
HE
IG
VD

Intelligence Artificielle pour les systèmes autonomes (IAA)

Modular pipeline: Object tracking and mission planning

Prof. Yann Thoma - Prof. Marina Zapater

Février 2024

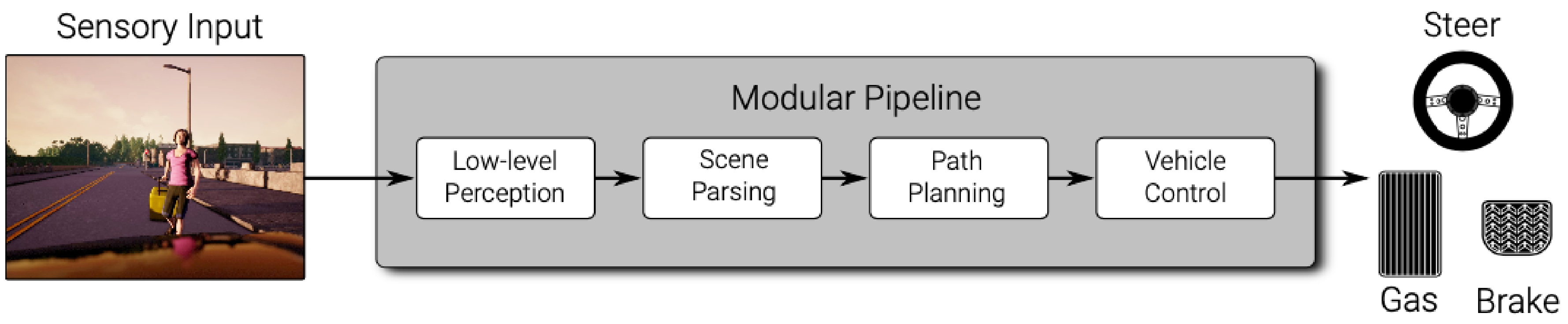
Basé sur le cours du Prof. A. Geiger

Hes·so
Haute Ecole Spécialisée
de Suisse occidentale

ReDS
Reconfigurable & embedded
Digital Systems

Modular Pipeline

Reminder of main blocks



→ Vehicle control

→ Low-level perception : Odometry, SLAM and global localization

→ Scene Parsing : **Object tracking**

→ **Path planning**

Summary

Today’s lesson

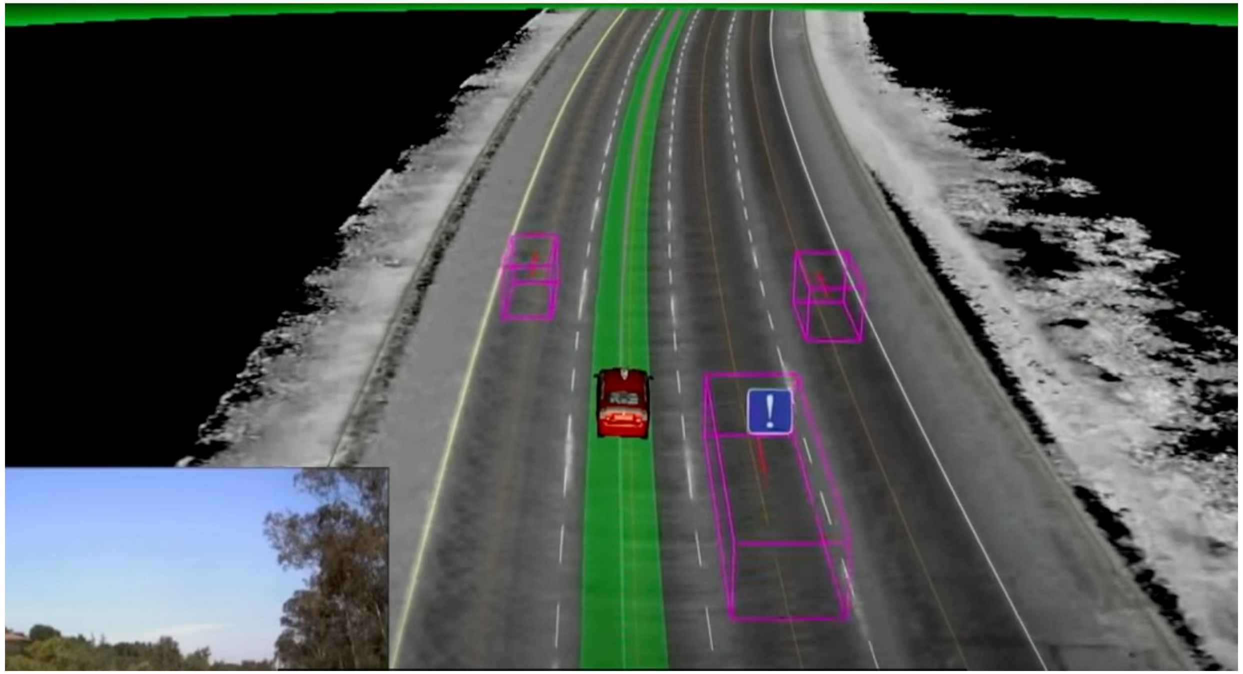
→ **Object tracking**

→ Holistic Scene Understanding

→ Path planning

What did we do in previous lecture?

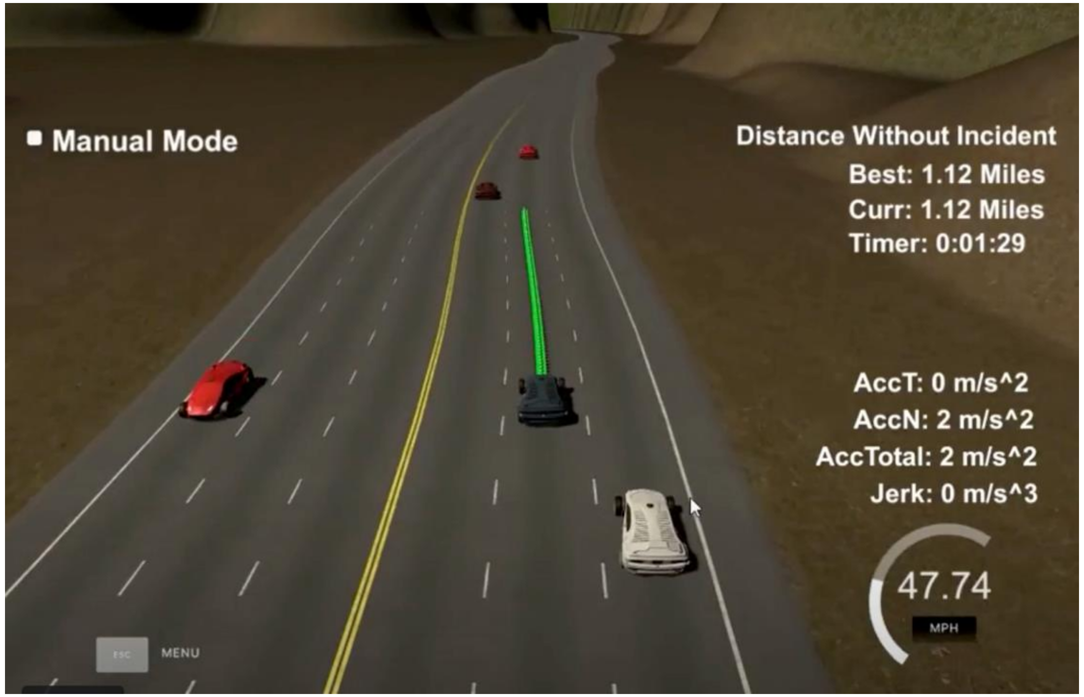
Analyse this video thinking about the theory of Chapter 8



1. What is the processing being done in this video?
2. Is it 2D or 3D? What are the main challenges?
3. What algorithms did we use for this?

What are the differences with the previous video?

What are we doing beyond what we just saw?



1. What is the car above doing (what kind of processing)? What are the inputs/outputs?
2. How would you tackle this problem?

What is object tracking?

Detecting, associating, filtering

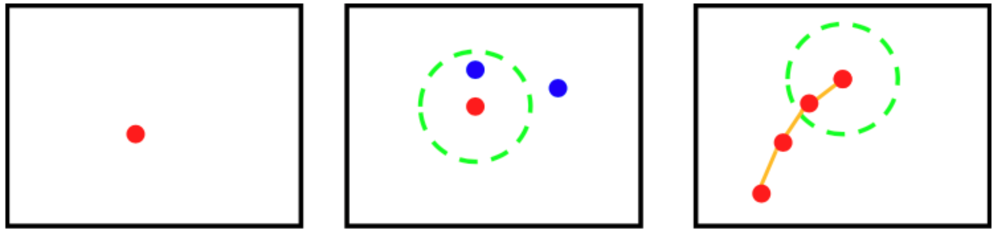
→ Given noisy object detections (bounding boxes) for each frame in a sequence, **associate** those to the same physical object

→ Reject detections that are false alarms

→ <u>Detection</u>: which are the candidate objects?

→ <u>Association</u>: which detection corresponds to which object?

→ <u>Filtering</u>: what is the most likely object state (e.g.: location, size)



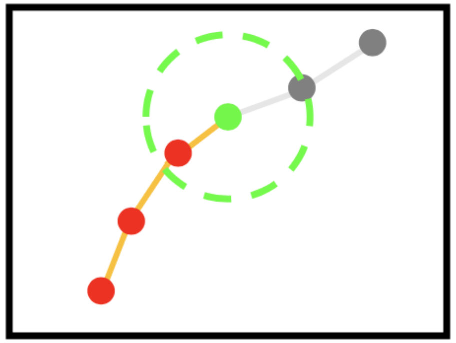
Detection Association Filtering

Online tracking for single objects

Filtering

→ Online tracking estimates current state given current and past observations
- (Off-line tracking uses all given observations – present and future)

→ If we track a single object, only detection and **filtering** are needed. No need for association.



