Introduction to Autonomy

Massachusetts Institute of Technology

On Robotics: High-level Broad Functional Differentiation

- **Mobile robots**
  - Wheeled/flying/ swimming robots
  - Legged robots

- **Manipulation**
  - Arms and coarse motion
  - Fingers and fine motion


Google
Google
self-driving car

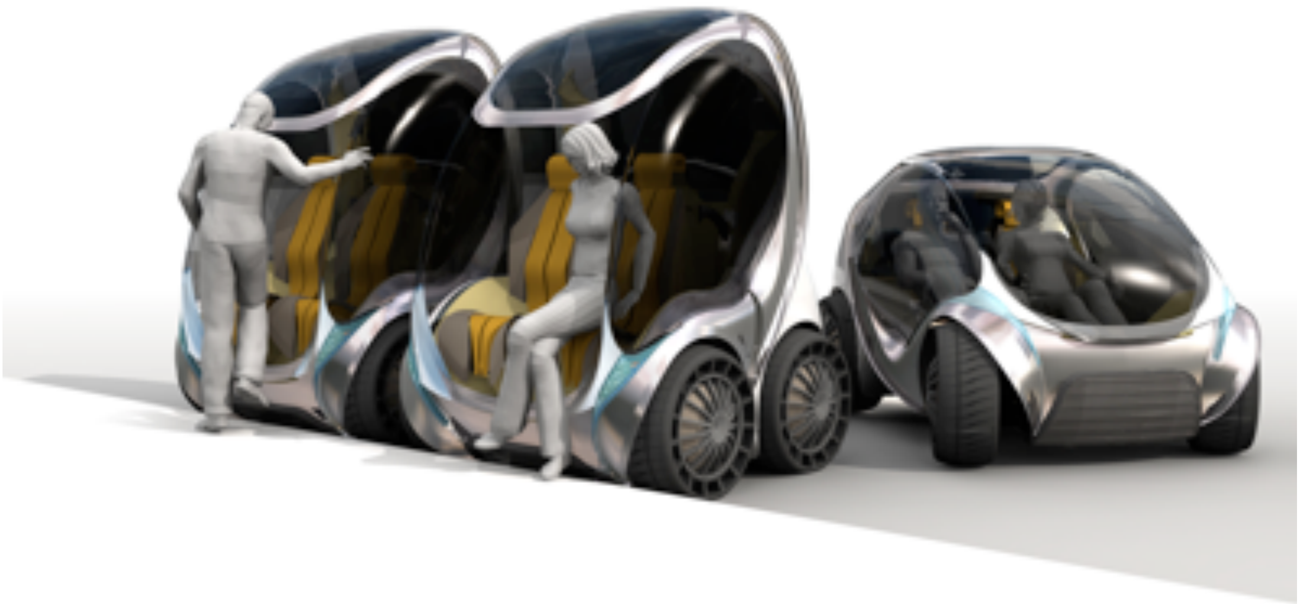
Self-driving cars

- We believe autonomous driving capabilities will play a fundamental role in future urban mobility systems:

- **Safety/comfort:** provide mobility to people who cannot, should not, or prefer not to drive

- **Efficiency/throughput:** autonomous vehicles can coordinate among themselves and with traffic control infrastructure to minimize the effects of congestion

- **Environment:** Autonomous driving can reduce emissions as much as 20-50%, and efficiently interface with smart power grids and hybrid engines



Towards full autonomy

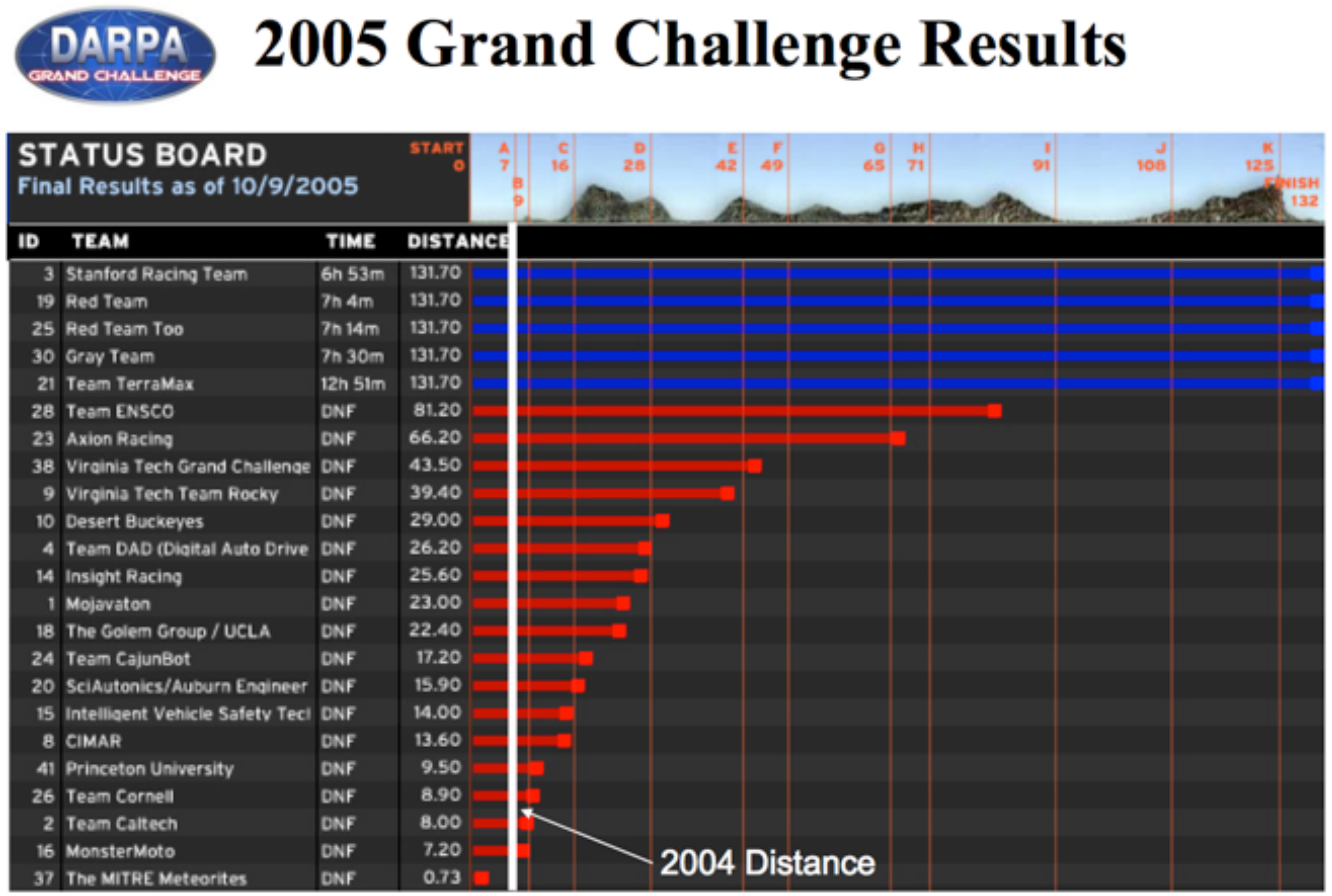
- Several projects on highway driving: Eureka project (Europe, ’87-’95), USDOT (US ’91-’97).
- US Congress mandate (’01) “one third of ground combat vehicles unmanned by 2015”
- First DARPA Grand Challenge ’04
- Second DARPA Grand Challenge ’05
- DARPA Urban Challenge ’07

DARPA Grand Challenge (March 2004)

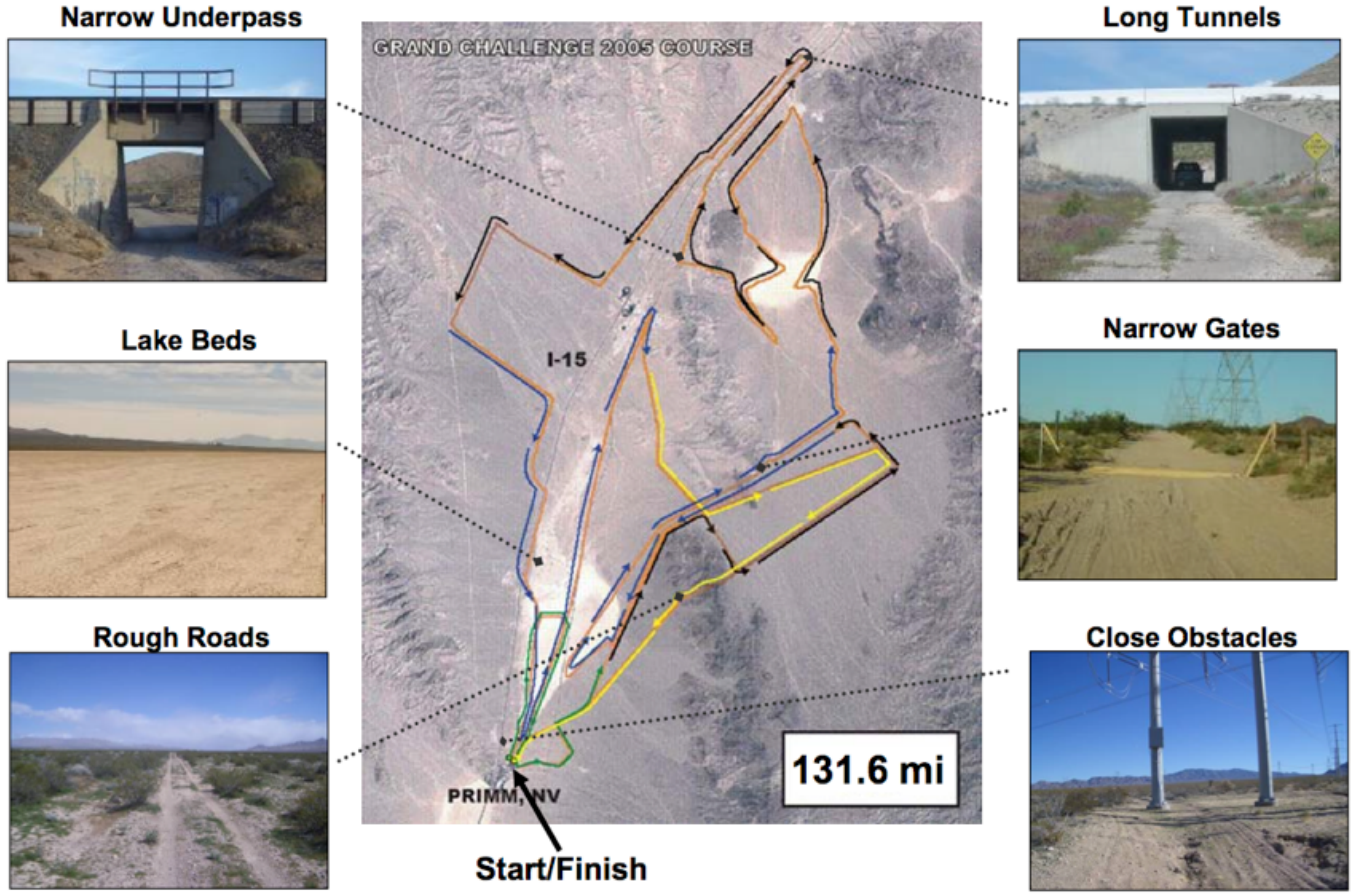
- **Mission:**
  - Drive 142 miles in less than 10 hours
  - Largely open desert and dirt roads
- **Incentives:**
  - $1M prize for the winner
- **Interest:**
  - 106 teams joined the competition.
- **Results:**
  - Within a few hours after the start, all vehicles had critical failures.
  - No vehicle went further than 7 miles.

DARPA Grand Challenge, Take 2 (October 2005)

- **Mission:** Drive 132 miles in less than 10 hours
- 195 teams participated, 5 vehicles finished, Stanford won the prize.



