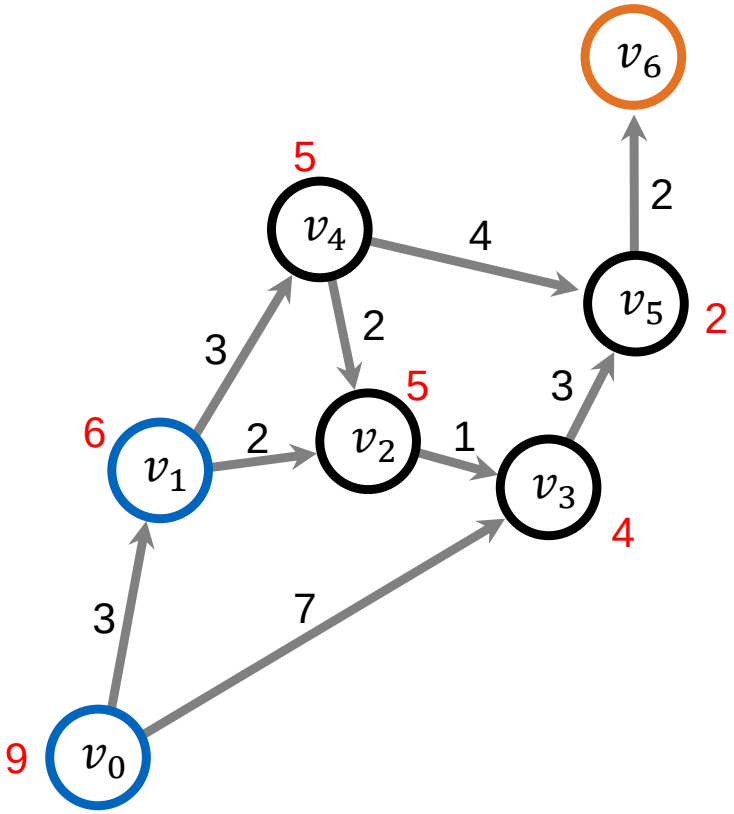
Step 1:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
g[v]	0	3	∞	7	∞	∞	∞
h[v]	9	6	5	4	5	2	-
f[v]	9	9	-	11	-	-	-
p[v]	-	v_0	-	v_0	-	-	-
c[v]	✓						

Priority Queue:

v	v_1	v_2	v_3	v_4	v_5	v_6
f[v]	9	∞	11	∞	∞	∞

Global Planning
A*-Algorithm



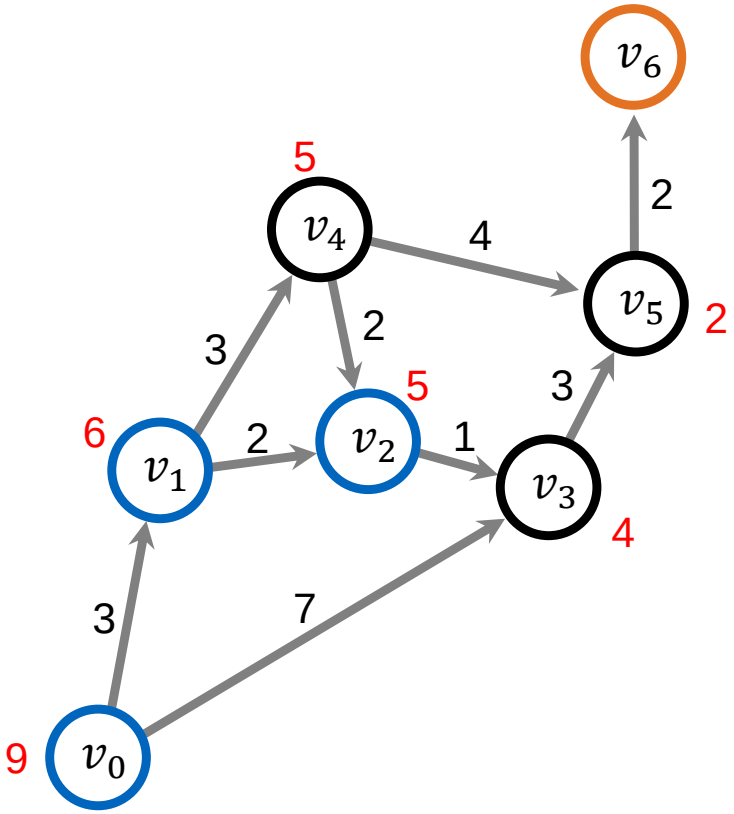
Step 2:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
g[v]	0	3	5	7	6	∞	∞
h[v]	9	6	5	4	5	2	-
f[v]	9	9	10	11	11	-	-
p[v]	-	v_0	v_1	v_0	v_1	-	-
c[v]	✓	✓					