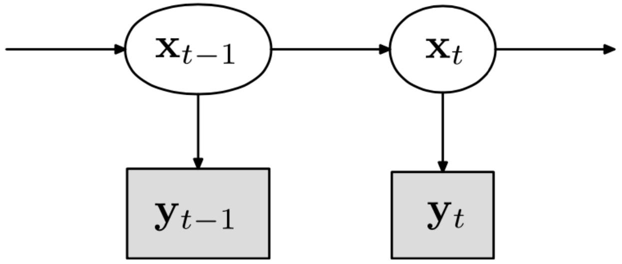
Online Tracking

→ The moving object is characterized by an underlying state x

→ State x can provide measurements or observations y

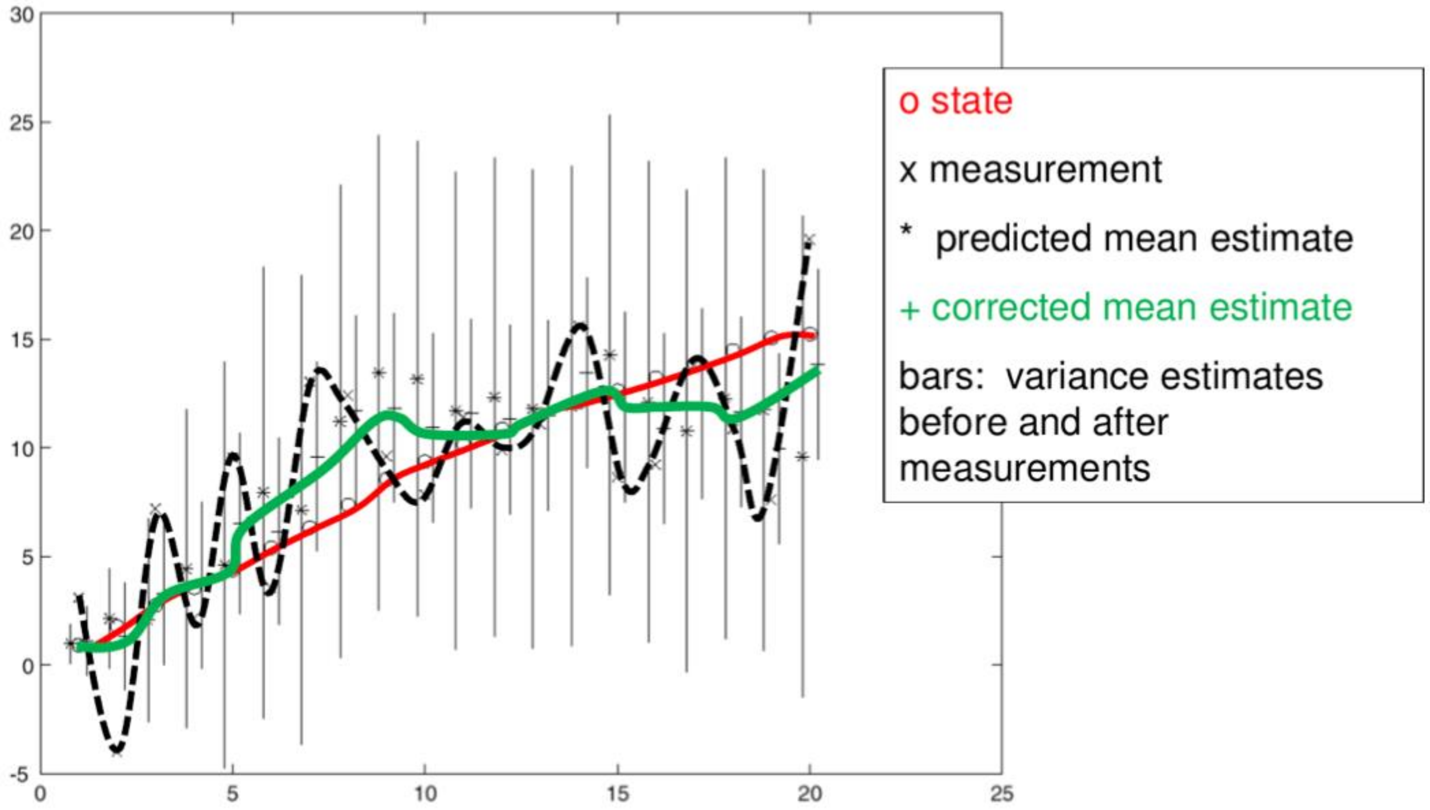
→ At each time t, state changes to x_t and provides observation y_t

→ Given a sequence of observations $Y=\{y_t\}$ we want to recover $X=\{x_t\}$



Filtering example

Using probabilistic estimates



Recursive Bayesian Estimation

Using Bayes Filtering for probabilistic inference

→ We assume that at each time there is only one object to track

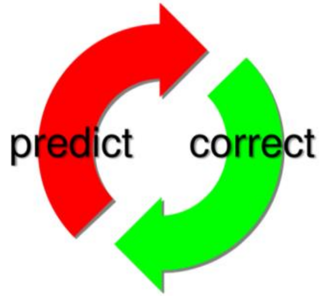
→ We assume that he have one (noisy) observation available per time t (per frame)

→ A Bayes filter is a probabilistic approach for estimating an unknown probability density function recursively and over time using:

- Incoming measurements. For example: detected object location
- A system process model. For example: constant velocity motion model

→ Two alternating steps:

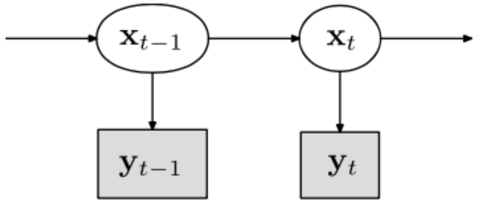
- <u>Predict</u> where the object should be in the next frame
- <u>Correct</u> prediction based on the current observation



The Bayes Filter

A bit of math for the interested ones

$$p(\mathbf{X}, \mathbf{Y}) = p(\mathbf{x}_1) \prod_{t=2}^{T} p(\mathbf{x}_t | \mathbf{x}_{t-1}) \prod_{t=1}^{T} p(\mathbf{y}_t | \mathbf{x}_t)$$



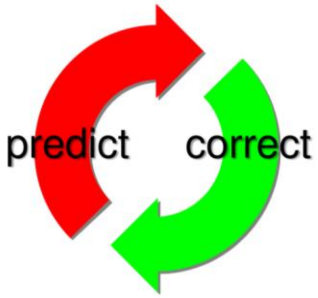
The Kalman Filter

One of the simplest Bayes filters

→ We assume linearity and Gaussian distribution of noise

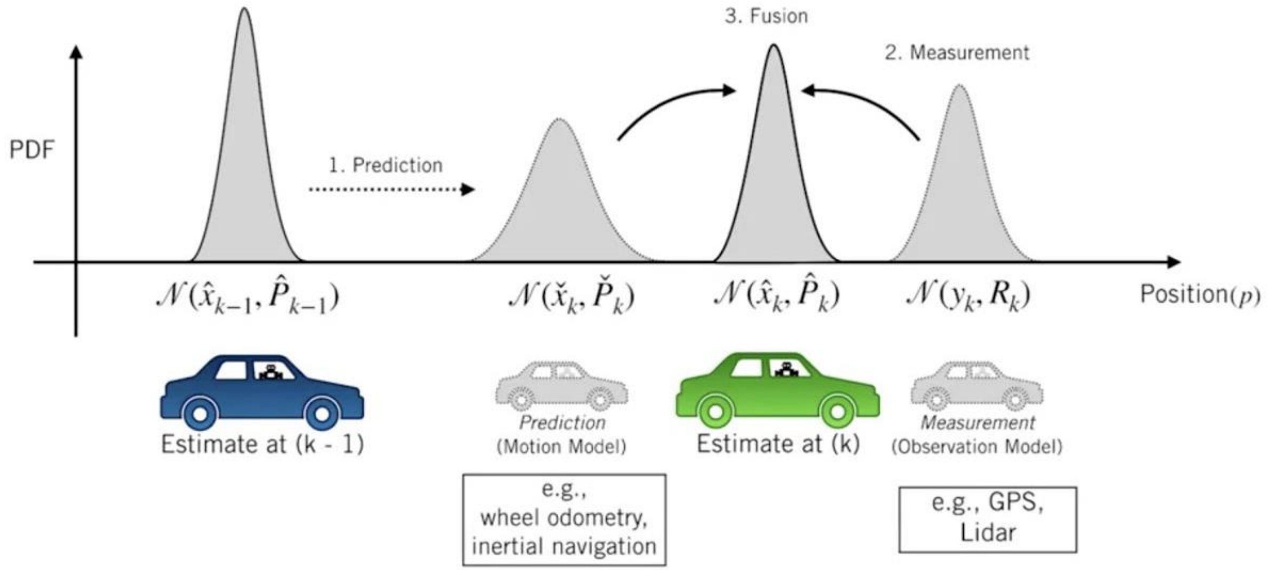
→ Solved by means of a Least Squares Regression

→ Based on the same predict-correct principle than before



→ Widely used in guidance, navigation and control of vehicles

- Aircraft, spacecraft and ship positioning
- Apollo 11 Lunar mission



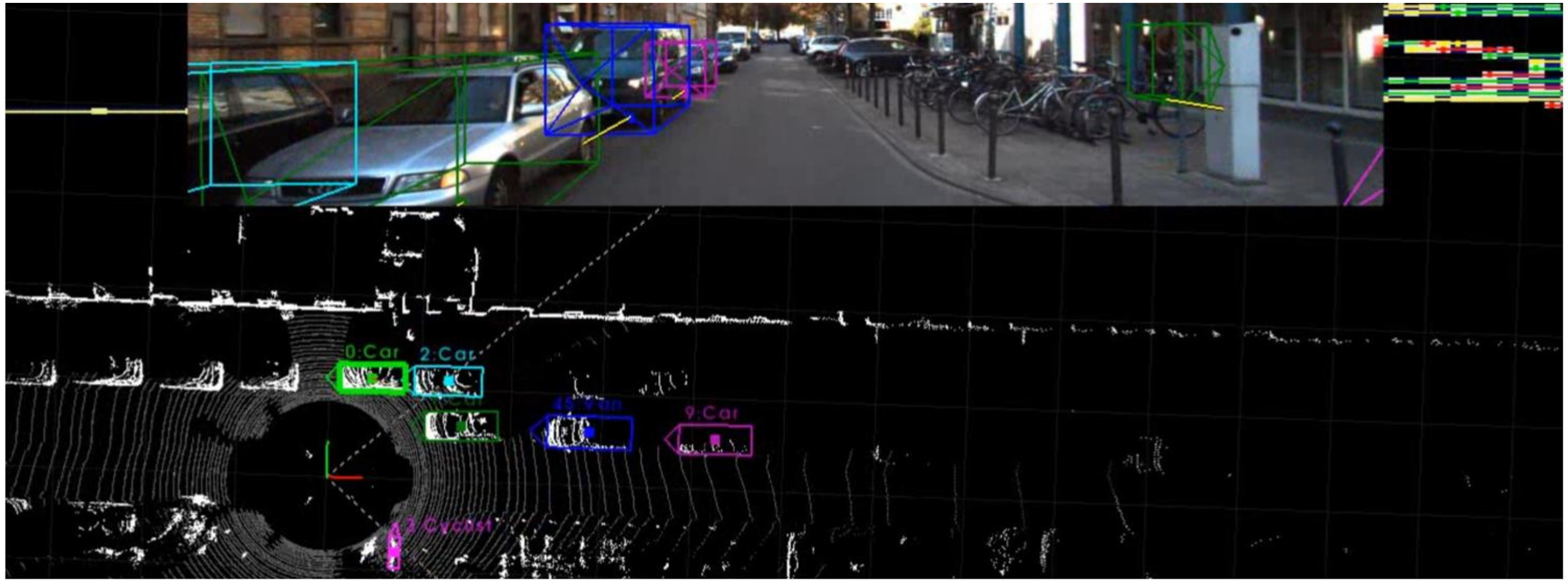
https://en.wikipedia.org/wiki/Kalman_filter