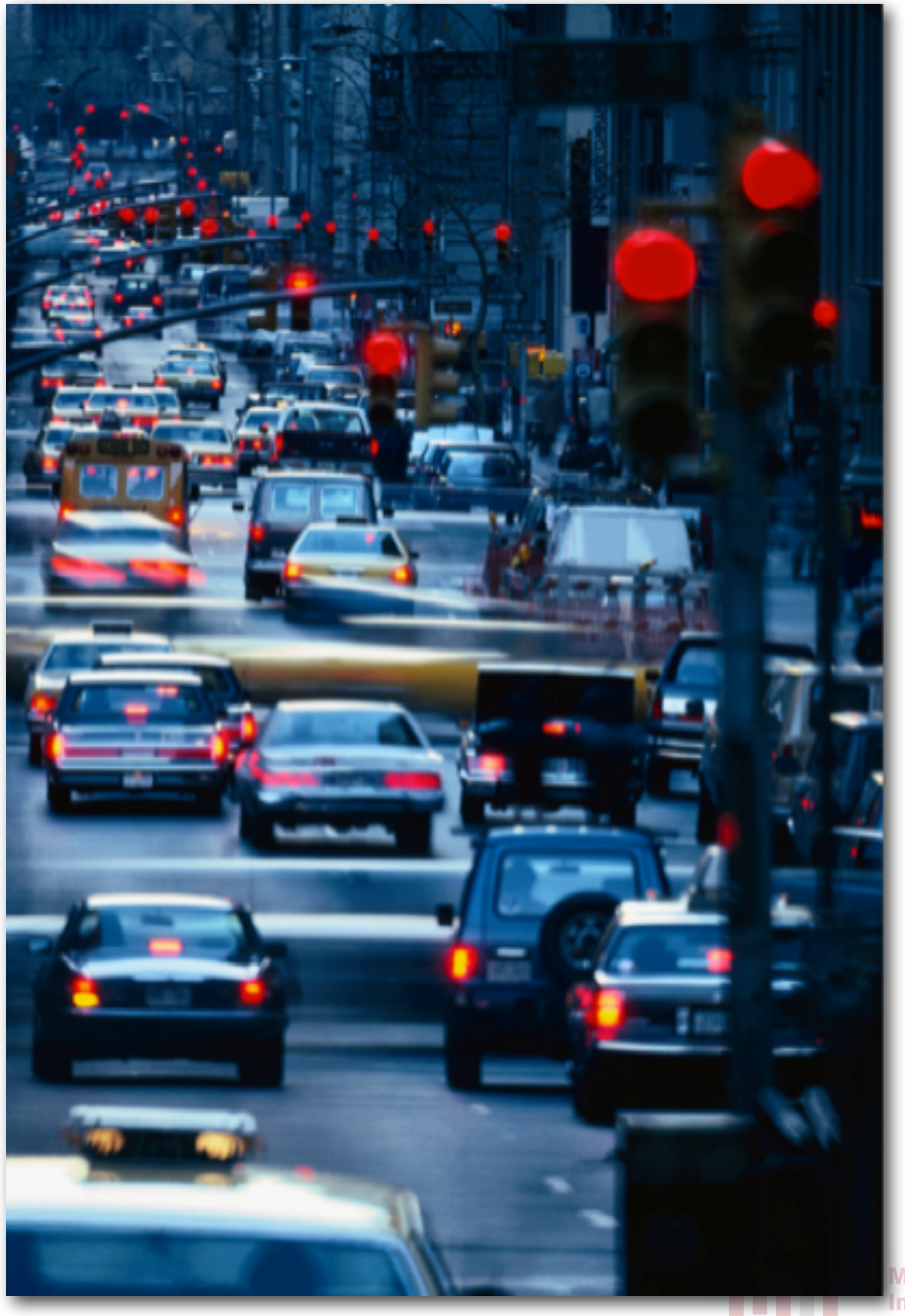
ID	TEAM	TIME	DISTANCE
3	Stanford Racing Team	6h 53m	131.70
19	Red Team	7h 4m	131.70
25	Red Team Too	7h 14m	131.70
30	Gray Team	7h 30m	131.70
21	Team TerraMax	12h 51m	131.70
28	Team ENSCO	DNF	81.20
23	Axion Racing	DNF	66.20
38	Virginia Tech Grand Challenge	DNF	43.50
9	Virginia Tech Team Rocky	DNF	39.40
10	Desert Buckeyes	DNF	29.00
4	Team DAD (Digital Auto Drive	DNF	26.20
14	Insight Racing	DNF	25.60
1	Mojavaton	DNF	23.00
18	The Golem Group / UCLA	DNF	22.40
24	Team CajunBot	DNF	17.20
20	SciAutonics/Auburn Engineer	DNF	15.90
15	Intelligent Vehicle Safety Teci	DNF	14.00
8	CIMAR	DNF	13.60
41	Princeton University	DNF	9.50
26	Team Cornell	DNF	8.90
2	Team Caltech	DNF	8.00
16	MonsterMoto	DNF	7.20
37	The MITRE Meteorites	DNF	0.73



The DARPA Urban Challenge (November 2007)

- **Urban challenge designed to be much harder than DGC I and II**
    - Urban course, with traffic (~70 vehicles)
    - 60 miles in 6 hours
    - Rules of the road (intersections, lanes, passing, merging into traffic)
    - Uncertain due to human and robotic vehicle traffic
    - Various maneuvers (parking, U-turns)
    - $2M for the winner

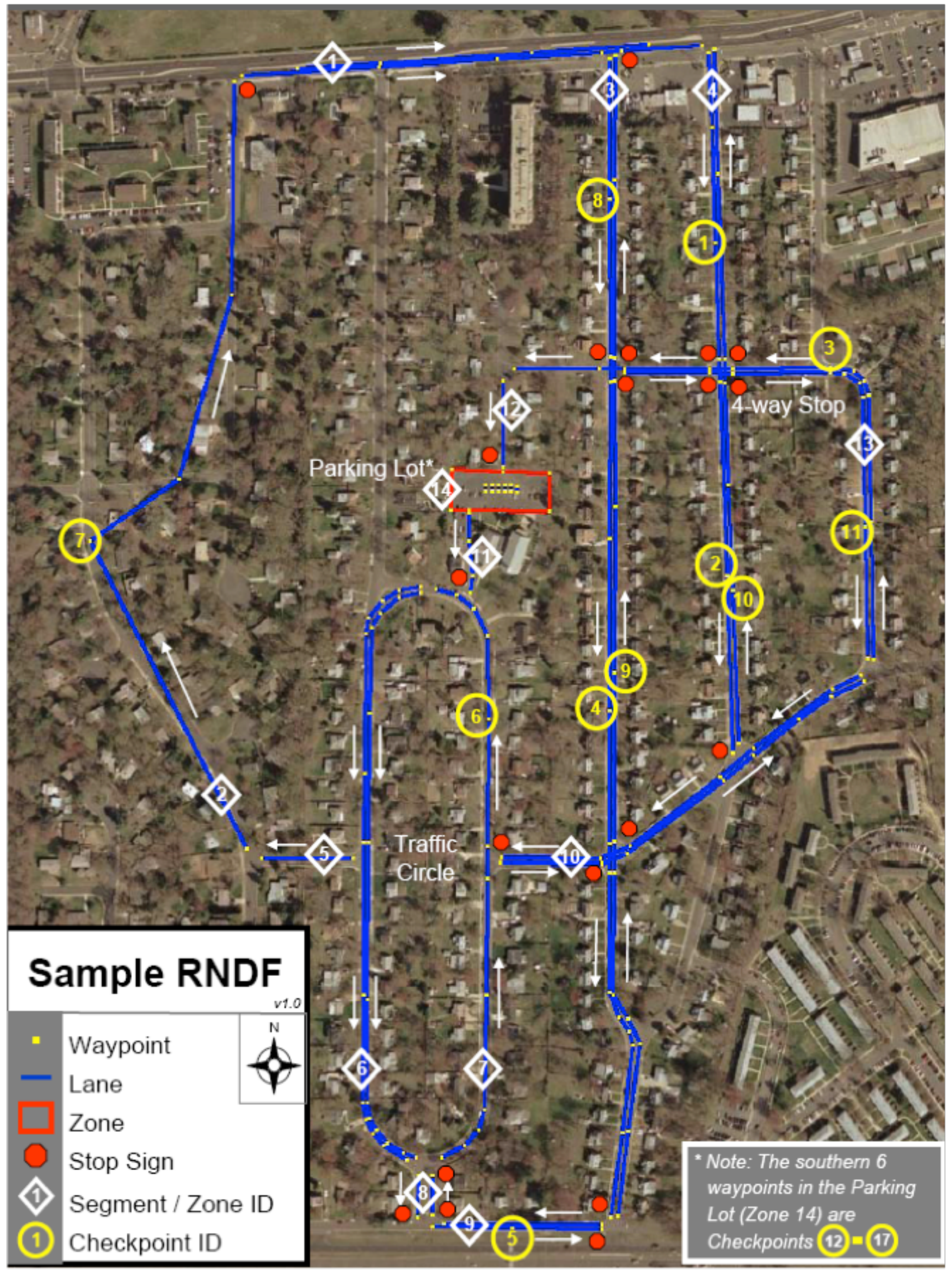
- 89 teams entered the race
- MIT's first serious entry



The Rules

- Route Network Definition File (RNDF)
  - What the road network looks like
  - Accurate, but incomplete
  - Given 24 hours before the race
- Mission Definition File (MDF)
  - Ordered waypoints to hit
  - Given 5 minutes before the race

RNDF



MDF

MIT's Team

- **MIT Faculty, postdocs, students**
  - Operating software, sensor/computer selection, configuration
  - 8 full time graduate students
- **Draper Labs**
  - System engineering, vehicle integration, test/ logistics support
- **Olin Collage of Engineering**
  - Vehicle engineering

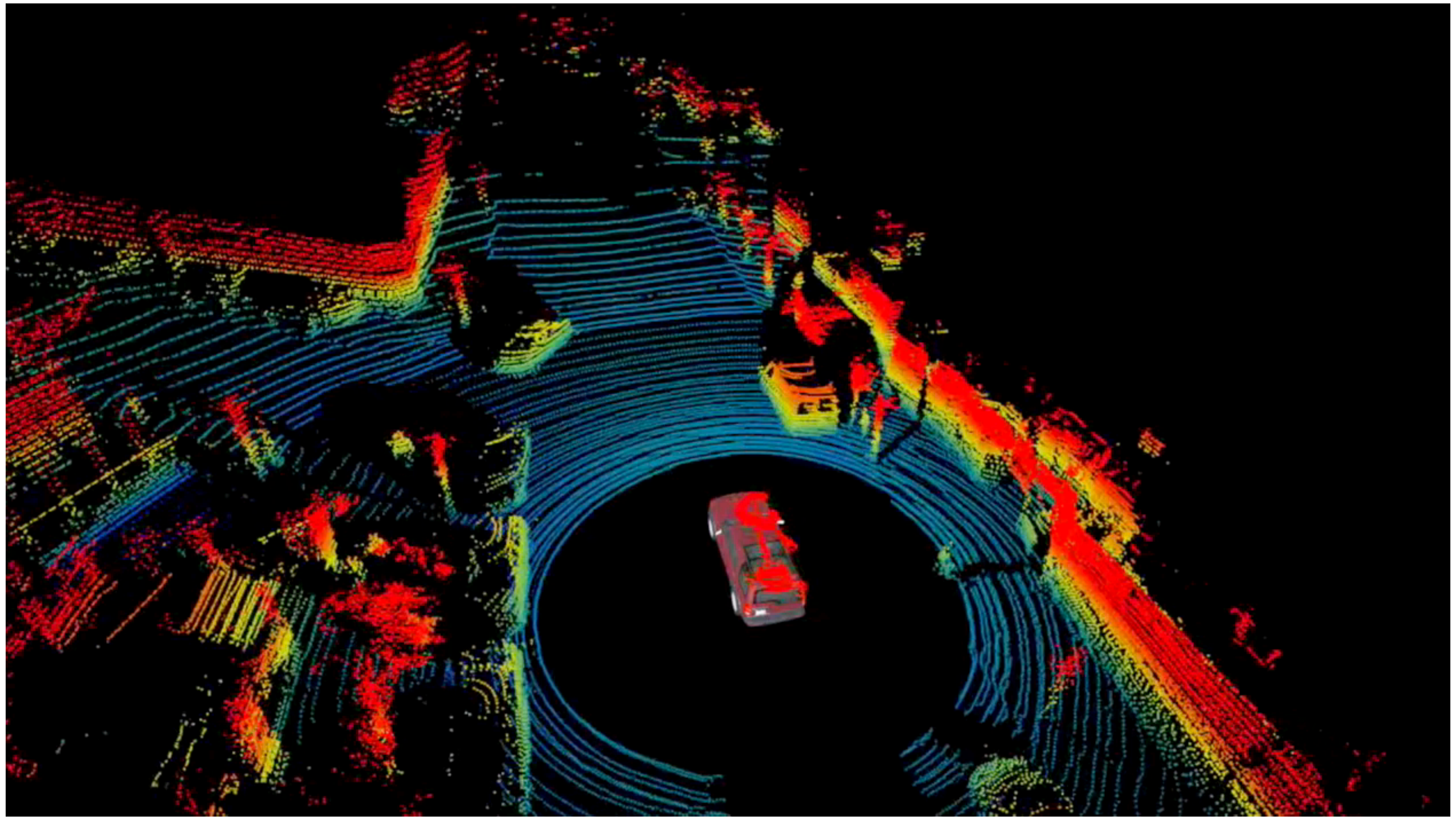
Massachusetts Institute of Technology
Quanta Computer
MITMECHE
MIT Lincoln Laboratory
MOBILEYE
tig
Acumentrics
DELPHI
Ford
IST Information Services and Technology
ADVANCED CIRCUITS
APPLANIX
NOKIA Connecting People
EECS

MIT's Vehicle

- Land Rover LR3
- EMC drive by wire
- Sensors:
  - 5 cameras
  - 16 radars
  - 12 planar laser scanners
  - 3D laser scanner
  - GPS/IMU
- Computational power:
  - 40 CPU cores
  - 40 GB RAM
- 6KW internally-mounted generator
- 2KW auxiliary air conditioner