Priority Queue:

v	v_2	v_3	v_4	v_5	v_6
f[v]	10	11	11	∞	∞

Global Planning
A*-Algorithm



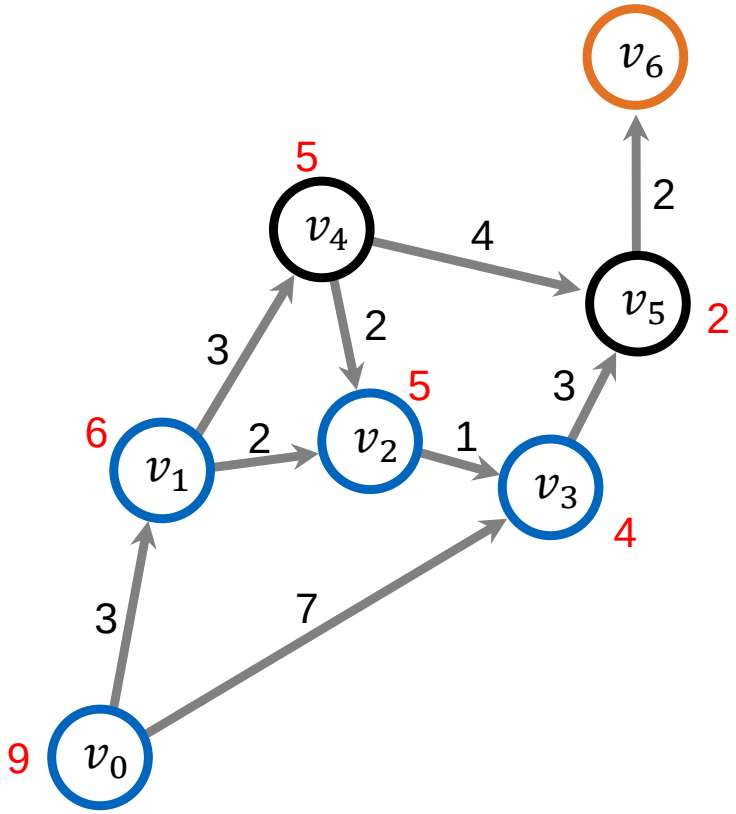
Step 3:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
g[v]	0	3	5	6	6	∞	∞
h[v]	9	6	5	4	5	2	-
f[v]	9	9	10	10	11	-	-
p[v]	-	v_0	v_1	v_2	v_1	-	-
c[v]	✓	✓	✓				

Priority Queue:

v	v_3	v_4	v_5	v_6
f[v]	10	11	∞	∞

Global Planning
A*-Algorithm



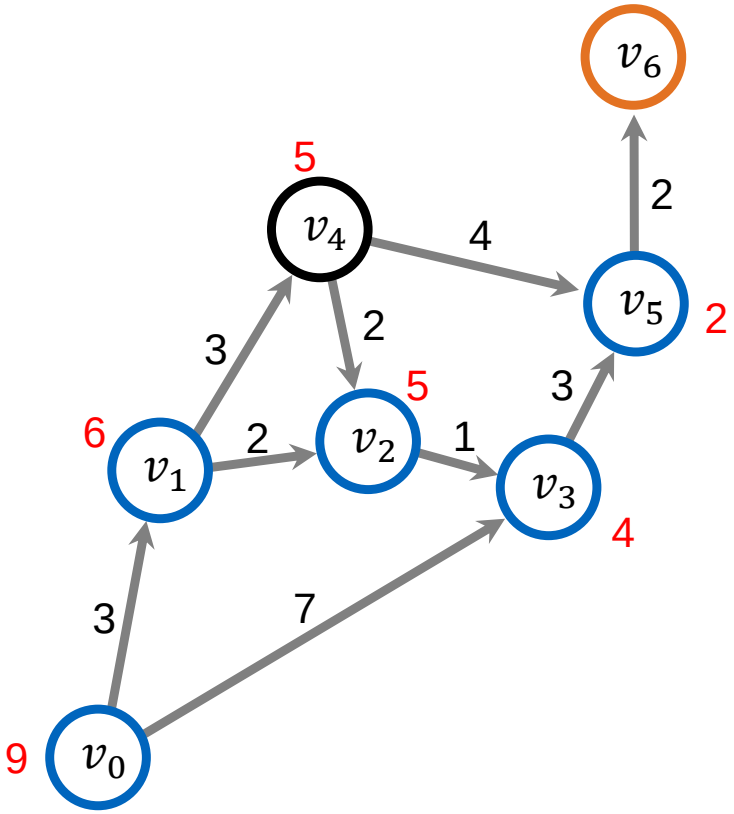
Step 4:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
g[v]	0	3	5	6	6	9	∞
h[v]	9	6	5	4	5	2	-
f[v]	9	9	10	10	11	7	-
p[v]	-	v_0	v_1	v_2	v_1	v_3	-
c[v]	✓	✓	✓	✓			

Priority Queue:

v	v_4	v_5	v_6
f[v]	11	7	∞

Global Planning
A*-Algorithm



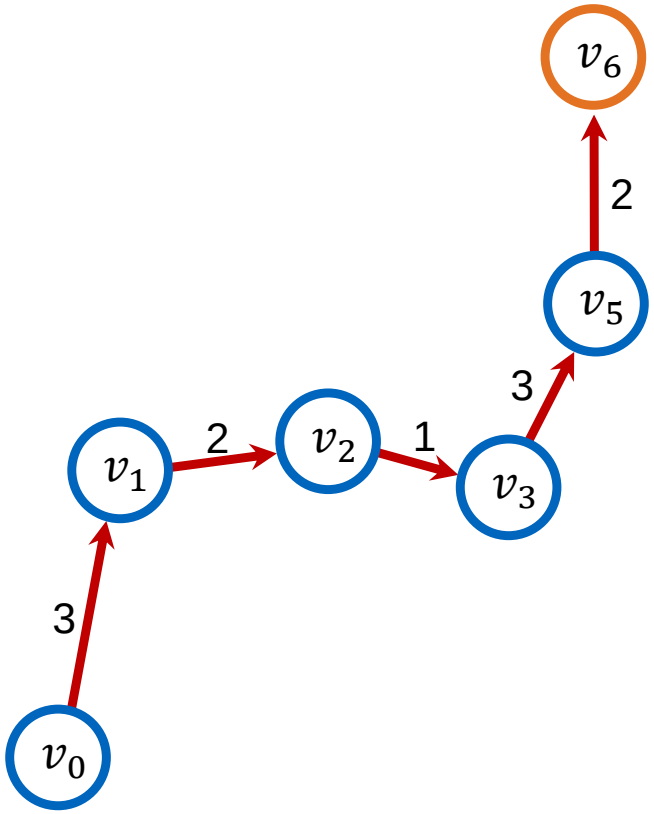
Step 5:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
g[v]	0	3	5	6	6	9	11
h[v]	9	6	5	4	5	2	-
f[v]	9	9	10	10	11	7	11
p[v]	-	v_0	v_1	v_2	v_1	v_3	v_5
c[v]	✓	✓	✓	✓		✓	