Multi-object Tracking

Association comes into play

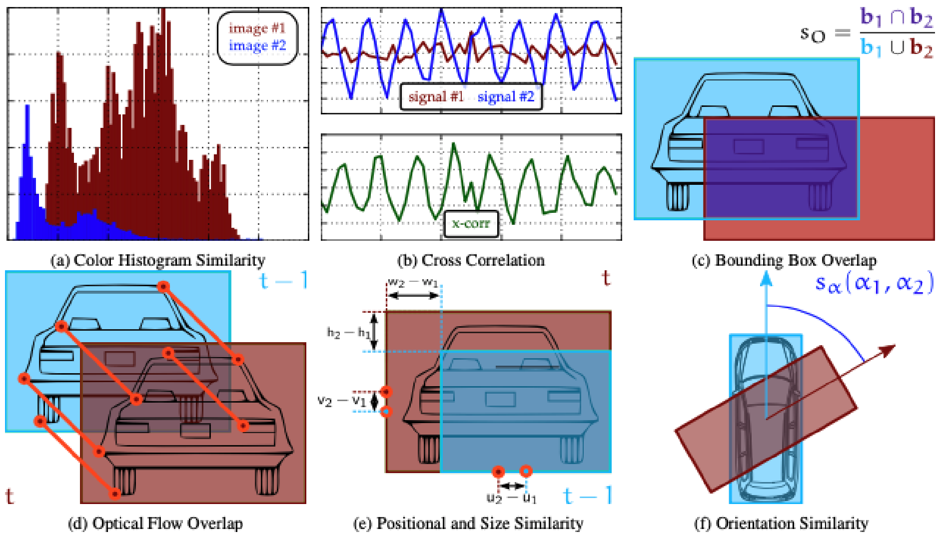
→ Real-case: we have to track multiple objects at the same time.

→ Goal of association: associate detections in a frame to existing object tracks



Multi-object tracking

Object Association Measures



(a) Color Histogram Similarity (b) Cross Correlation (c) Bounding Box Overlap (d) Optical Flow Overlap (e) Positional and Size Similarity (f) Orientation Similarity

Algorithm:

1. Predict objects from previous frame and detect objects in current frame
2. Associate detections to object tracks
3. Correct predictions with observations (Kalman Filter)

When do observations belong to the same object?

Several techniques:

→ Predict bounding box (via motion model) and measure overlap

→ Compare color histograms or normalized cross-correlation

→ Estimate optical flow and measure agreement

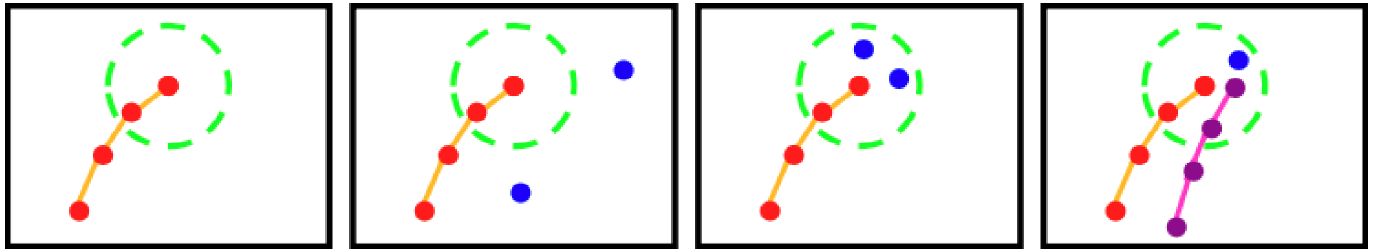
→ Compare relative location and size of bounding box

→ Compare orientation of detected objects

Solving correspondence Ambiguities

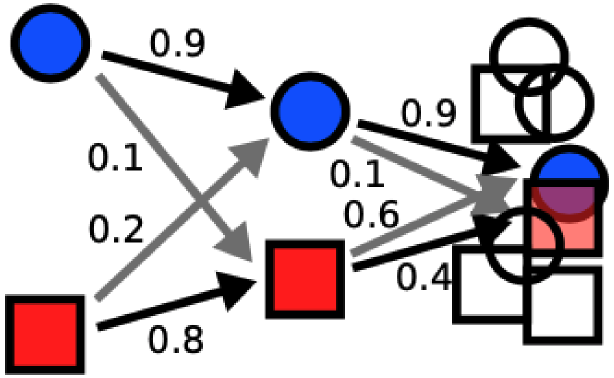
Nearest Neighbor Association

→ Issue: Objects can cease to exist (or be ocluded), measurements can be unexpected, a measurement can match multiple predictions, etc.



→ Proposal: only consider measurement within a certain area around the prediction

- Associate detection with the closest prediction
- Numbers represent association scores (higher is better)
- Still… this fails because of ambiguities…



Bipartite Graph Matching

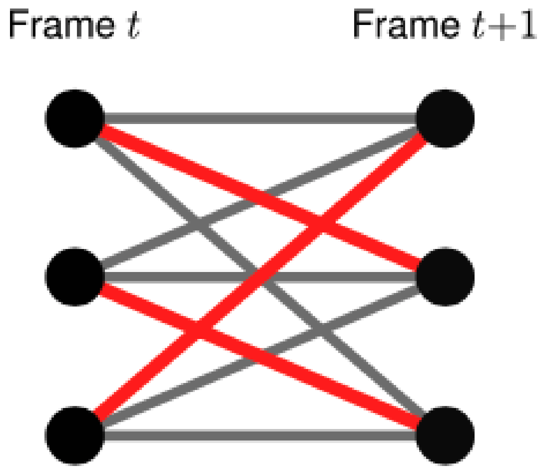
One of the most common techniques

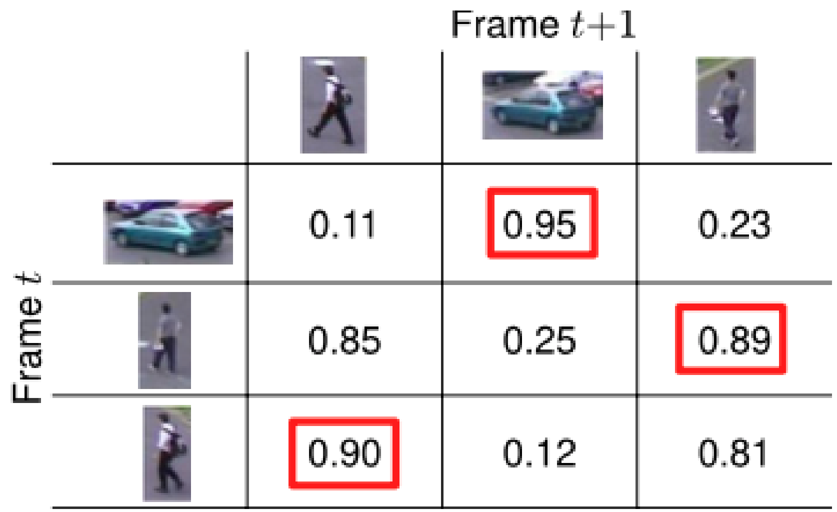
→ A bipartite graph (bigraph) is a graph whose vertices can be divided into two disjoint sets

→ Given a bipartite graph, a matching is a subset of the edges for which every vertex belongs to exactly one of the edges

→ Frame-to-frame tracking can be formulated as a bipartite graph matching problem

- Chooses at most one match in each row and column while maximizing score





A more formal definition

Bipartite Graph Matching

→ If we have N detections in the previous frame and M detections in the current frame

→ We can construct a M x N table of matching scores

→ **Task**: choose the 1:1 matching that maximizes the sum of scores

	1	2	3	4	5
1	0.95	0.76	0.62	0.41	0.06
2	0.23	0.46	0.79	0.94	0.35
3	0.61	0.02	0.92	0.92	0.81
4	0.49	0.82	0.74	0.41	0.01
5	0.89	0.44	0.18	0.89	0.14

→ How many differences assignments are possible?

- 5 x 4 x 3 x 2 x 1 = 120 (N!)

→ Exhaustive search becomes intractable very quickly!