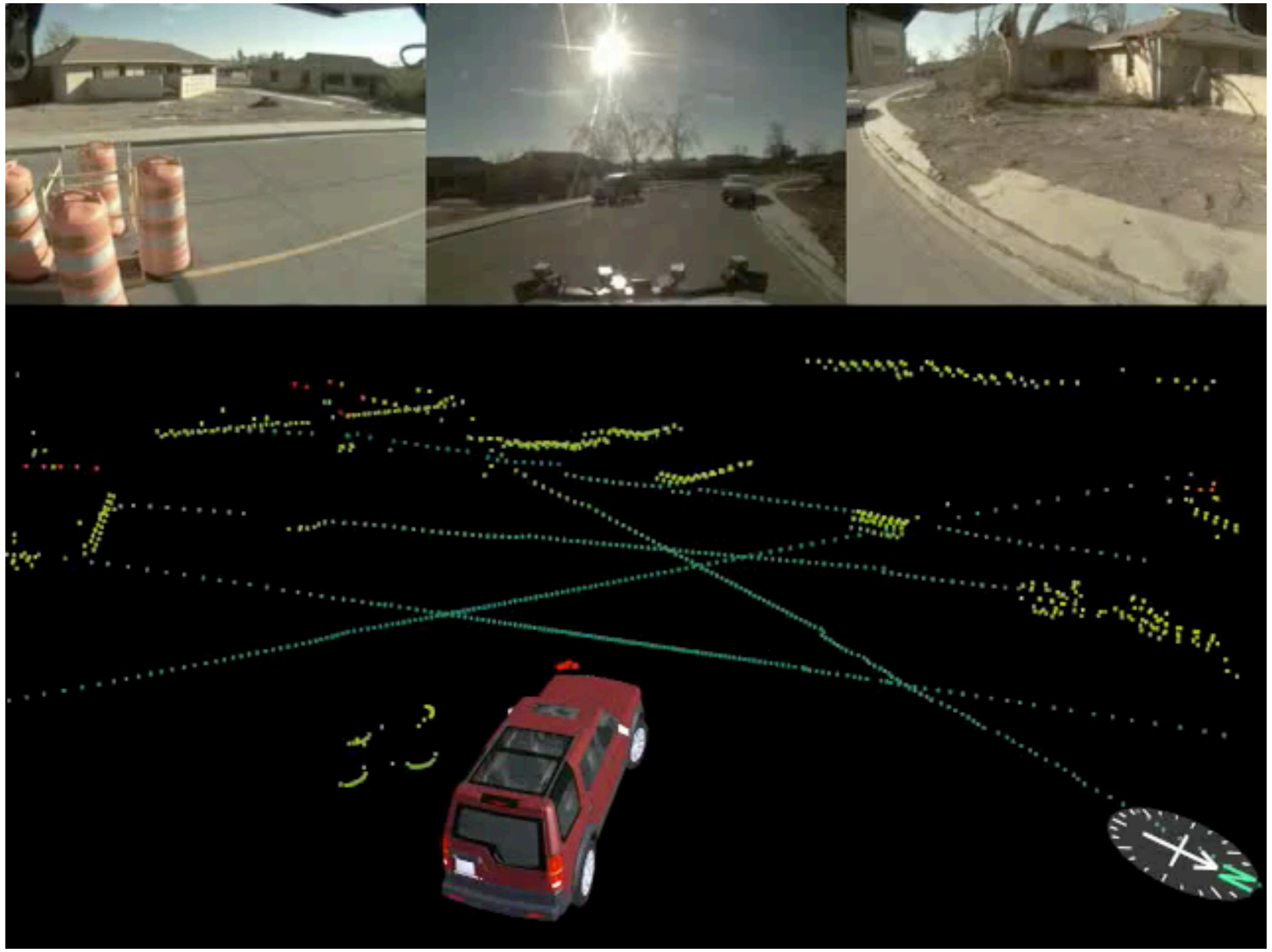
Velodyne

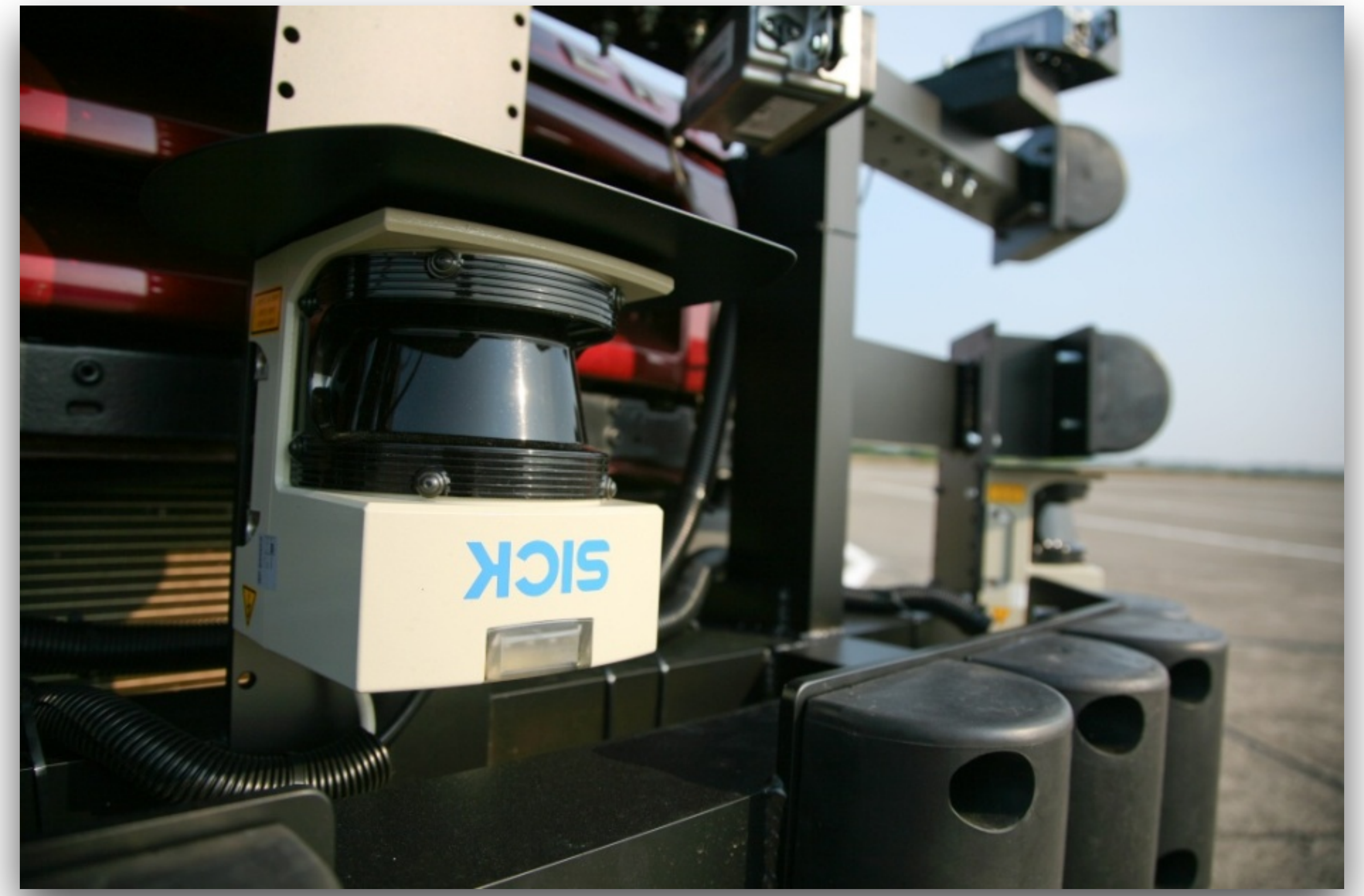
- 64 laser scanners on a vertical plane;
  rotates 15Hz to provide a 3D view.
- Main sensory equipment for all finishers.
- Used by the Google car as the primary sensor



Planar Laser Scanners

- Planar laser scan, ~50m range
- 7 on skirts (obstales), 5 on pushbrooms (ground)



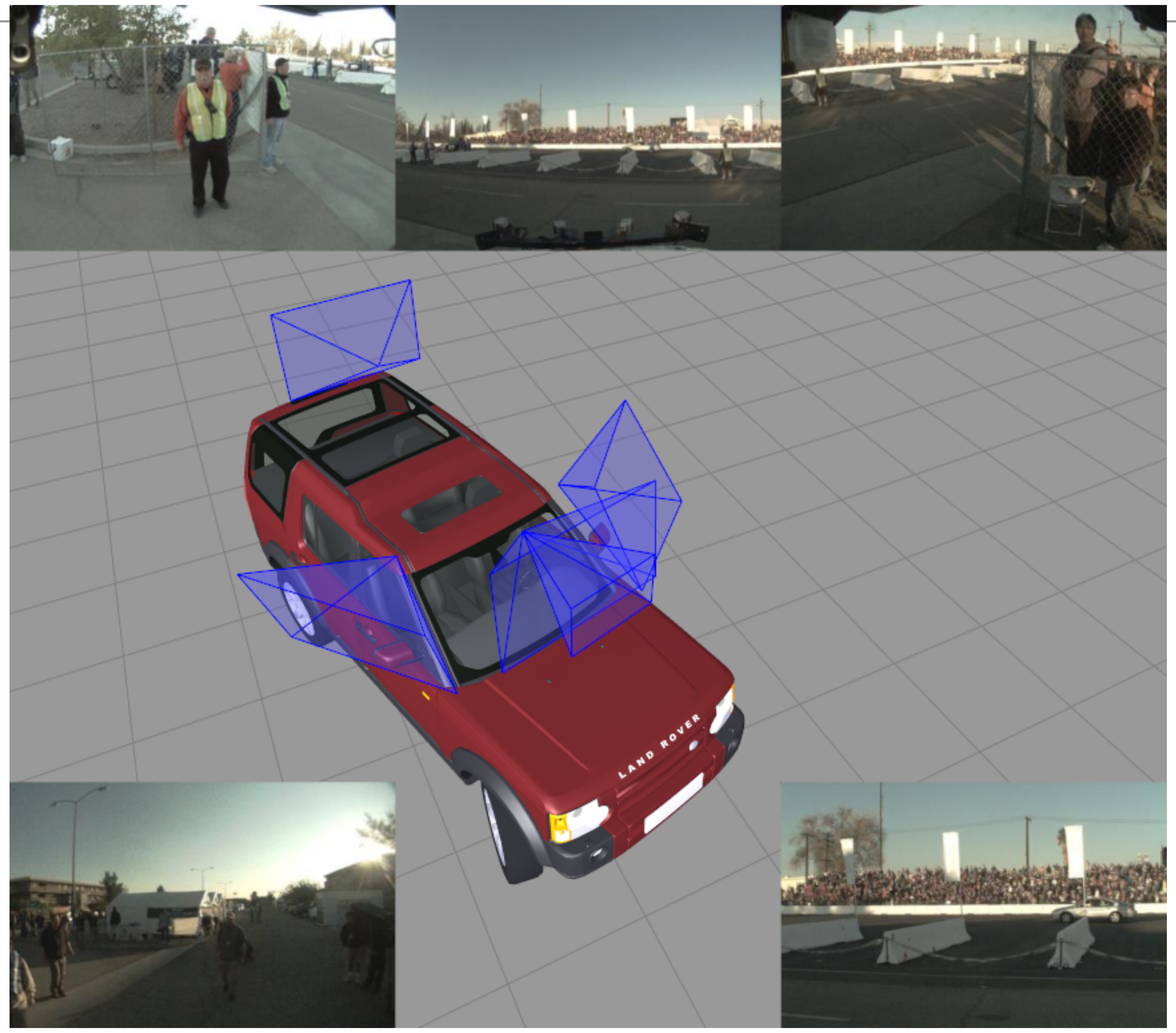


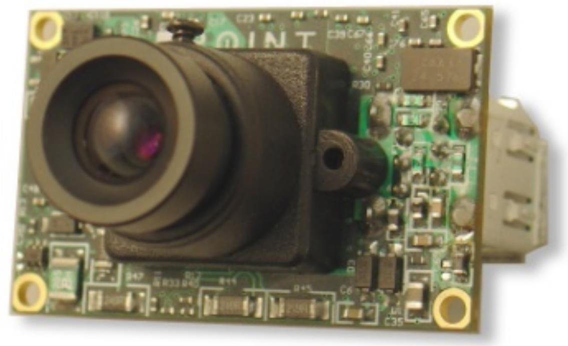
Radars

- Range, bearing, closing rate
- Narrow field of view (16 to cover 288 degrees)
- Very long range (~150m)

Cameras

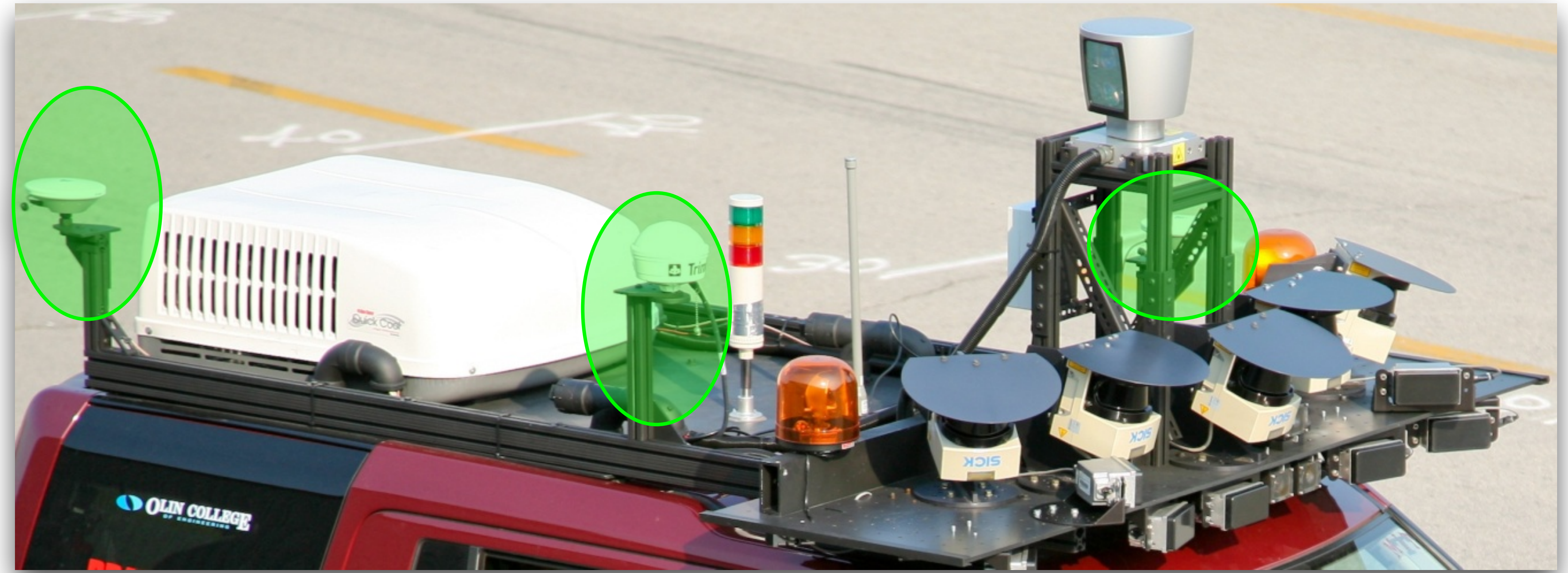
- 720x480 @ 22.8 fps
- 5 cameras for lane detection