Priority Queue:

v	v_4	v_6
f[v]	11	11

Global Planning
A*-Algorithm



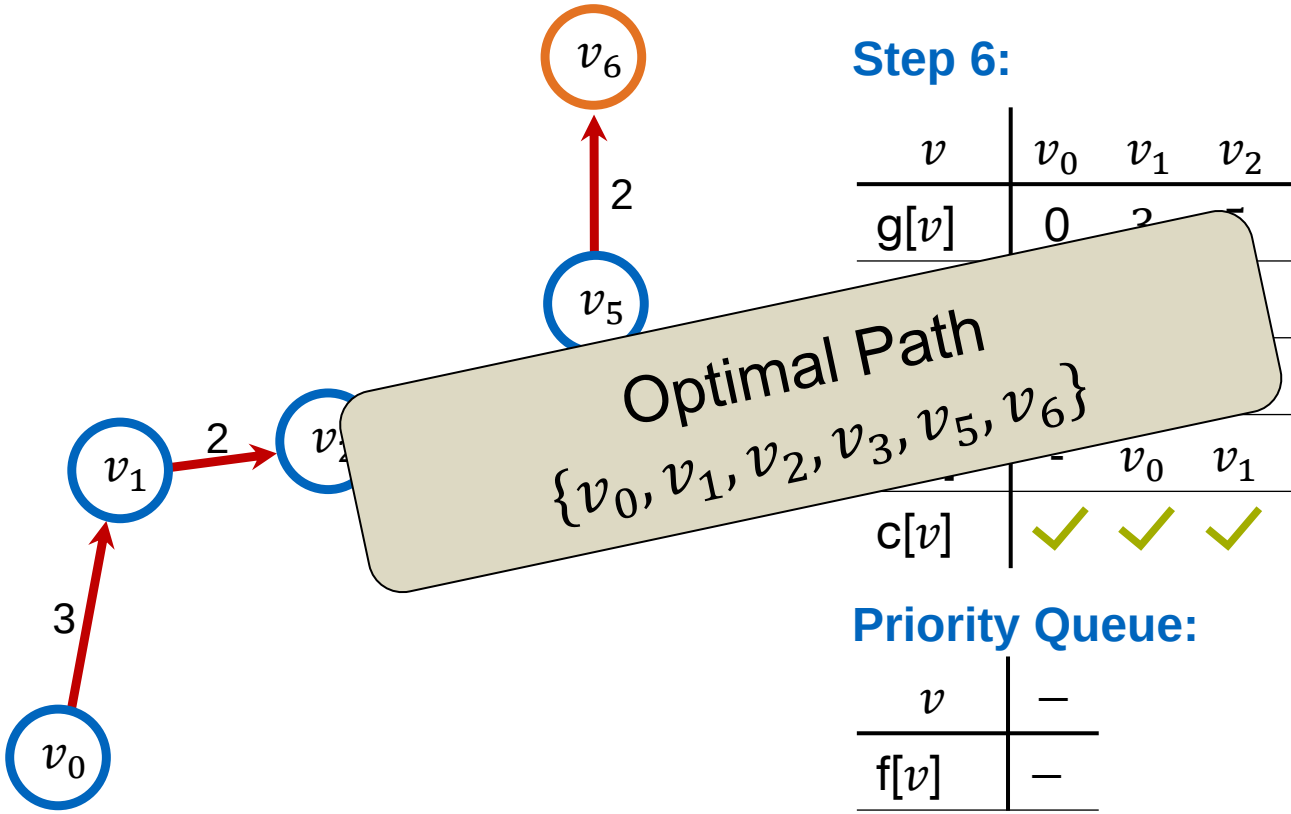
Step 6:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
g[v]	0	3	5	6	6	9	11
h[v]	9	6	5	4	5	2	-
f[v]	9	9	10	10	11	7	11
p[v]	-	v_0	v_1	v_2	v_1	v_3	v_5
c[v]	✓	✓	✓	✓		✓	✓

Priority Queue:

v	–
f[v]	–

Global Planning
A*-Algorithm



Step 6:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
g[v]	0	3	5	6	6	9	11
				4	5	2	-
				10	11	7	11
	-	v_0	v_1	v_2	v_1	v_3	v_5
c[v]	✓	✓	✓	✓		✓	✓

Priority Queue:

v	–
f[v]	–