- It can be solved by in integer linear program (ILP)
- Or by greedy techniques

Greedy Matching

Using a Greedy Algorithm for bipartite graph matching

→ Start with an unmarked matrix

→ For i=1..N

- Mark largest value in the entire matrix
  that isn’t in a row/column with already marked entries

	1	2	3	4	5
1	$\boxed{0.95}$	~~0.76~~	~~0.62~~	~~0.41~~	~~0.06~~
2	~~0.23~~	~~0.46~~	~~0.79~~	$\boxed{0.94}$	~~0.35~~
3	~~0.61~~	~~0.02~~	$\boxed{0.92}$	~~0.92~~	~~0.81~~
4	~~0.49~~	$\boxed{0.82}$	~~0.74~~	~~0.41~~	~~0.01~~
5	~~0.89~~	~~0.44~~	~~0.18~~	~~0.89~~	$\boxed{0.14}$

→ Greedy matching is easy to implement, quick to run and finds good solutions

- But not the optimal

	1	2	3	4	5
1	$\boxed{0.95}$	0.76	0.62	0.41	0.06
2	0.23	0.46	0.79	$\boxed{0.94}$	0.35
3	0.61	0.02	$\boxed{0.92}$	0.92	0.81
4	0.49	$\boxed{0.82}$	0.74	0.41	0.01
5	0.89	0.44	0.18	0.89	$\boxed{0.14}$

Greedy Solution: Score = 3.77

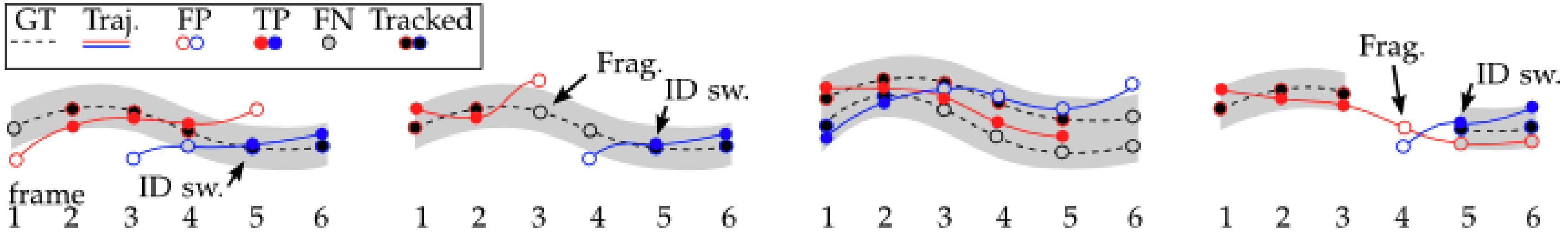
	1	2	3	4	5
1	$\boxed{0.95}$	0.76	0.62	0.41	0.06
2	0.23	0.46	$\boxed{0.79}$	0.94	0.35
3	0.61	0.02	0.92	0.92	$\boxed{0.81}$
4	0.49	$\boxed{0.82}$	0.74	0.41	0.01
5	0.89	0.44	0.18	$\boxed{0.89}$	0.14

Optimal Solution: Score = 4.26

Multi-object tracking evaluation

An open-source benchmark

→ Associate prediction with ground truth tracks (via bipartite graph matching)



→ **MOTA**: Multiple Object Tracking Accuracy

- FN/FP: False Negative/Positive detections
- IDSW: ID switches
- GT: GT objects

$$MOTA = 1 - \frac{\sum_t FN_t + FP_t + IDSW_t}{\sum_t GT_t}$$

→ **HOTA**: A Higher-Order Metric for Evaluating Multi-object Tracking

https://github.com/JonathonLuiten/TrackEval

HEIG-VD

Summary

Today’s lesson

→ Object tracking

→ **Holistic Scene Understanding**

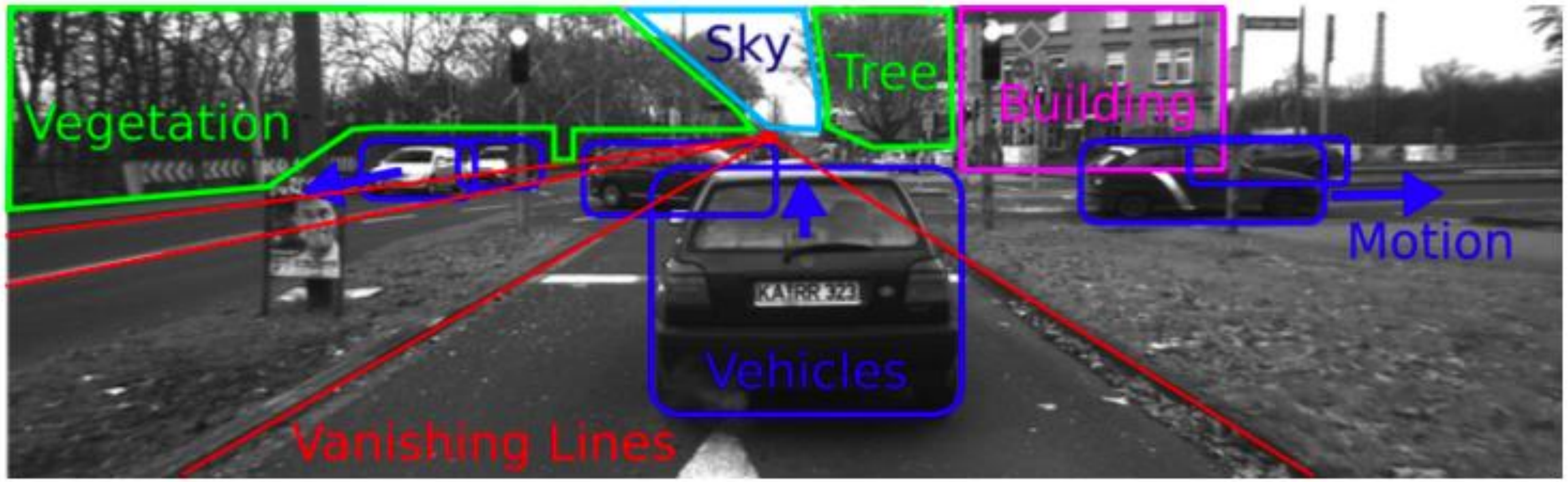
→ Path planning

An intersection from a human perspective

Putting eveything together and fusing information

→ We must fuse all cues to obtain a complete understanding

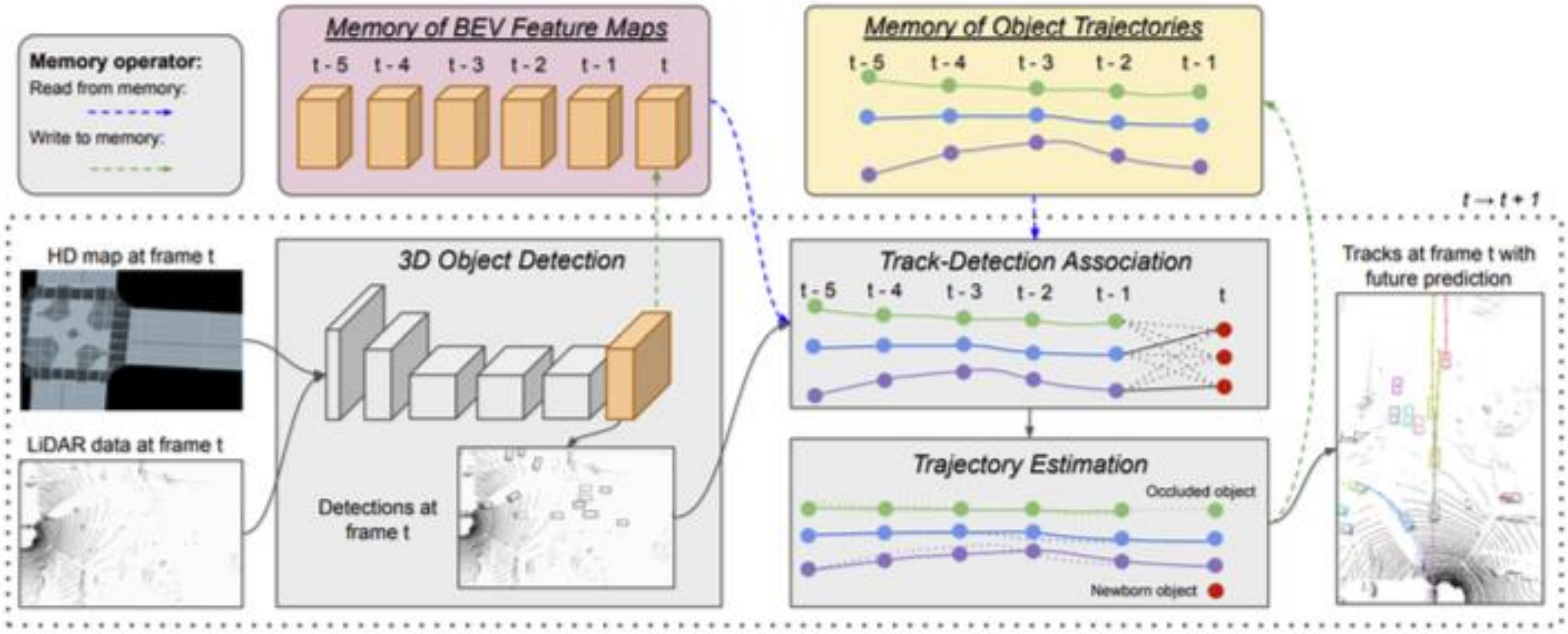
→ This fusion (should) lead to more robust estimates



→ We need to put everything together!