GPS / IMU / Wheel Odometry

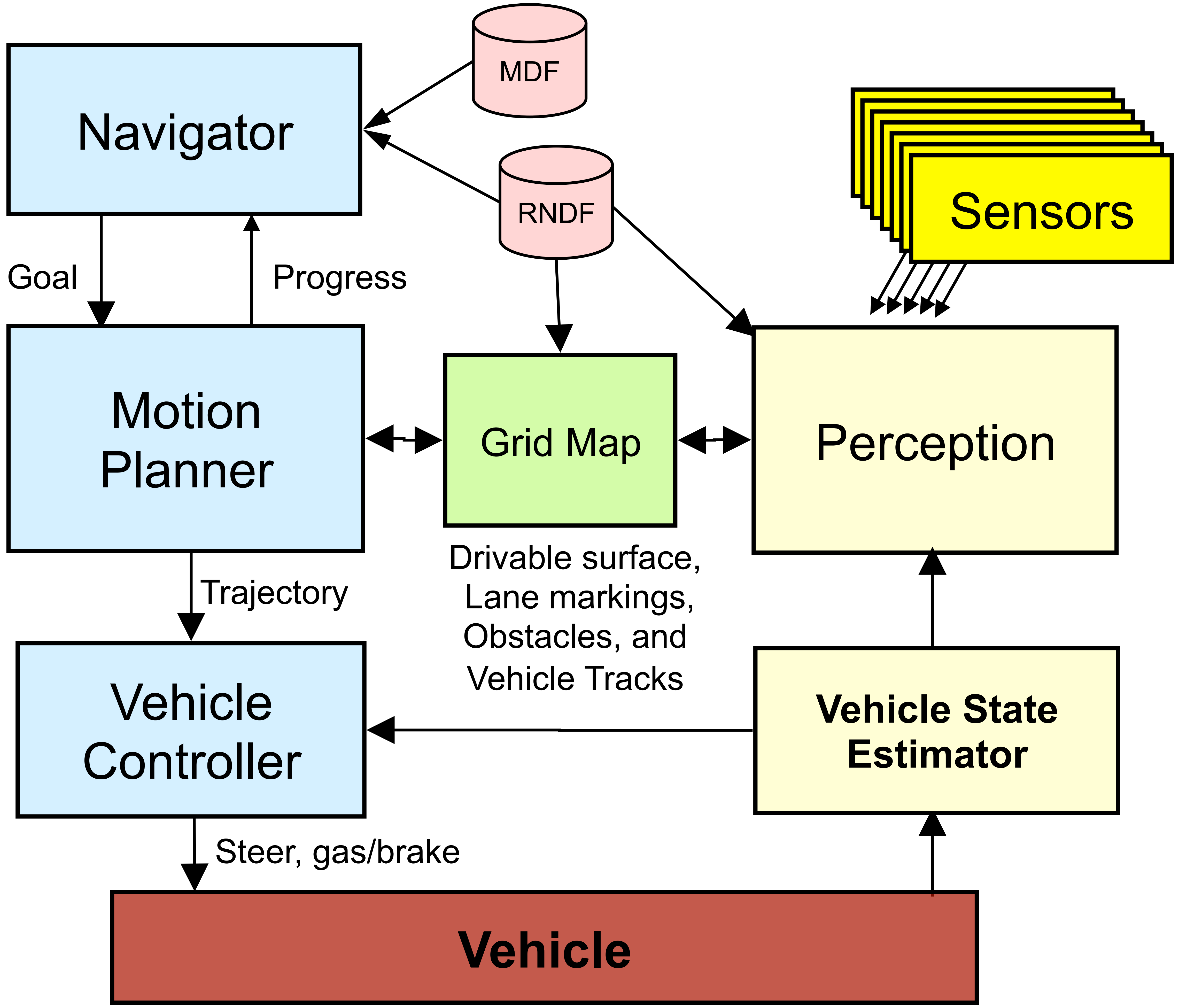
GPS

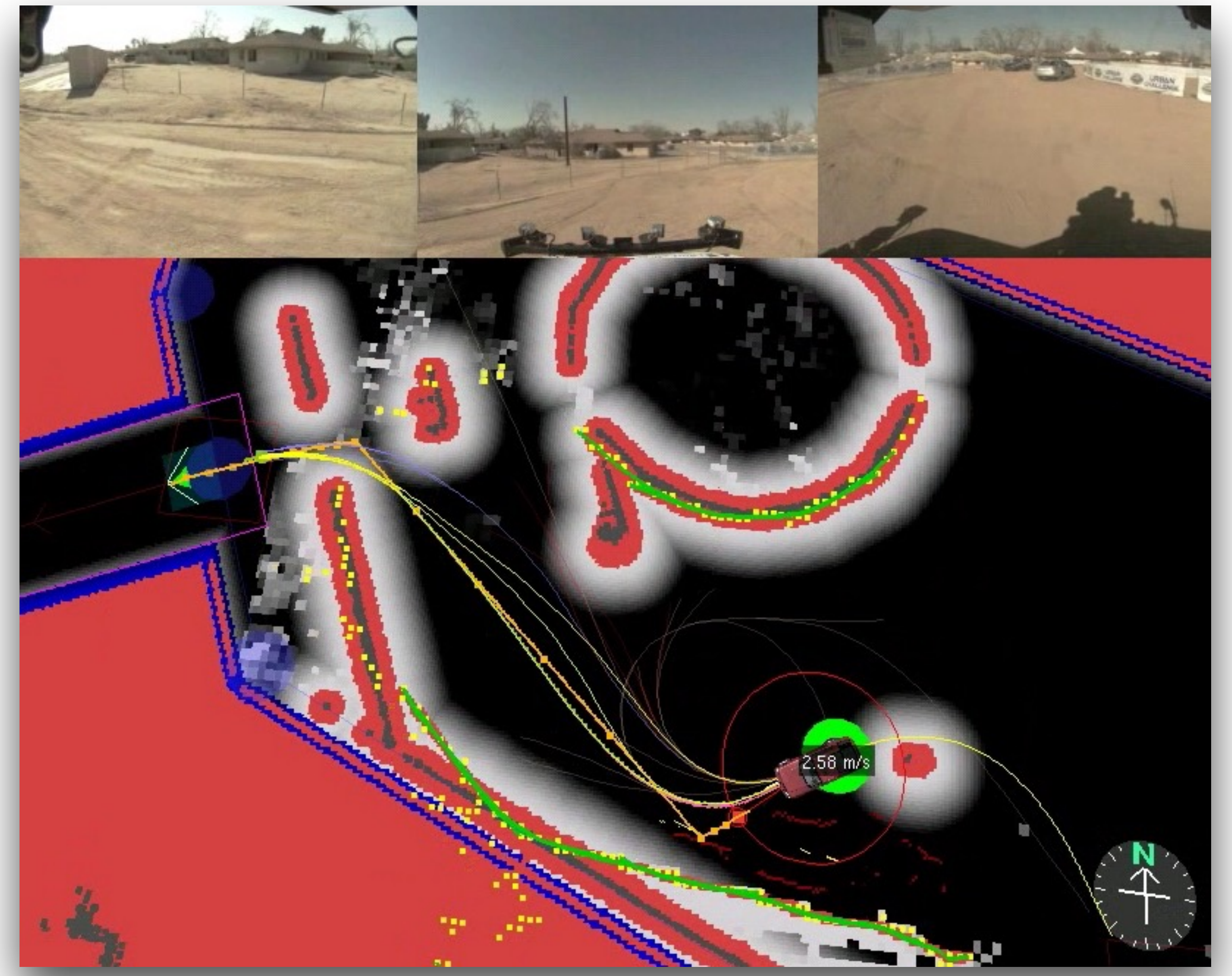


Odometry

IMU

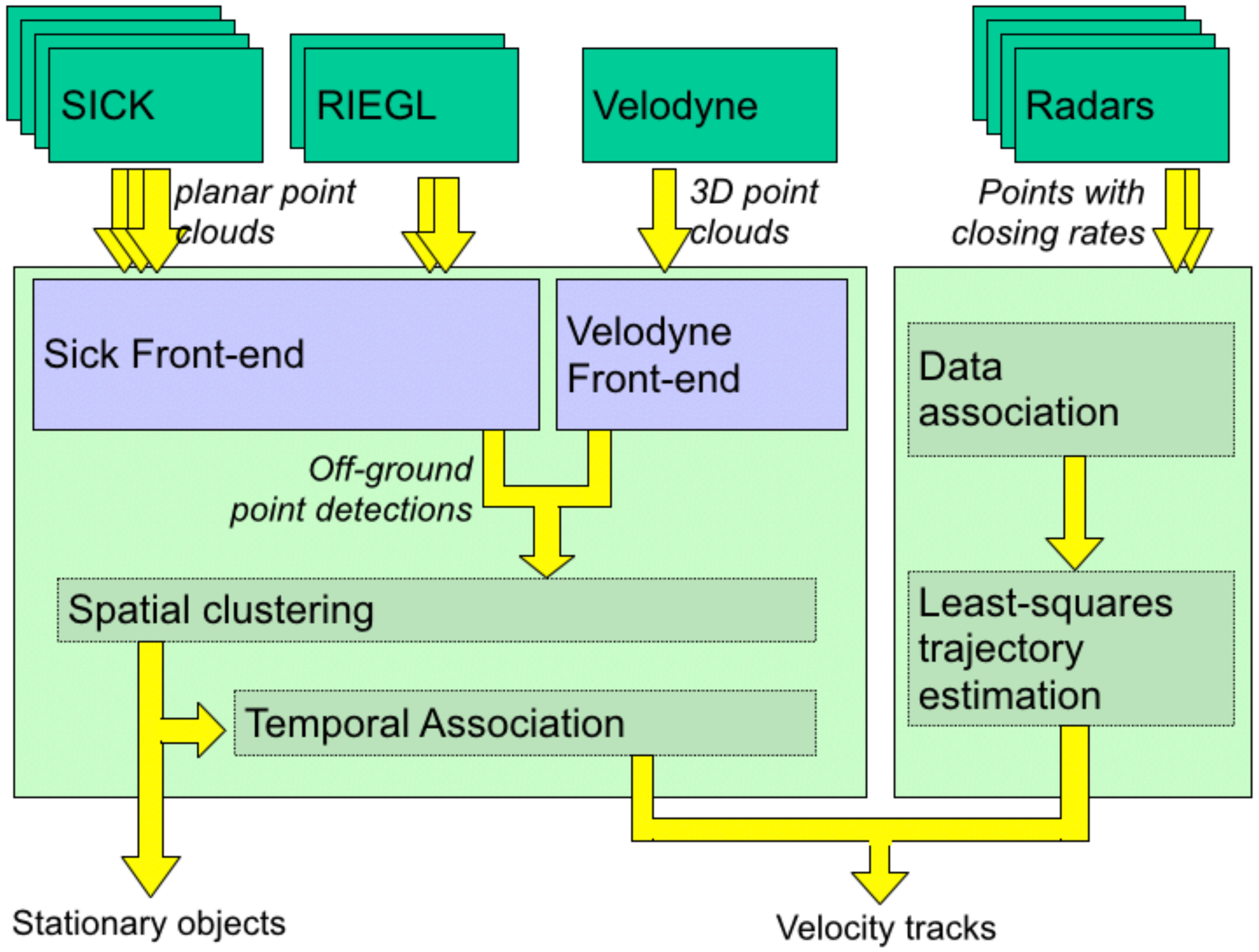
Software architecture



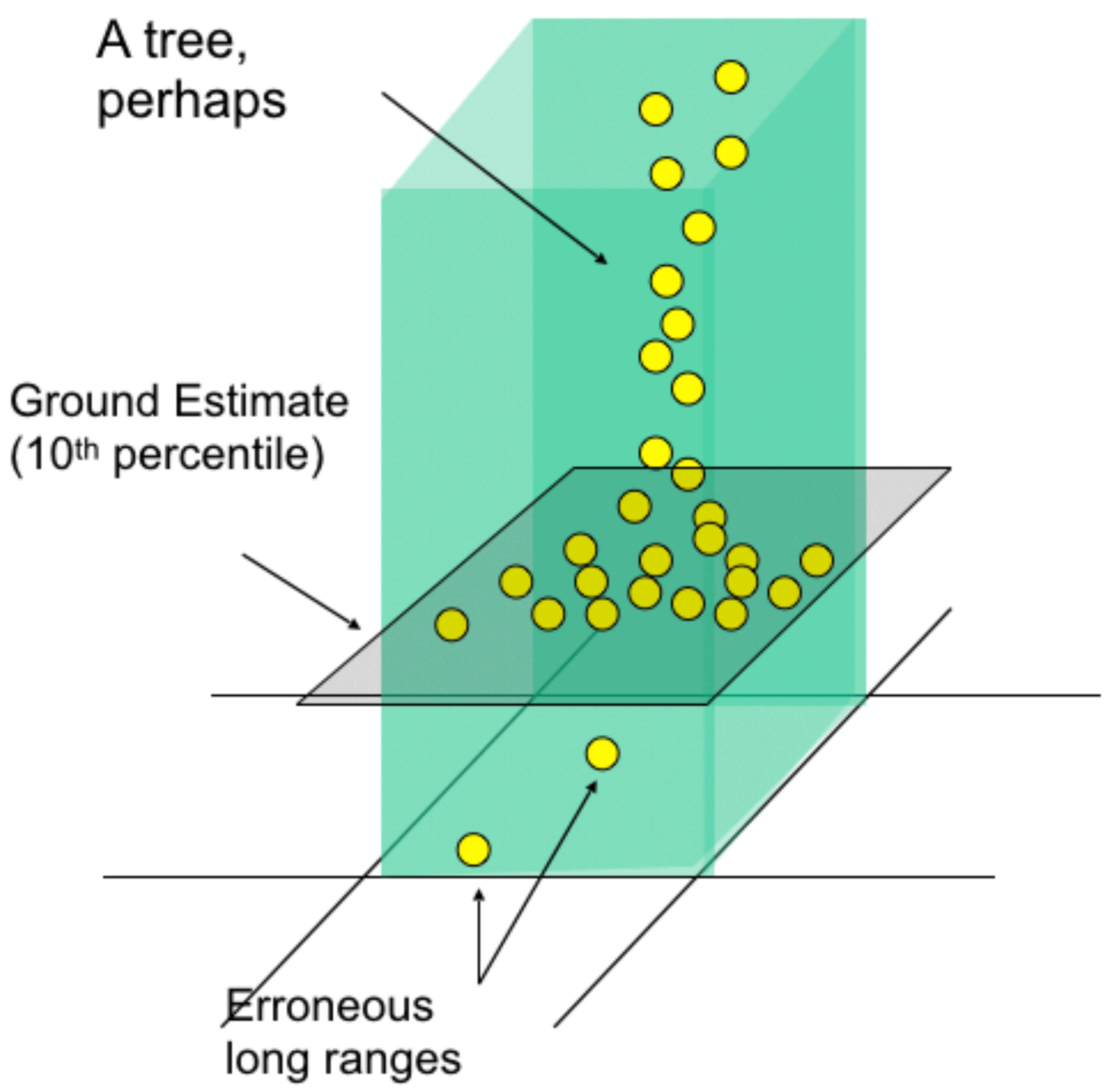


Perception systems

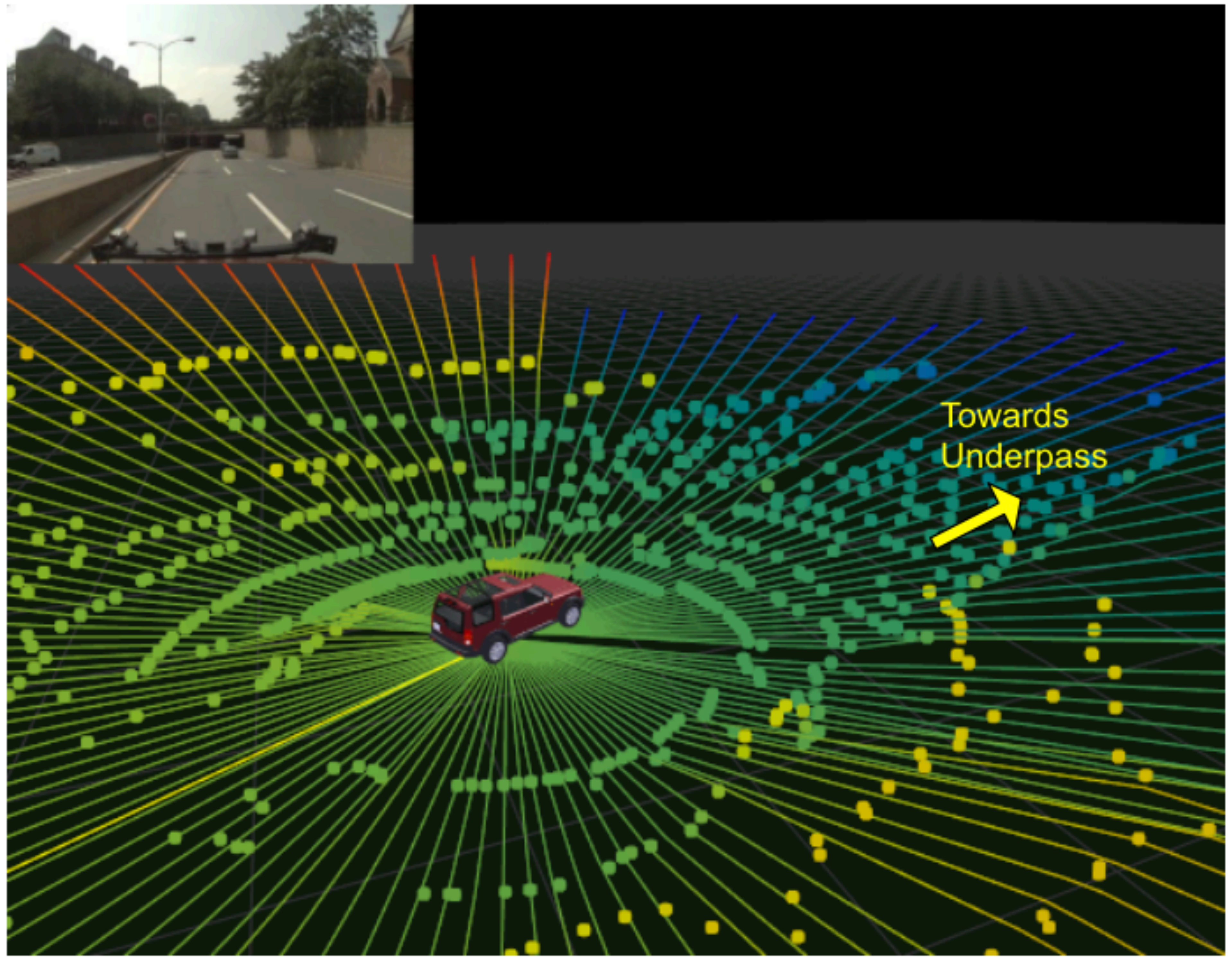
- **Obstacle Detection/Tracking**
  - Laser-based
  - Radar-based
- **Hazards and Road-Edge Detection**
  - Hazards = bad but traversable
  - Tend to appear at road-edges
- **Lane Estimation**
  - Road paint detection
  - Curve fitting
  - Lane estimation



Velodyne frontend



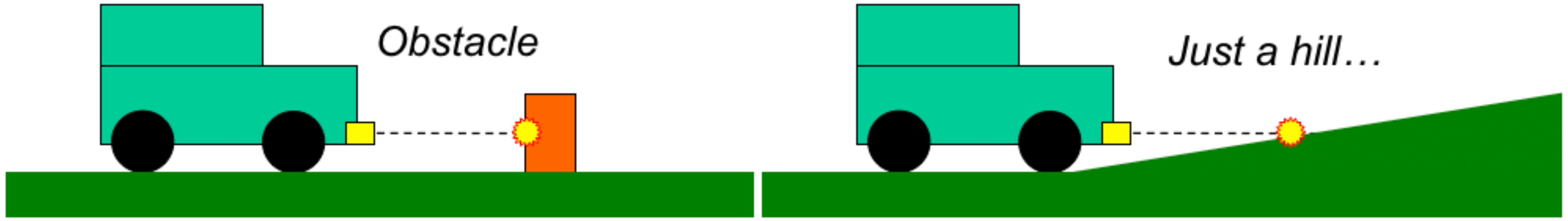
One Grid Cell



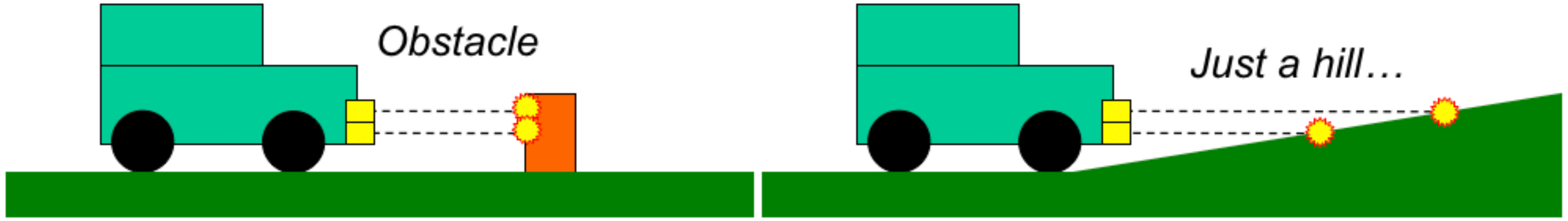
Interpolated ground detections, colored by height

Planar LiDAR front end

- *Given planar scans, extract those that are not the ground.*
- Problem: Obstacles and hills look the same



- Solution: Use multiple scanners at different heights
  - Implemented using occupancy grid with sensor ids