PnPNet: Perception and Prediction with Tracking in the Loop

Joint perception, tracking and motion forecasting / trajectory estimation



https://arxiv.org/abs/2005.14711

Summary

Today’s lesson

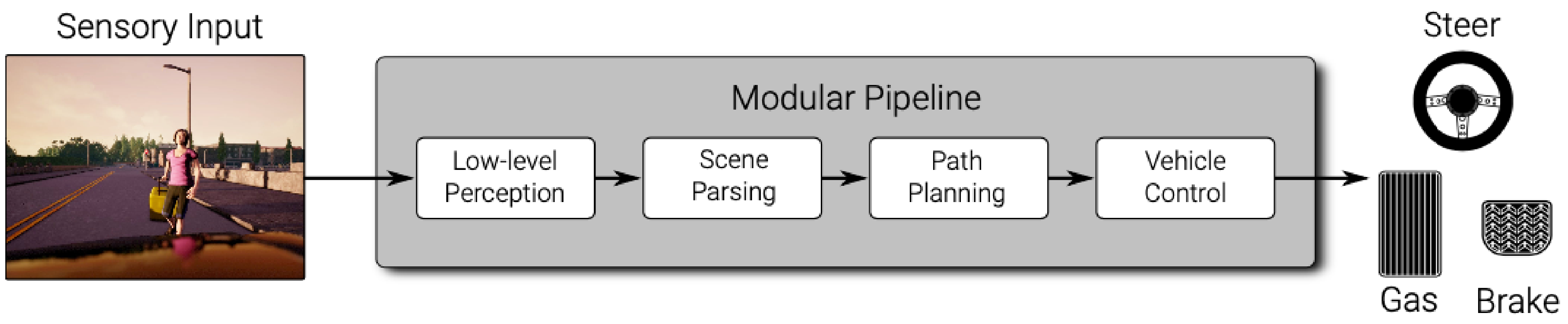
→ Object tracking

→ Holistic Scene Understanding

→ **Path planning**

Modular Pipeline

Reminder of main blocks



→ Vehicle control

→ Low-level perception : Odometry, SLAM and global localization

→ Scene Parsing : **Object tracking**

→ **Path planning**

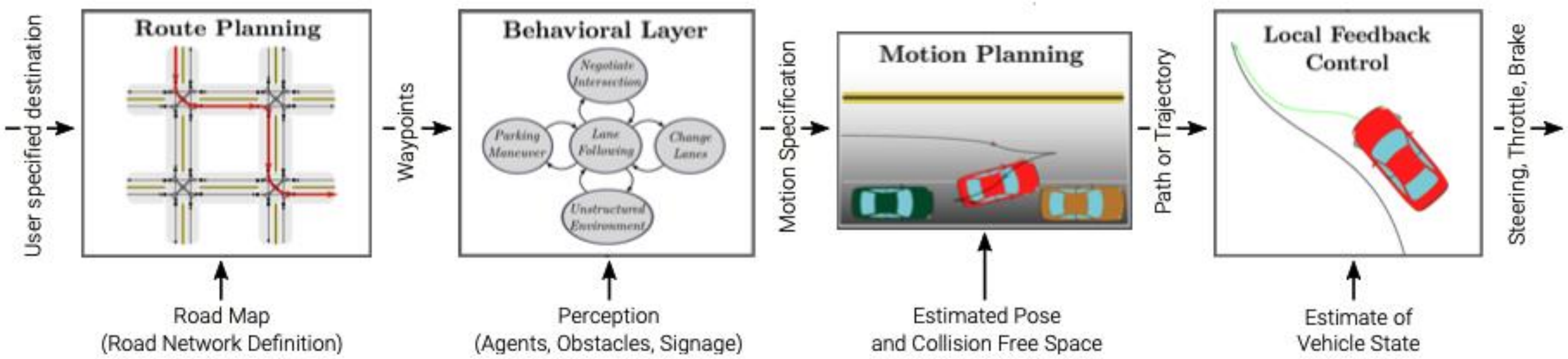
Path Planning and Decision Making

Breaking planning problem into a hierarchy of simpler problems

→ **Goal**: find and follow a path from current location to destination

→ **Input**: vehicle and environment state (via perception stack)

→ **Output**: path or trajectory as input to vehicle controller

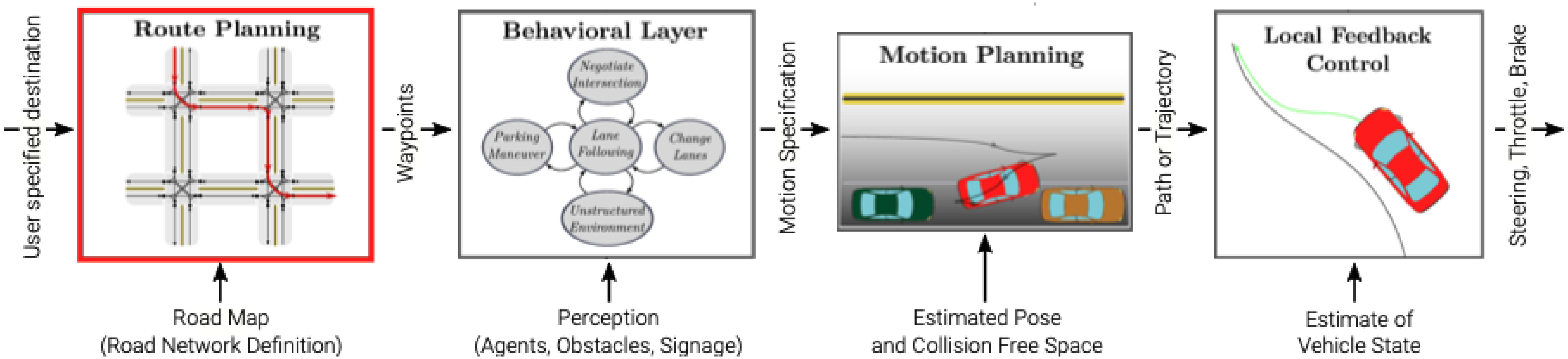


→ Challenges:

- Driving situations are very complex and difficult to model
- So we break the problem into pieces

Breaking the planning problem

Step 1: Route planning



→ Represent road network as directed graph

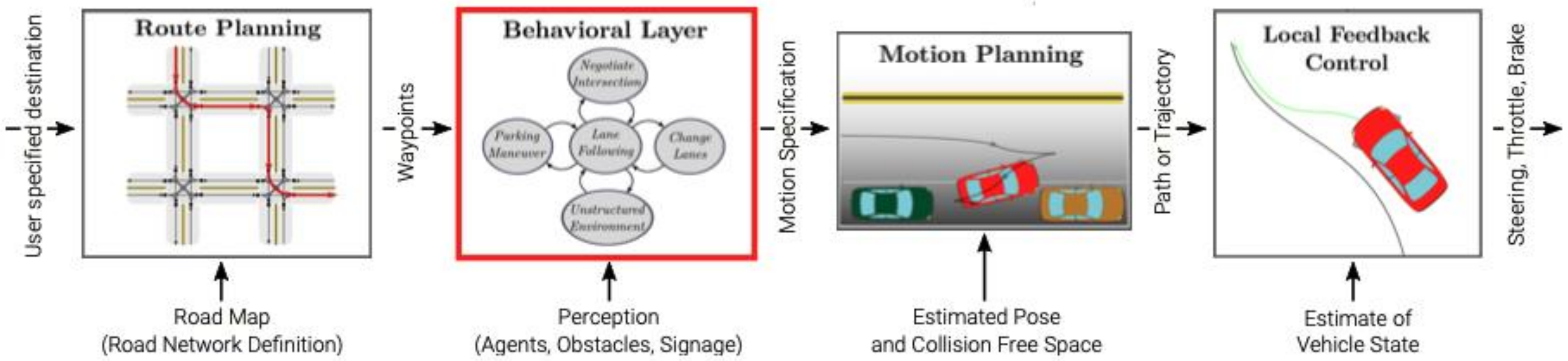
→ Edge weights correspond to road segments length (or travel time)

→ Problem translates into a **minimum-cost graph network problem**

- Dijkstra, A*, etc.

Breaking the planning problem

Step 2: Behaviour planning



→ Select driving behavior based on current vehicle/environment state

- E.g.: at stop line you must stop, observe traffic and traverse

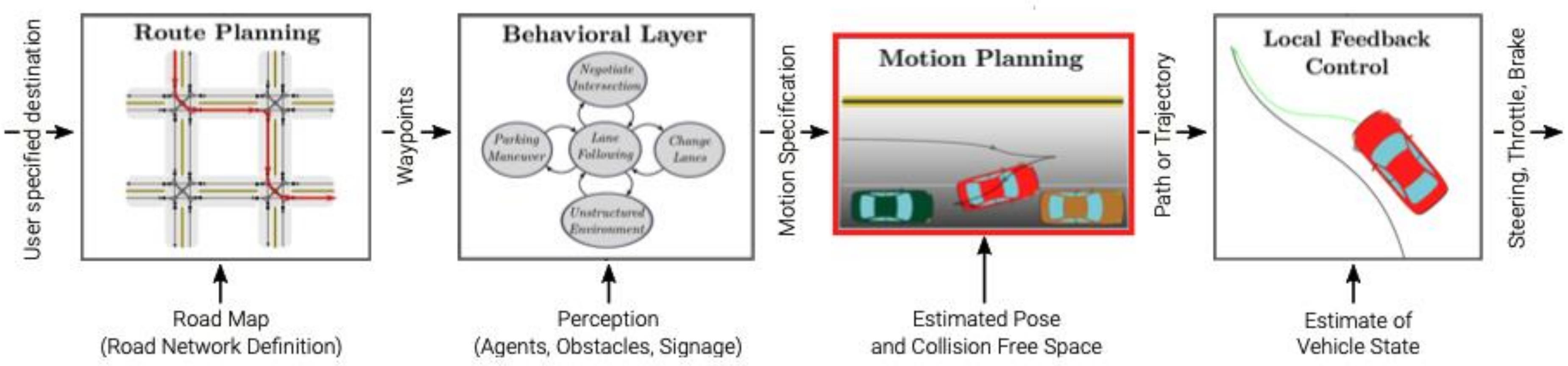
→ Often modeled via a **finite state machine**

- Transitions governed by perception

→ Can also be modeled by Markov Decision Processes (MDPs)

Breaking the planning problem

Step 3: Motion planning



→ Find feasible, comfortable, safe and fast vehicle path/trajectories

→ Exact solutions are intractable (too large space)

→ **Numerical approximations**

Summary

Today’s lesson

→ Object tracking

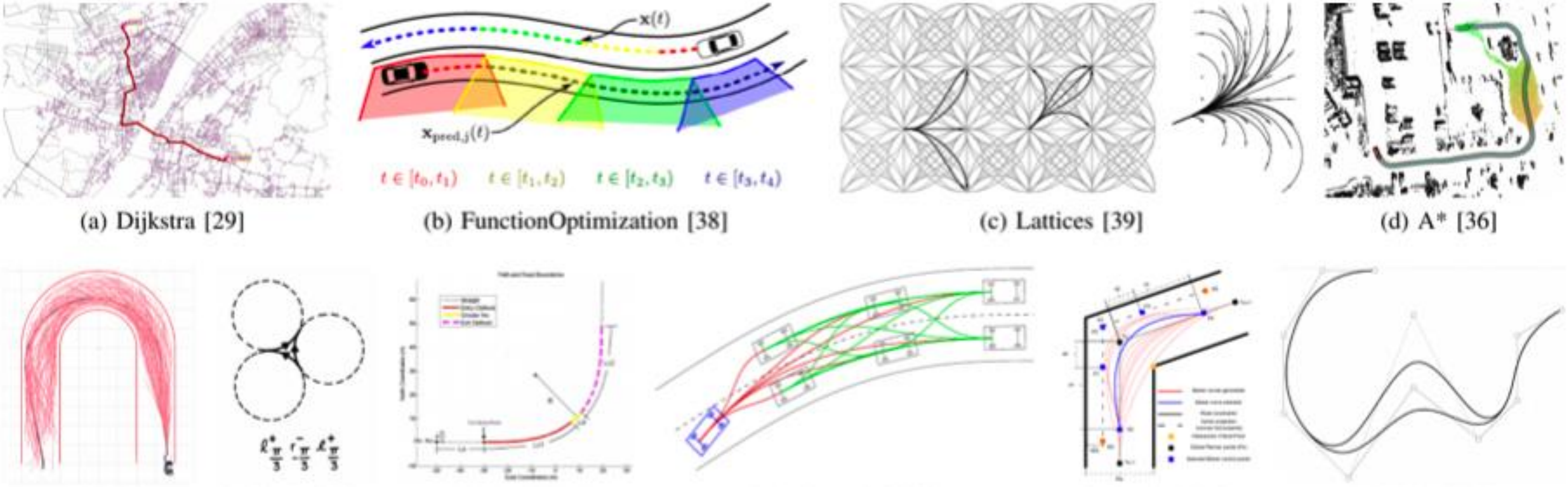
→ Holistic Scene Understanding

→ **Path planning**

- **Step 1: Route Planning**

Planning Algorithms

Many, many, many… we'll just see a couple today



(a) Dijkstra [29] (b) FunctionOptimization [38] (c) Lattices [39] (d) A* [36]