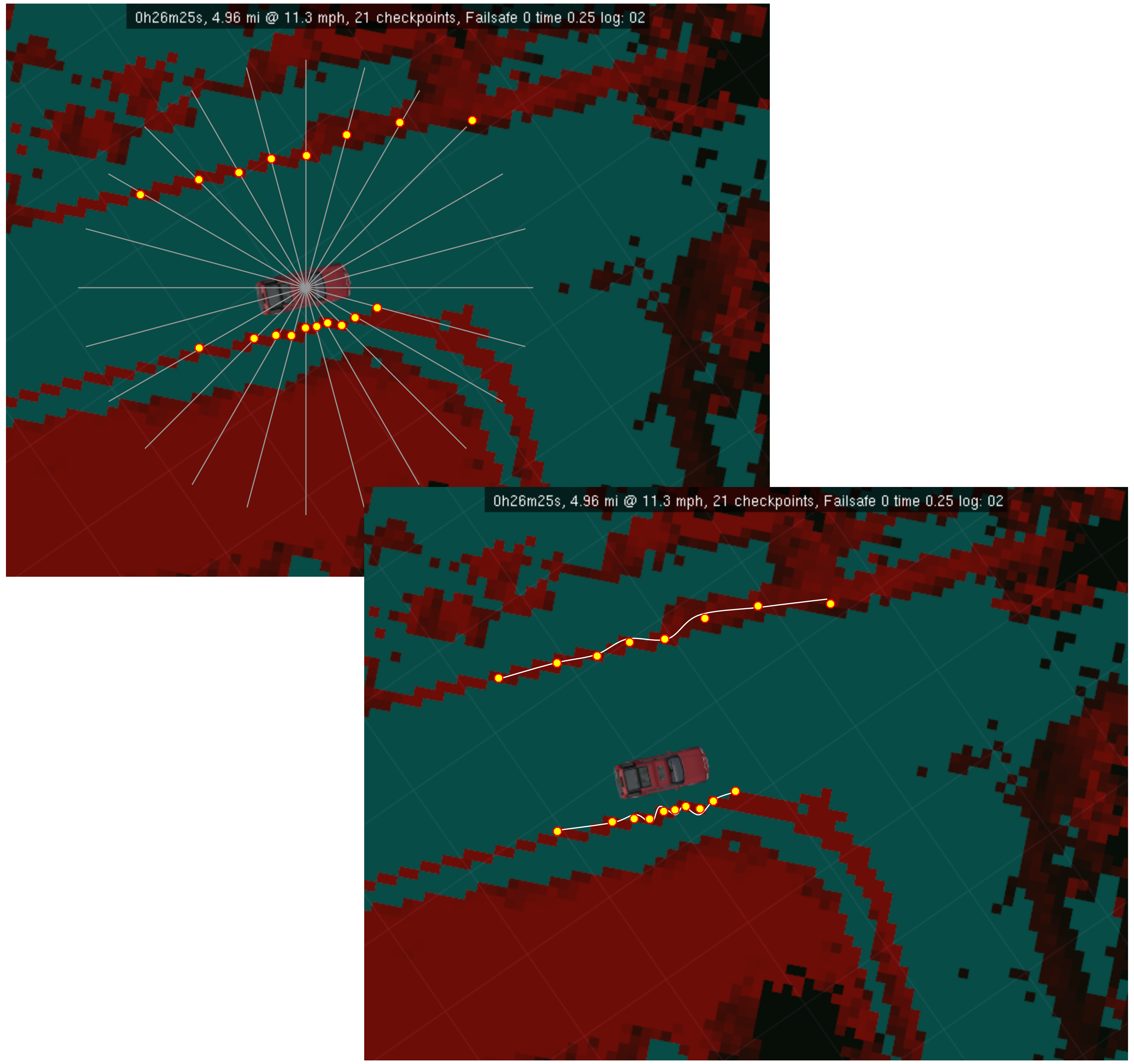


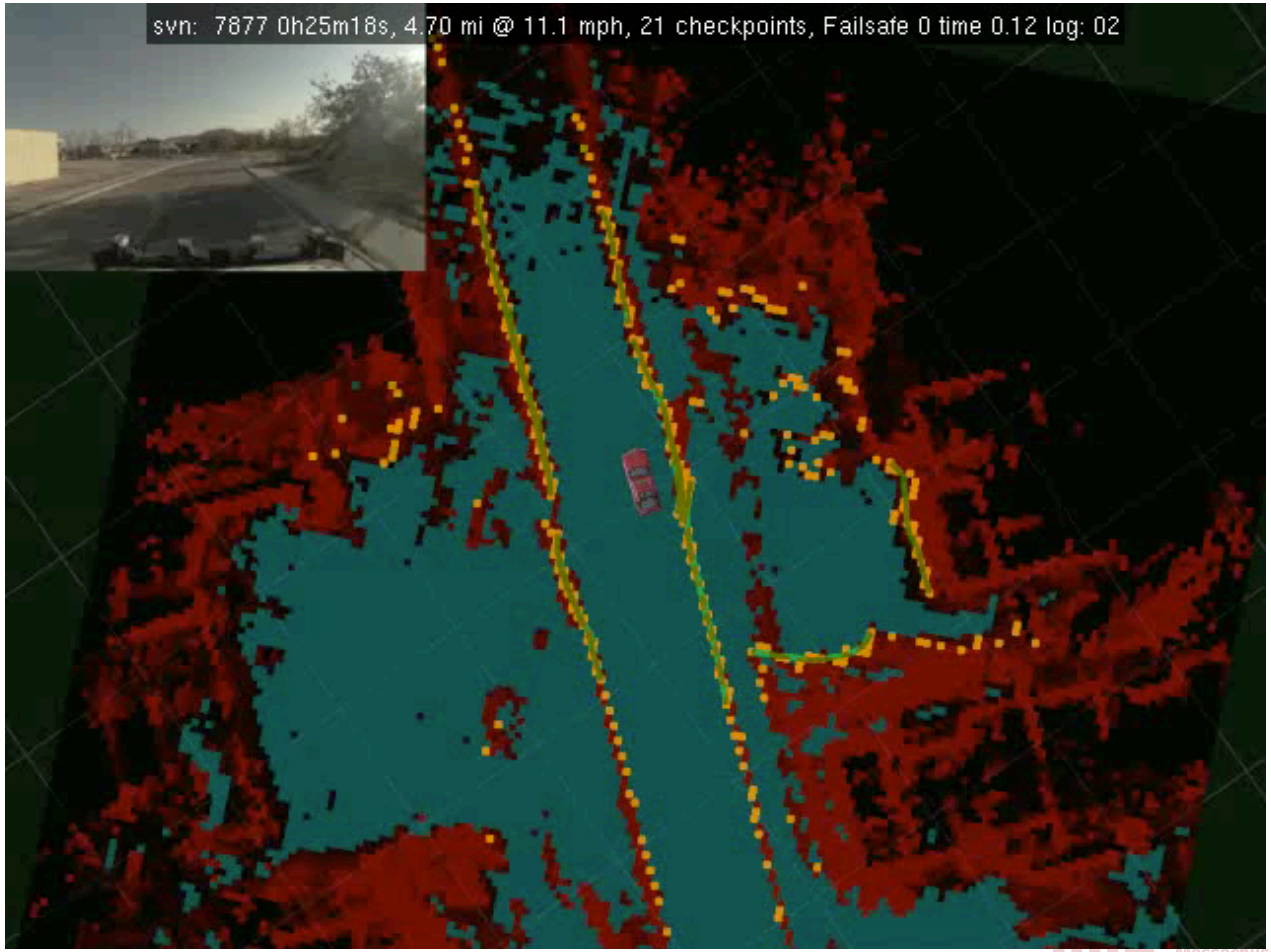
Sensors agree ➔ sharp vertical feature

Sensors disagree ➔ not an obstacle

Finding road edges

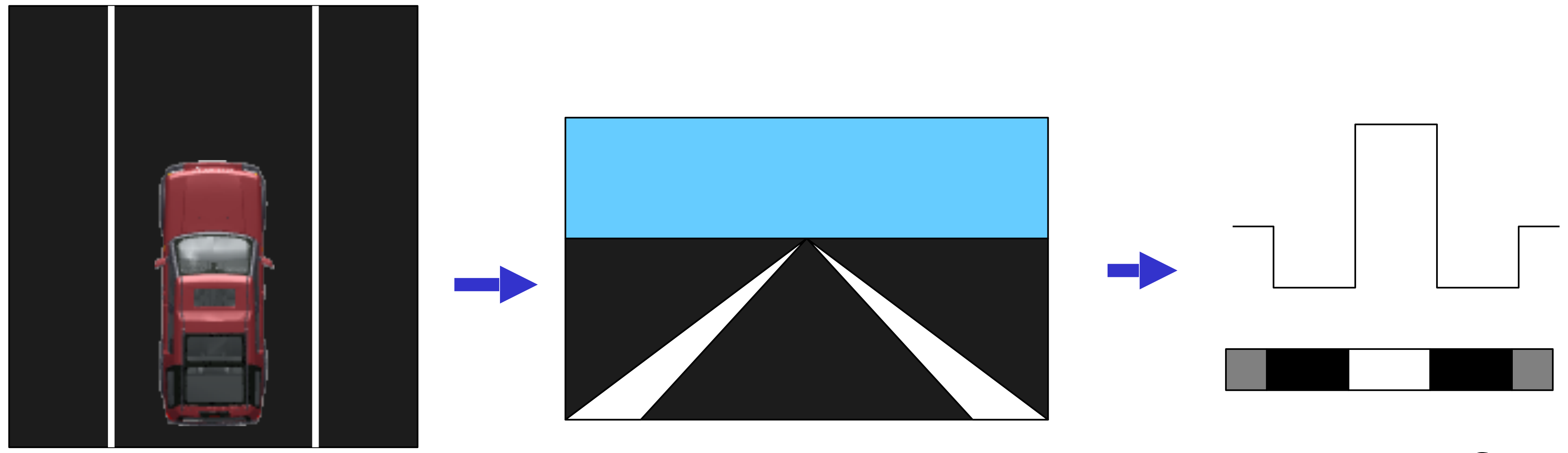
- Opted for simple algorithms:





Finding the lanes

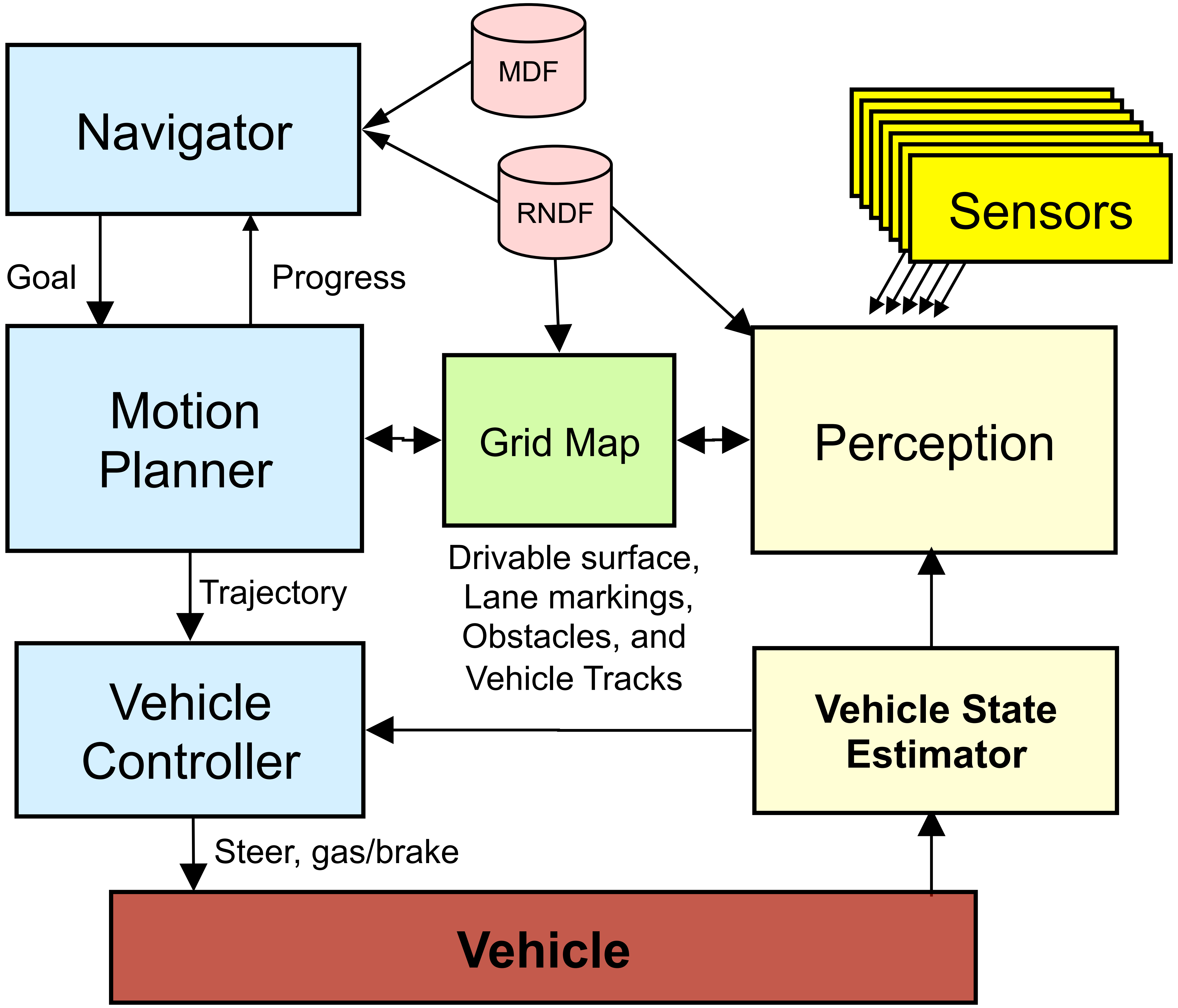
- Used computer vision

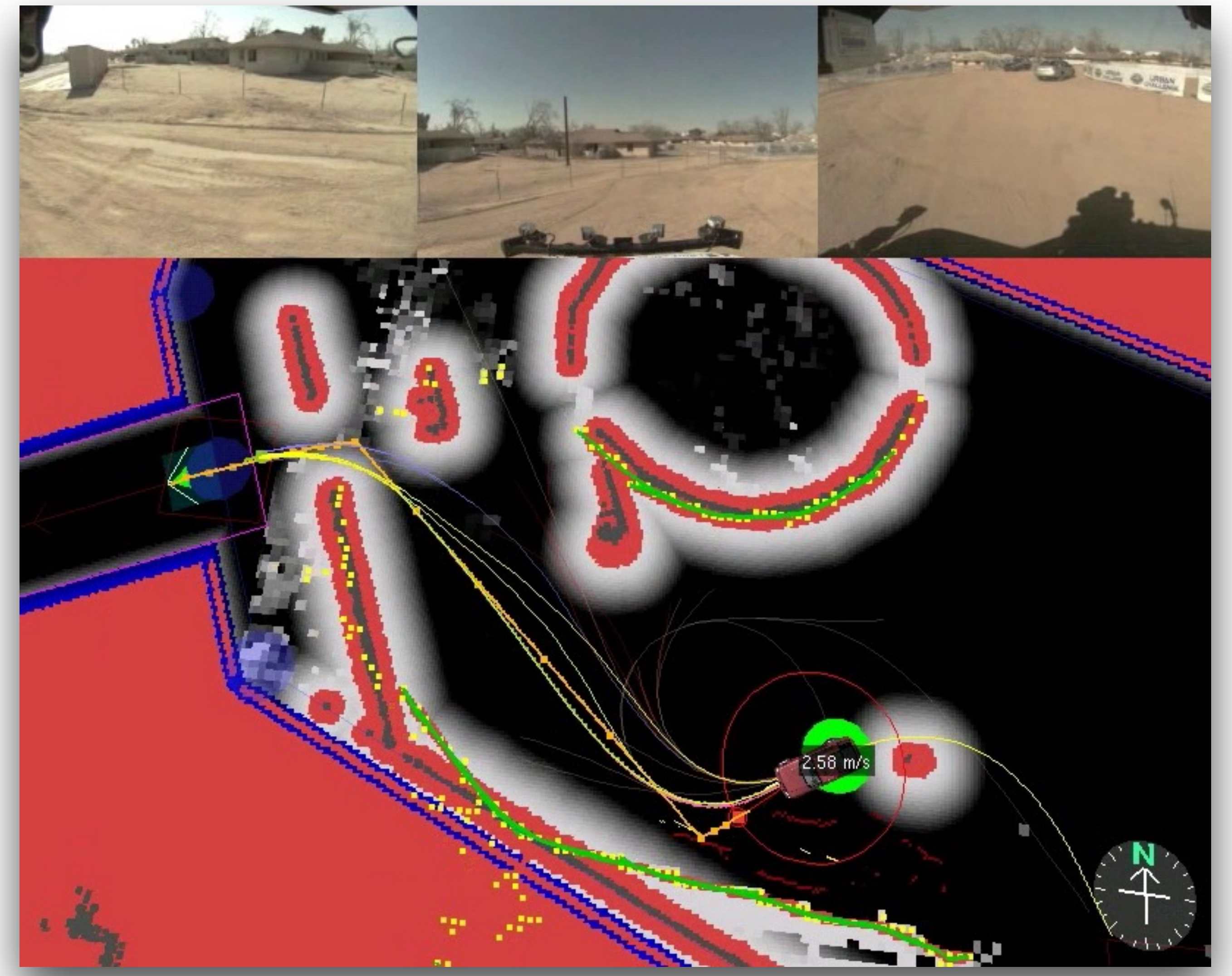


Horizontal filter

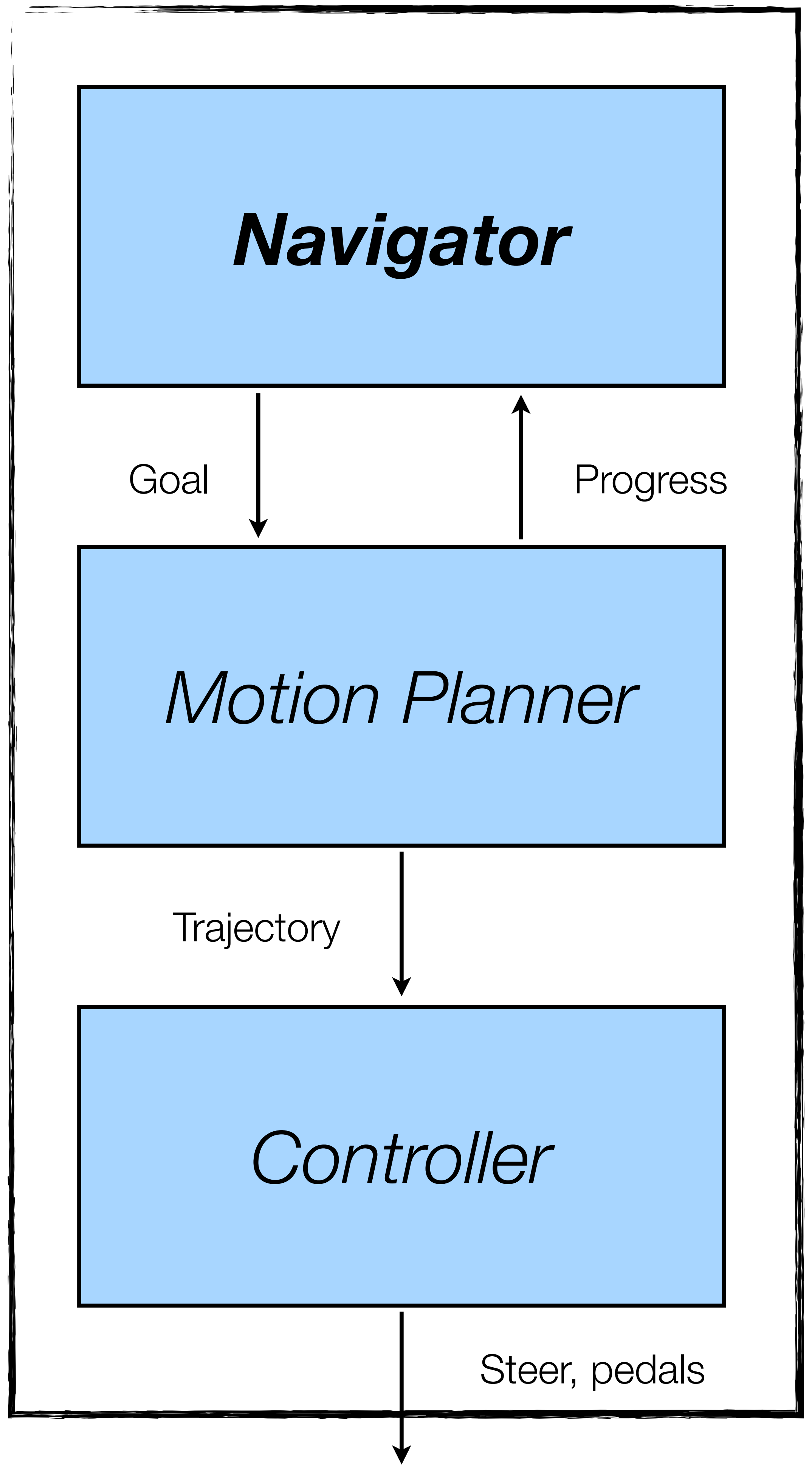

RUN

Software architecture

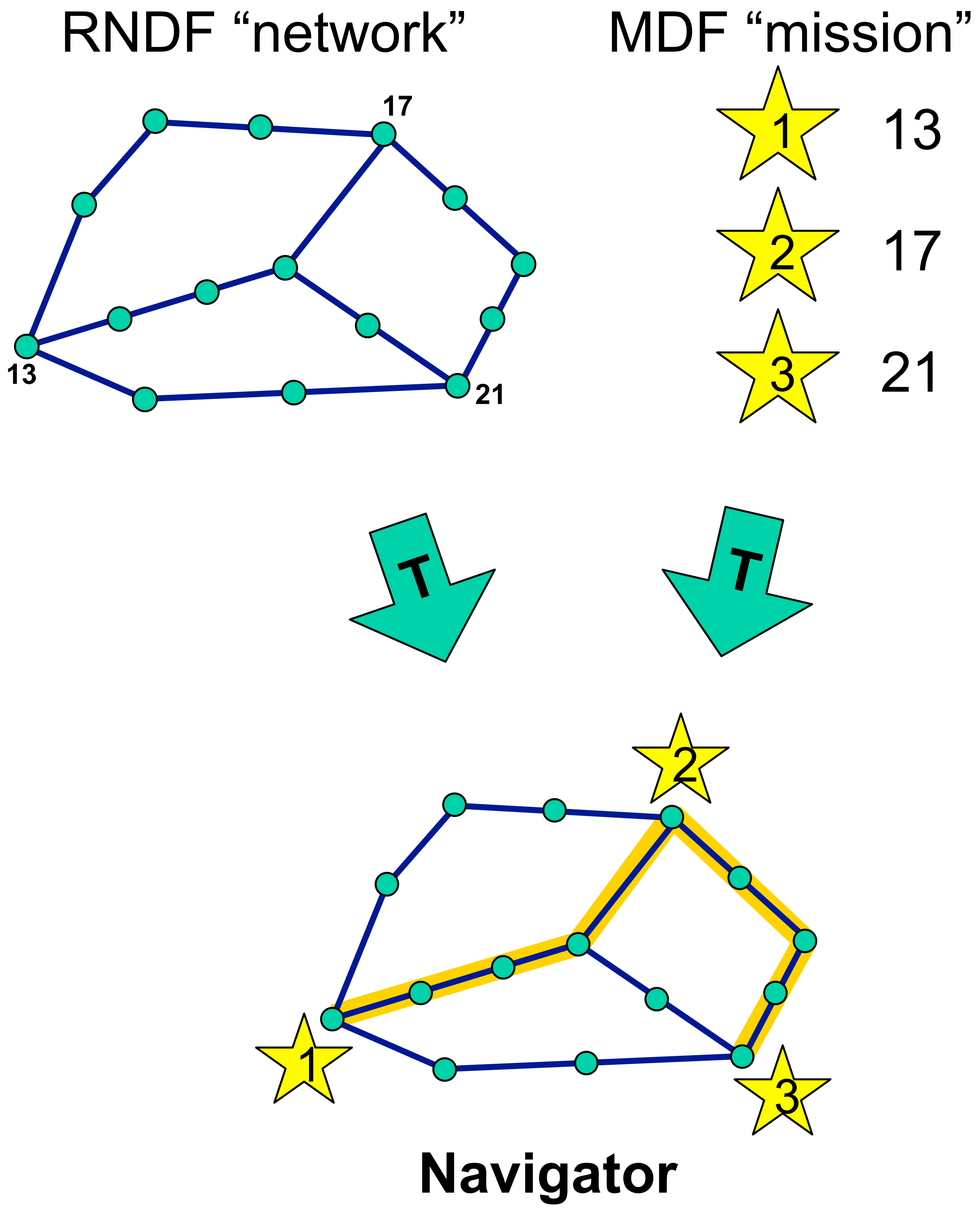




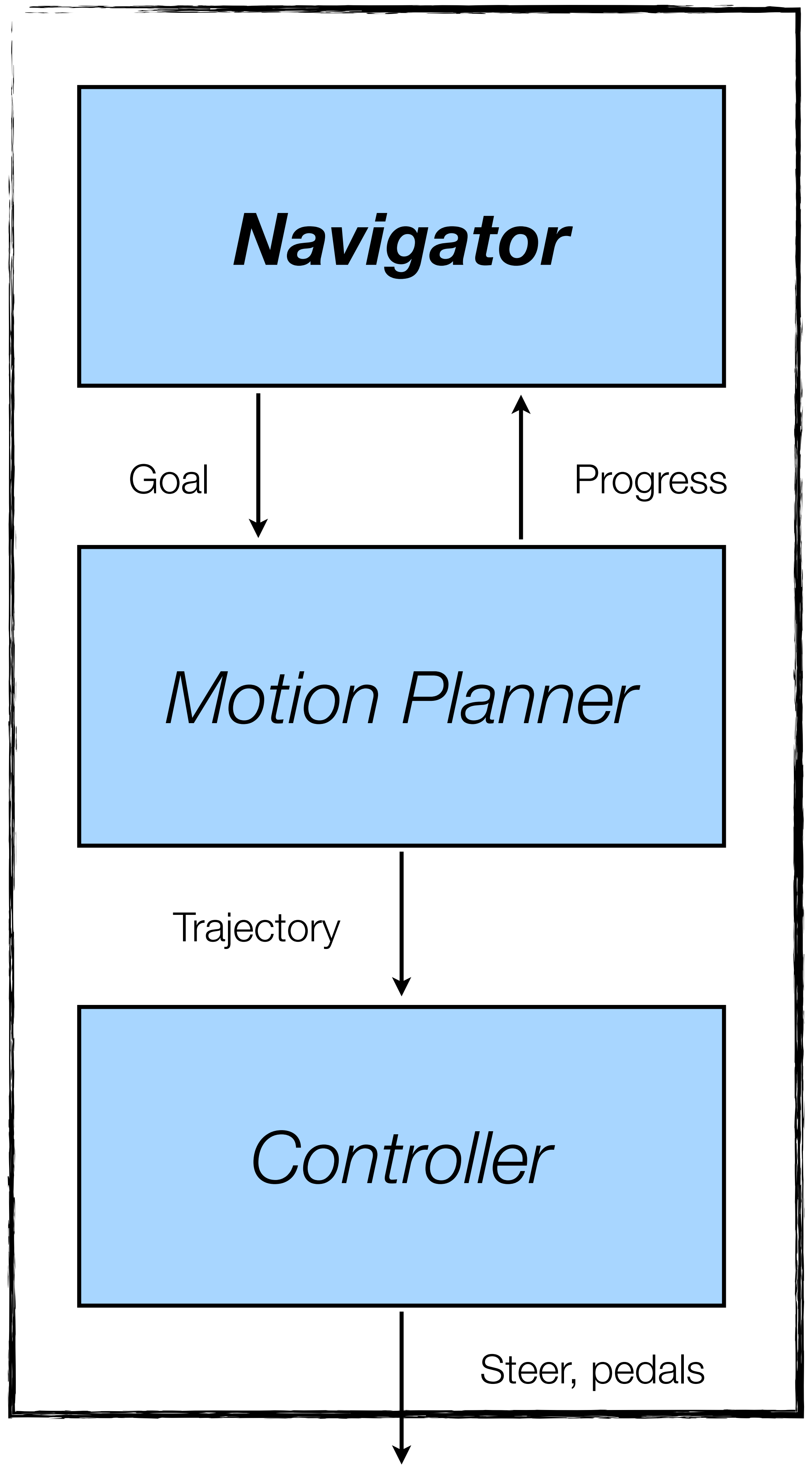
Navigator