(e) RRT [40] (f) Line&Circle [41] (g) Clothoid [42] (h) Polynomial [43] (i) Bezier [44] (j) Spline [45]

https://arxiv.org/pdf/1604.07446.pdf

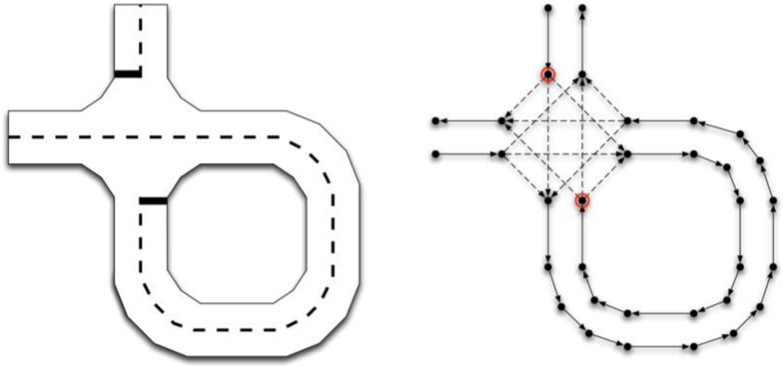
Route Planning

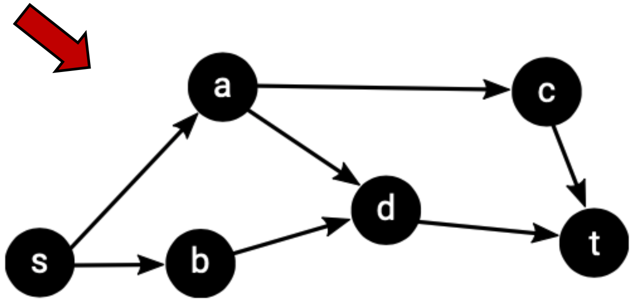
Navigate from point A to point B

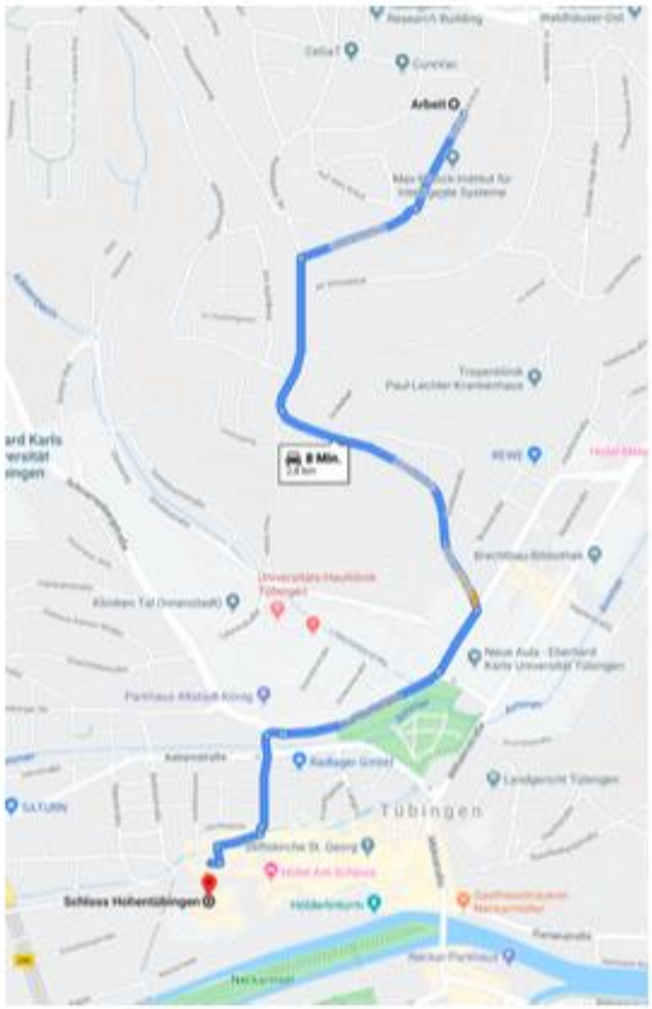
→ High-level planning where low details are abstracted

→ Road Networks can be seen as directed graphs

→ And we need to find the shortest path







Dijkstra’s Shortest Path Algorithm

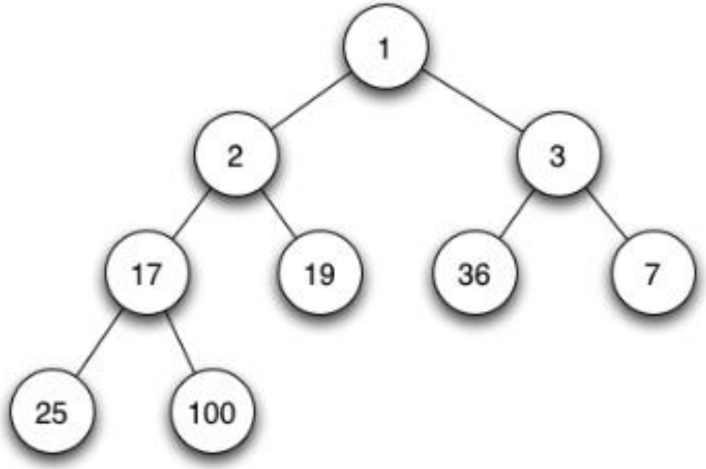
One of the most common algorithms to find the shortest path

→ Constructs a specialized tree-based data structure

→ **Min heap** property: the value of any node in the tree is smaller than all its children

→ Efficient implementation of a priority queue

- Smallest element stored at root note
- Partially ordered, but not sorted



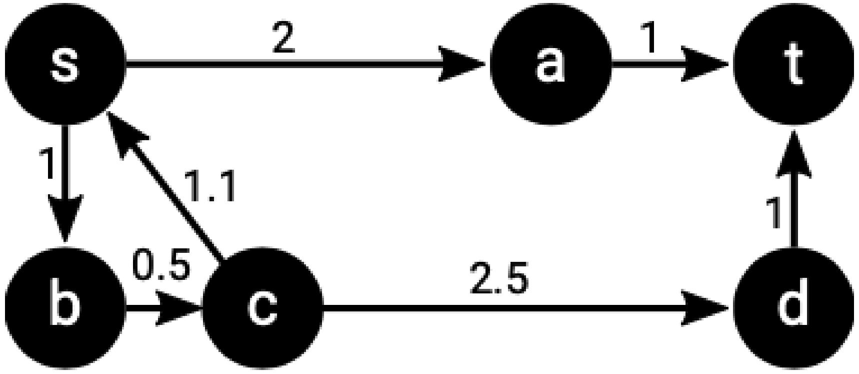
https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

Dijkstra’s Shortest Path Algorithm

Example

Algorithm 2 Dijkstra(G,s,t)

```
open ← MinHeap(), closed ← Set(), predecessors ← Dict()
open.push(s,0)
while !open.isEmpty() do
    u, uCost ← open.pop()
    if u = t then
        return extractPath(u,predecessors)
    for all v ∈ u.successors() do
        if v ∉ closed then
            uvCost ← edgeCost(u,v)
            if v ∈ open then
                if uCost + uvCost < open[v] then
                    open[v] ← uCost + uvCost
                    predecessors[v] ← u
            else
                open.push(v,uCost + uvCost)
                predecessors[v] ← u
    closed.add(u)
```



Open Min Heap:

Node	s
Cost to Vertex	0

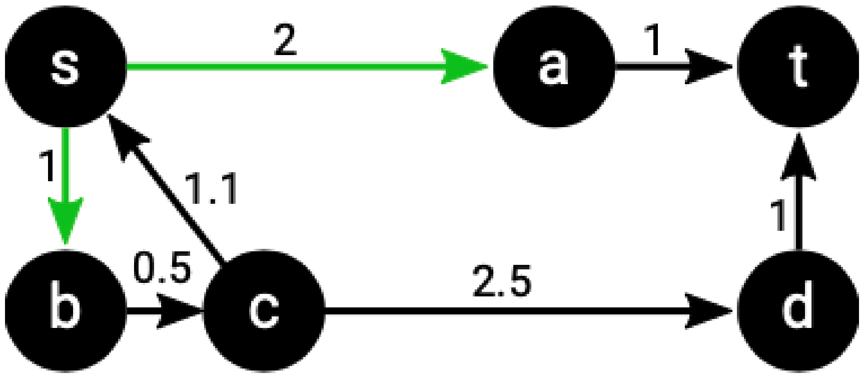
Closed Set:

Dijkstra's Shortest Path Algorithm

Example

Algorithm 2 Dijkstra(G,s,t)

```
open ← MinHeap(), closed ← Set(), predecessors ← Dict()
open.push(s,0)
while !open.isEmpty() do
    u, uCost ← open.pop()
    if u = t then
        return extractPath(u,predecessors)
    for all v ∈ u.successors() do
        if v ∉ closed then
            uvCost ← edgeCost(u,v)
            if v ∈ open then
                if uCost + uvCost < open[v] then
                    open[v] ← uCost + uvCost
                    predecessors[v] ← u
            else
                open.push(v,uCost + uvCost)
                predecessors[v] ← u
    closed.add(u)
```



Open Min Heap:

Node	b	a
Cost to Vertex	1	2

Closed Set:

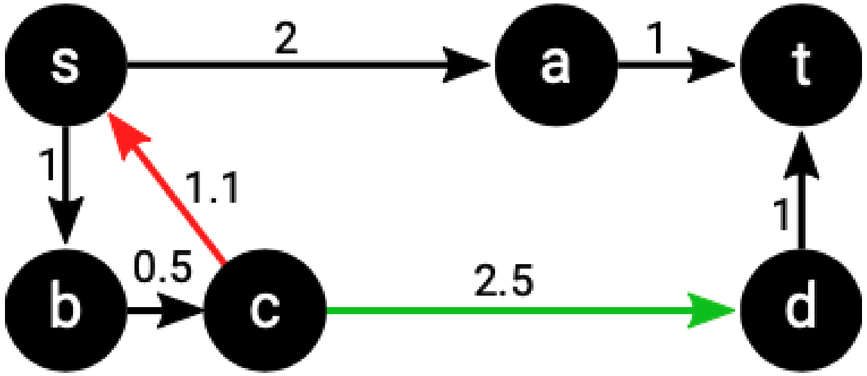
s

Dijkstra’s Shortest Path Algorithm

Example

Algorithm 2 Dijkstra(G,s,t)

```
open ← MinHeap(), closed ← Set(), predecessors ← Dict()
open.push(s,0)
while !open.isEmpty() do
    u, uCost ← open.pop()
    if u = t then
        return extractPath(u,predecessors)
    for all v ∈ u.successors() do
        if v ∉ closed then
            uvCost ← edgeCost(u,v)
            if v ∈ open then
                if uCost + uvCost < open[v] then
                    open[v] ← uCost + uvCost
                    predecessors[v] ← u
            else
                open.push(v,uCost + uvCost)
                predecessors[v] ← u
    closed.add(u)
```



Open Min Heap:

Node	a	d
Cost to Vertex	2	4

Closed Set:

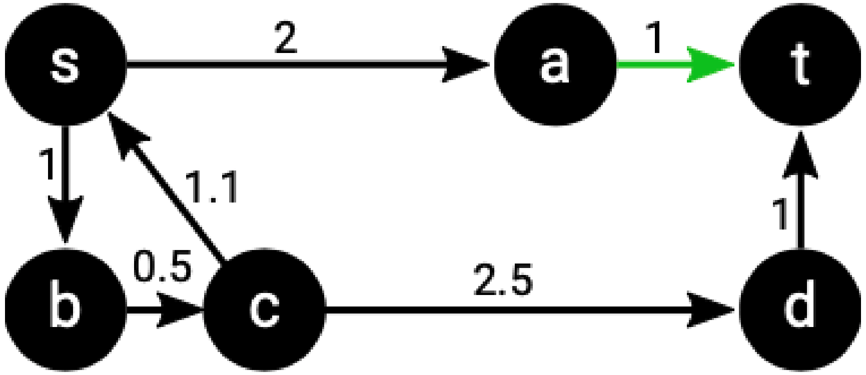
s	b	c

Dijkstra’s Shortest Path Algorithm

Example

Algorithm 2 Dijkstra(G,s,t)

```
open ← MinHeap(), closed ← Set(), predecessors ← Dict()
open.push(s,0)
while !open.isEmpty() do
    u, uCost ← open.pop()
    if u = t then
        return extractPath(u,predecessors)
    for all v ∈ u.successors() do
        if v ∉ closed then
            uvCost ← edgeCost(u,v)
            if v ∈ open then
                if uCost + uvCost < open[v] then
                    open[v] ← uCost + uvCost
                    predecessors[v] ← u
            else
                open.push(v,uCost + uvCost)
                predecessors[v] ← u
    closed.add(u)
```



Open Min Heap:

Node	t	d
Cost to Vertex	3	4

Closed Set:

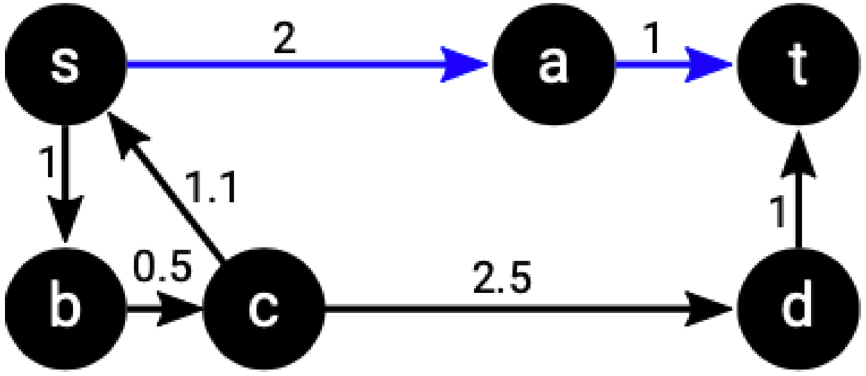
s	b	c	a

Dijkstra’s Shortest Path Algorithm

Example

Algorithm 2 Dijkstra(G,s,t)

```
open ← MinHeap(), closed ← Set(), predecessors ← Dict()
open.push(s,0)
while !open.isEmpty() do
    u, uCost ← open.pop()
    if u = t then
        return extractPath(u,predecessors)
    for all v ∈ u.successors() do
        if v ∉ closed then
            uvCost ← edgeCost(u,v)
            if v ∈ open then
                if uCost + uvCost < open[v] then
                    open[v] ← uCost + uvCost
                    predecessors[v] ← u
            else
                open.push(v,uCost + uvCost)
                predecessors[v] ← u
    closed.add(u)
```



Final Path: s a t

Closed Set: s b c a

Dijkstra’s Shortest Path Algorithm

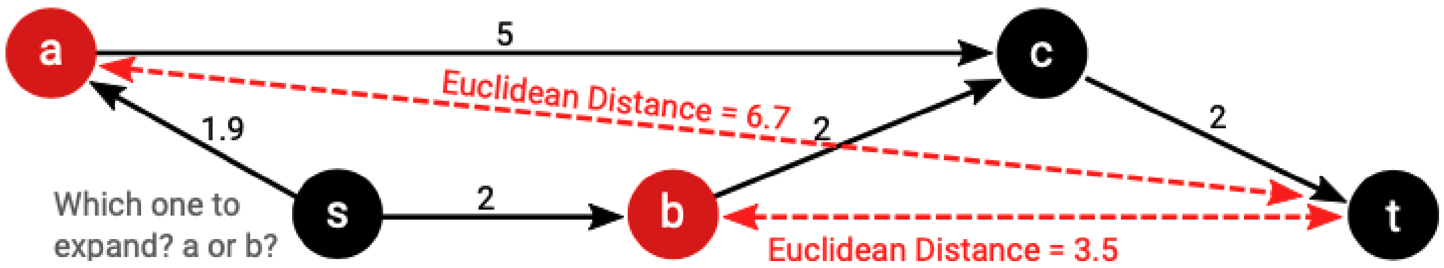
Scalability issues

→ Not scaling to very large graphs

→ Computationally expensive (slow!)

→ What can we do?

- Add a heurestic to plan more efficiently
- In particular, euclidean planning heuristics, which exploits structure of planar graphs
- Hint: the straight line between vertices is a useful estimation of the distance along the path
- A* algorithms



A* algorithm

Comparison against Dijkstra

Algorithm 2 Dijkstra(G,s,t)

```
open ← MinHeap(), closed ← Set(), predecessors ← Dict()
open.push(s,0)
while !open.isEmpty() do
    u, uCost ← open.pop()
    if u = t then
        return extractPath(u,predecessors)
    for all v ∈ u.successors() do
        if v ∉ closed then
            uvCost ← edgeCost(u,v)
            if v ∈ open then
                if uCost + uvCost < open[v] then
                    open[v] ← uCost + uvCost
                    predecessors[v] ← u
            else
                open.push(v,uCost + uvCost)
                predecessors[v] ← u
    closed.add(u)
```

Algorithm 3 A*(G,s,t)

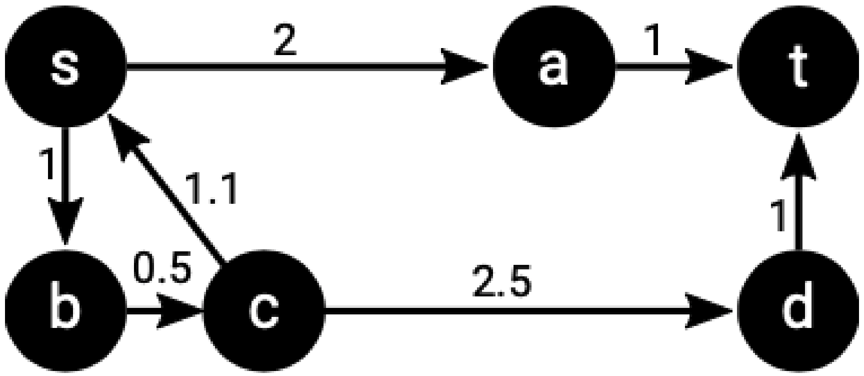
```
open ← MinHeap(), closed ← Set(), predecessors ← Dict()
open.push(s,0,h(s))
while !open.isEmpty() do
    u, uCost, uHeuristic ← open.pop()     [based on cost+heuristic]
    if u = t then
        return extractPath(u,predecessors)
    for all v ∈ u.successors() do
        if v ∉ closed then
            uvCost ← edgeCost(u,v)
            if v ∈ open then
                if uCost + uvCost + h(v) < fullCost(v) then
                    open[v] ← uCost + uvCost, h(v)
                    predecessors[v] ← u
            else
                open.push(v,uCost + uvCost, h(v))
                predecessors[v] ← u
    closed.add(u)
```

A* Algorithm

Example

Algorithm 3 A*(G,s,t)

```
open ← MinHeap(), closed ← Set(), predecessors ← Dict()
open.push(s,0,h(s))
while !open.isEmpty() do
    u, uCost, uHeuristic ← open.pop()    [based on cost+heuristic]
    if u = t then
        return extractPath(u,predecessors)
    for all v ∈ u.successors() do
        if v ∉ closed then
            uvCost ← edgeCost(u,v)
            if v ∈ open then
                if uCost + uvCost + h(v) < fullCost(v) then
                    open[v] ← uCost + uvCost, h(v)
                    predecessors[v] ← u
            else
                open.push(v,uCost + uvCost, h(v))
                predecessors[v] ← u
    closed.add(u)
```



Open Min Heap:

Node	s
Cost + Heuristic	3

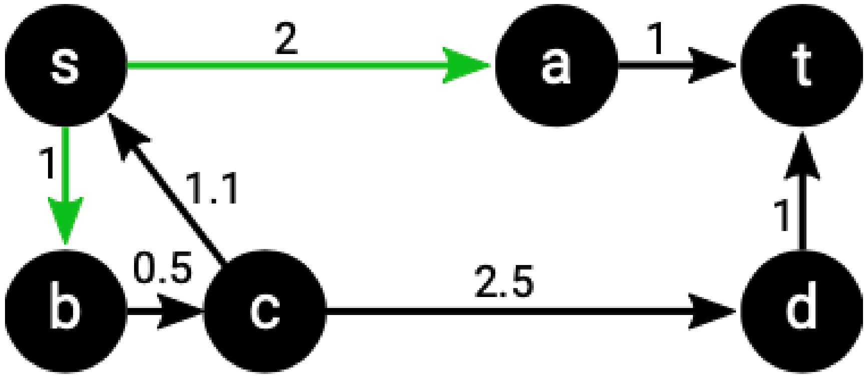
Closed Set:

A* Algorithm

Example

Algorithm 3 A*(G,s,t)

```
open ← MinHeap(), closed ← Set(), predecessors ← Dict()
open.push(s,0,h(s))
while !open.isEmpty() do
    u, uCost, uHeuristic ← open.pop()      [based on cost+heuristic]
    if u = t then
        return extractPath(u,predecessors)
    for all v ∈ u.successors() do
        if v ∉ closed then
            uvCost ← edgeCost(u,v)
            if v ∈ open then
                if uCost + uvCost + h(v) < fullCost(v) then
                    open[v] ← uCost + uvCost, h(v)
                    predecessors[v] ← u
            else
                open.push(v,uCost + uvCost, h(v))
                predecessors[v] ← u
    closed.add(u)
```



Open Min Heap:

Node	a	b
Cost + Heuristic	3	4.2

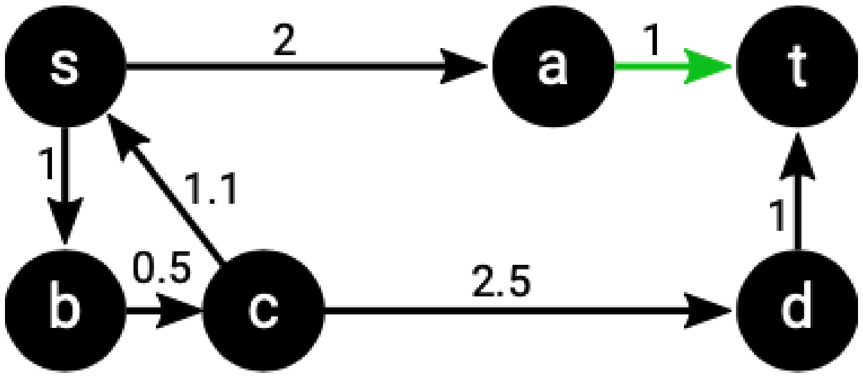
Closed Set: s

A* Algorithm

Example

Algorithm 3 A*(G,s,t)

```
open ← MinHeap(), closed ← Set(), predecessors ← Dict()
open.push(s,0,h(s))
while !open.isEmpty() do
    u, uCost, uHeuristic ← open.pop()    [based on cost+heuristic]
    if u = t then
        return extractPath(u,predecessors)
    for all v ∈ u.successors() do
        if v ∉ closed then
            uvCost ← edgeCost(u,v)
            if v ∈ open then
                if uCost + uvCost + h(v) < fullCost(v) then
                    open[v] ← uCost + uvCost, h(v)
                    predecessors[v] ← u
            else
                open.push(v,uCost + uvCost, h(v))
                predecessors[v] ← u
    closed.add(u)
```



Open Min Heap:

Node	t	b
Cost + Heuristic	3	4.2

Closed Set:

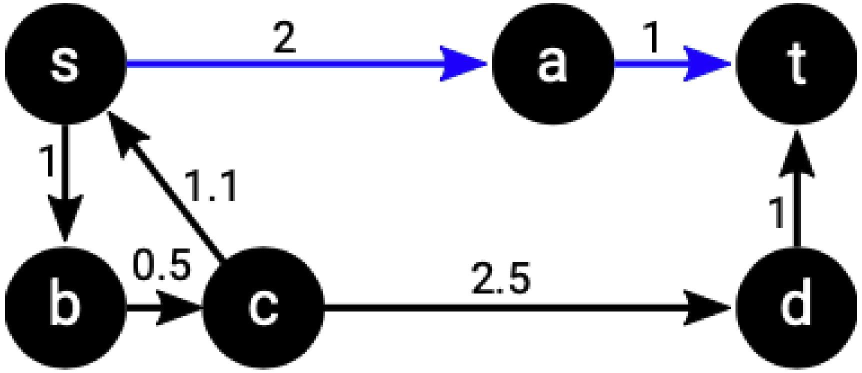
s	a

A* Algorithm

Example

Algorithm 3 A*(G,s,t)

```
open ← MinHeap(), closed ← Set(), predecessors ← Dict()
open.push(s,0,h(s))
while !open.isEmpty() do
    u, uCost, uHeuristic ← open.pop()    [based on cost+heuristic]
    if u = t then
        return extractPath(u,predecessors)
    for all v ∈ u.successors() do
        if v ∉ closed then
            uvCost ← edgeCost(u,v)
            if v ∈ open then
                if uCost + uvCost + h(v) < fullCost(v) then
                    open[v] ← uCost + uvCost, h(v)
                    predecessors[v] ← u
            else
                open.push(v,uCost + uvCost, h(v))
                predecessors[v] ← u
    closed.add(u)
```

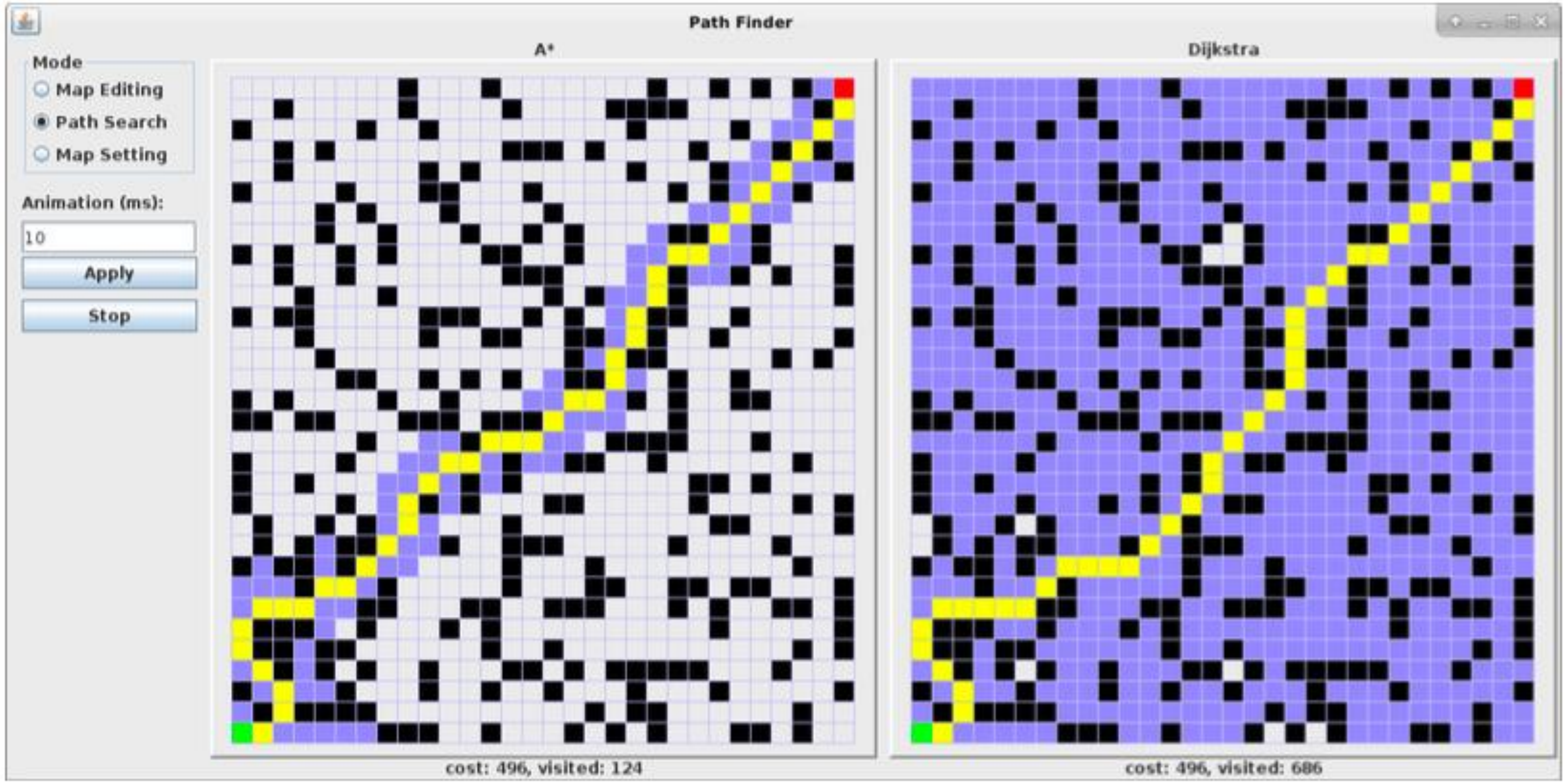


Final Path: s a t

Closed Set: s a

PathFinder: A* and Dijkstra visualization

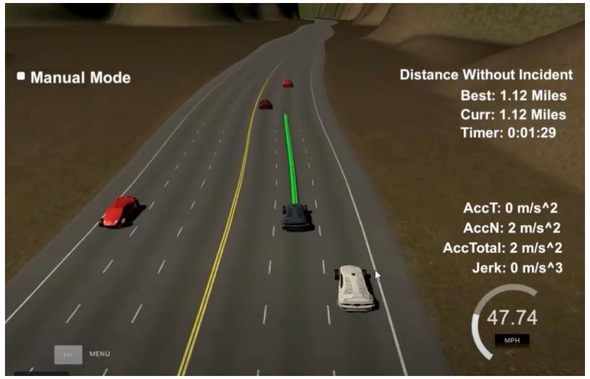
An java-based open-source program to visualize



https://github.com/kevinwang1975/PathFinder

Try it yourself!

Open-source solution provided by a student



→ Goal:

- Create smooth save path for the car to follow

→ Code:

- https://github.com/AndrewGls/CarND-Path-Planning-Project

→ Paper:

M. Werling, J. Ziegler, S. Kammel and S. Thrun, "Optimal trajectory generation for dynamic street scenarios in a Frenét Frame," *2010 IEEE International Conference on Robotics and Automation*, Anchorage, AK, USA, 2010, pp. 987-993, doi: 10.1109/ROBOT.2010.5509799.

→ Full video available at:

- https://www.youtube.com/watch?v=ihJIQOgtAWc

HEIG VD

REDS
Institut
Reconfigurable
and Embedded
Digital Systems

Hes·so
Haute Ecole Spécialisée
de Suisse occidentale