A Algorithm for Navigating through the Road Network*

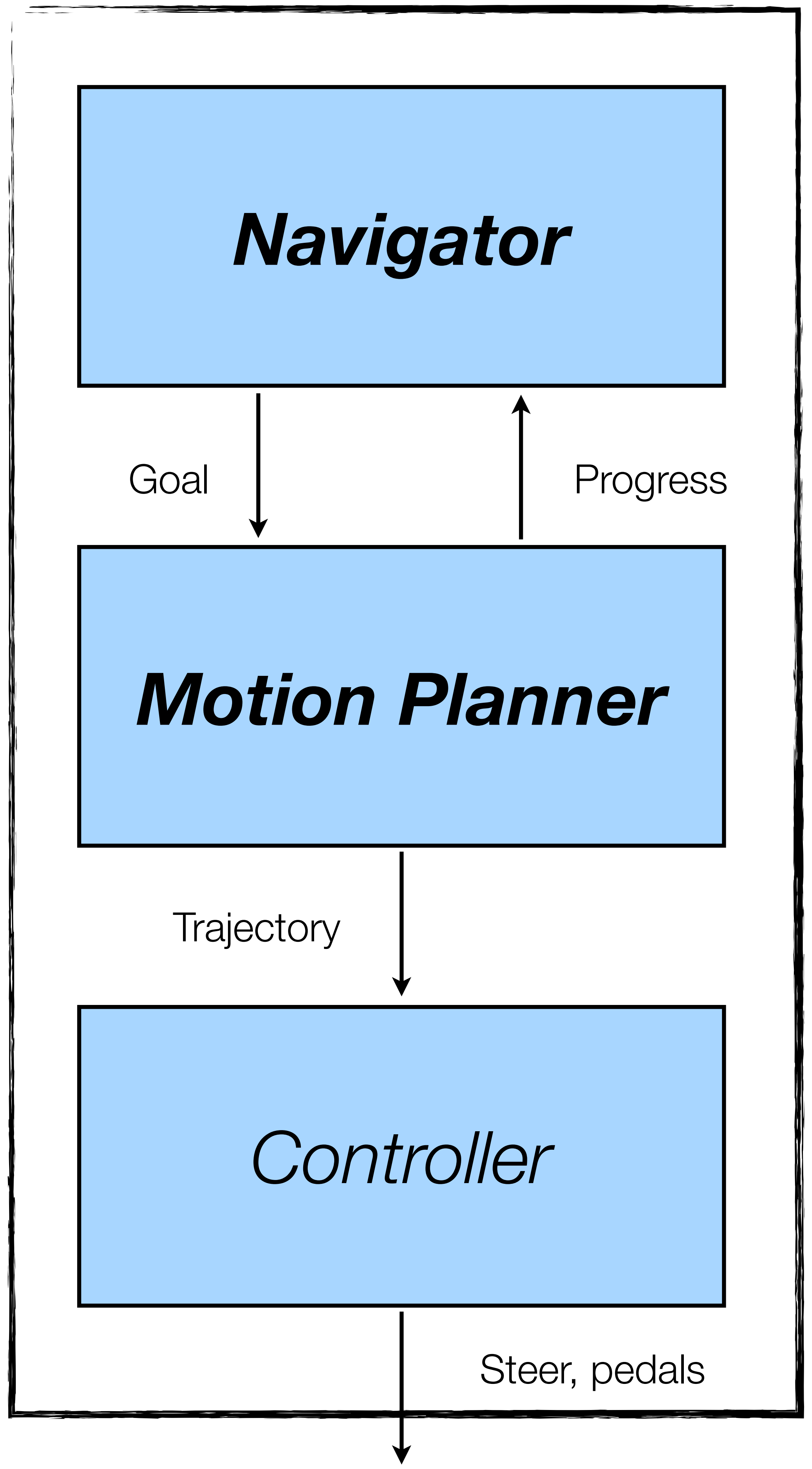


Navigator

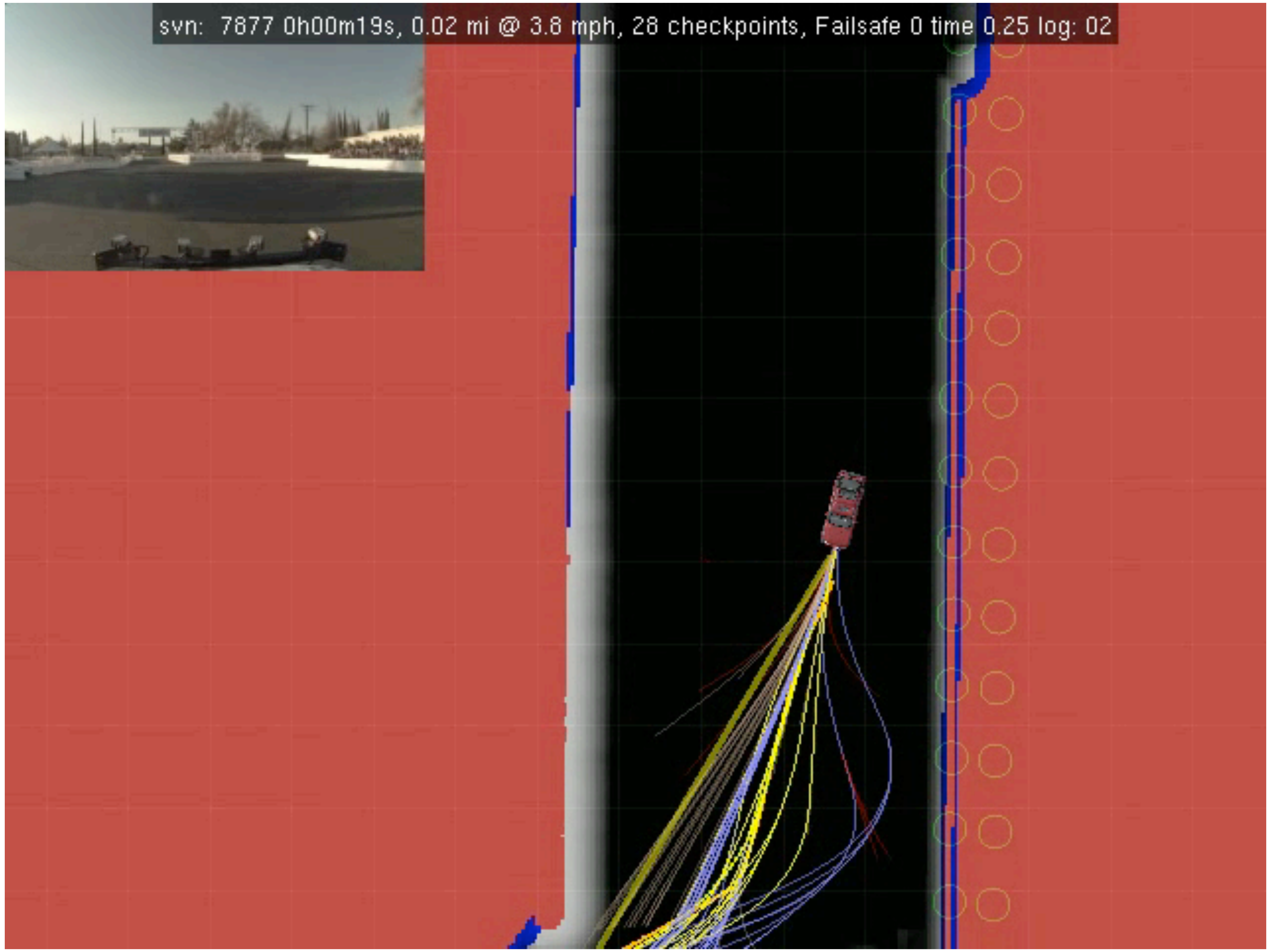


How can we program the navigator?

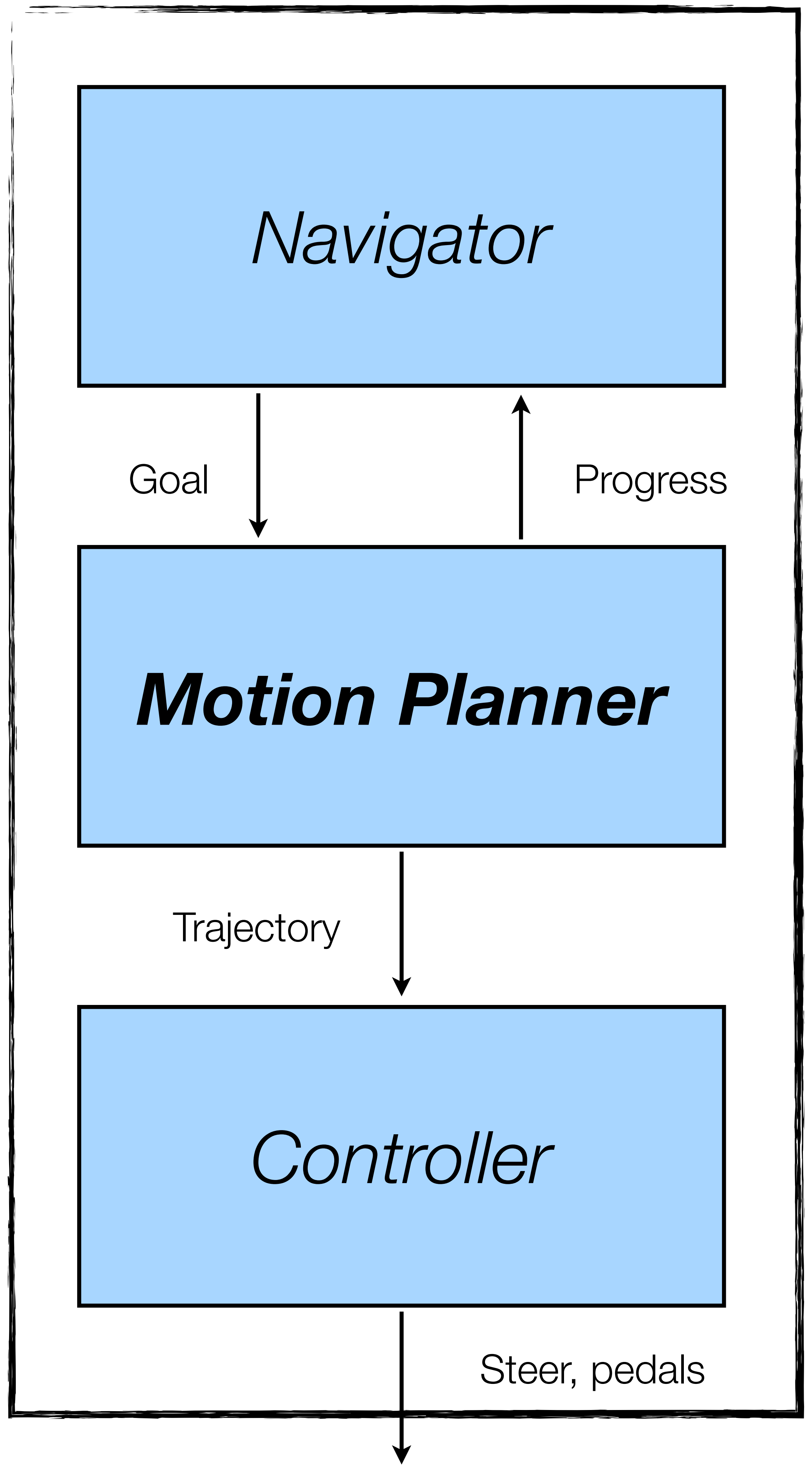
Navigator



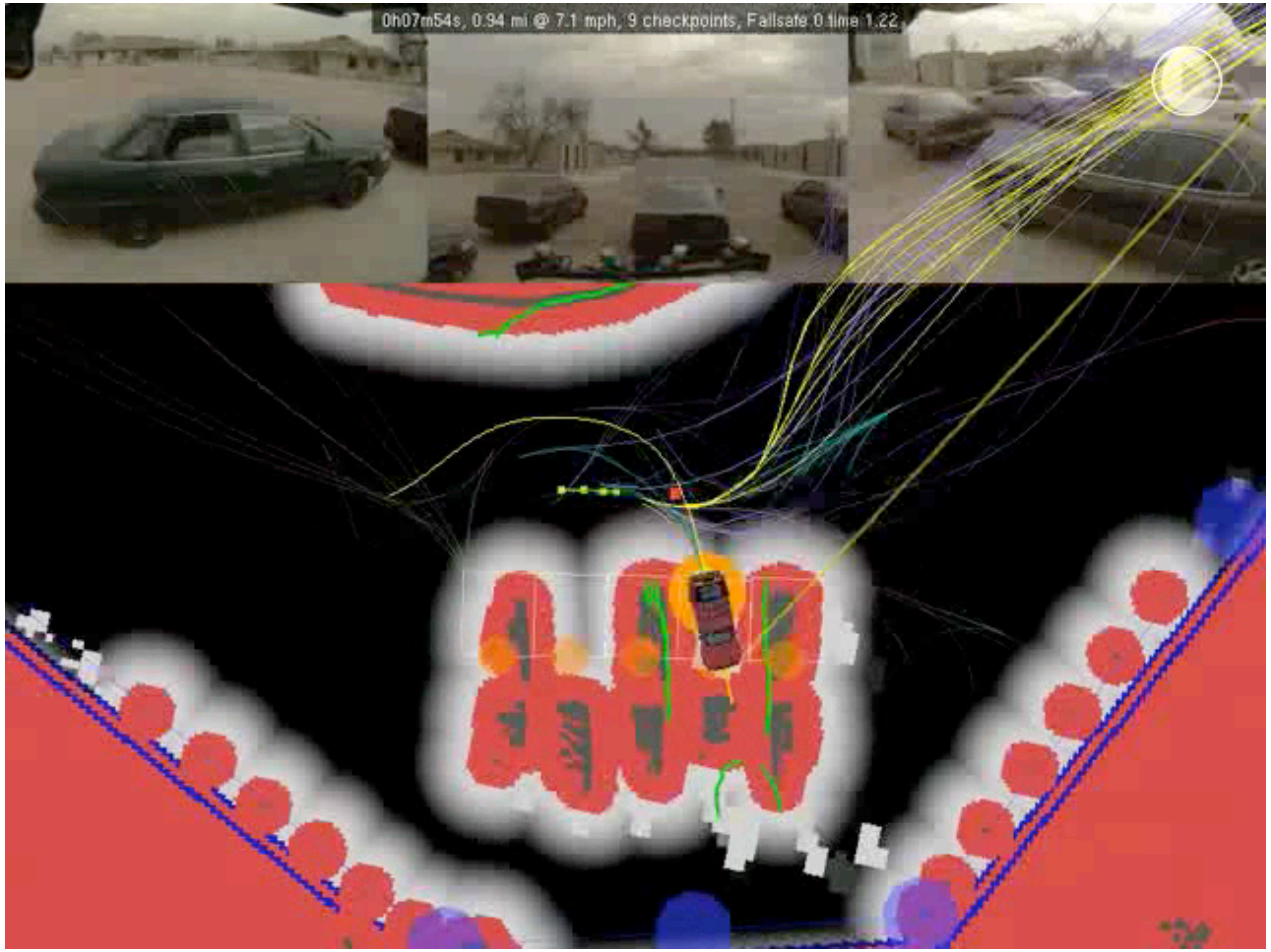
Task and Motion Planning Interaction

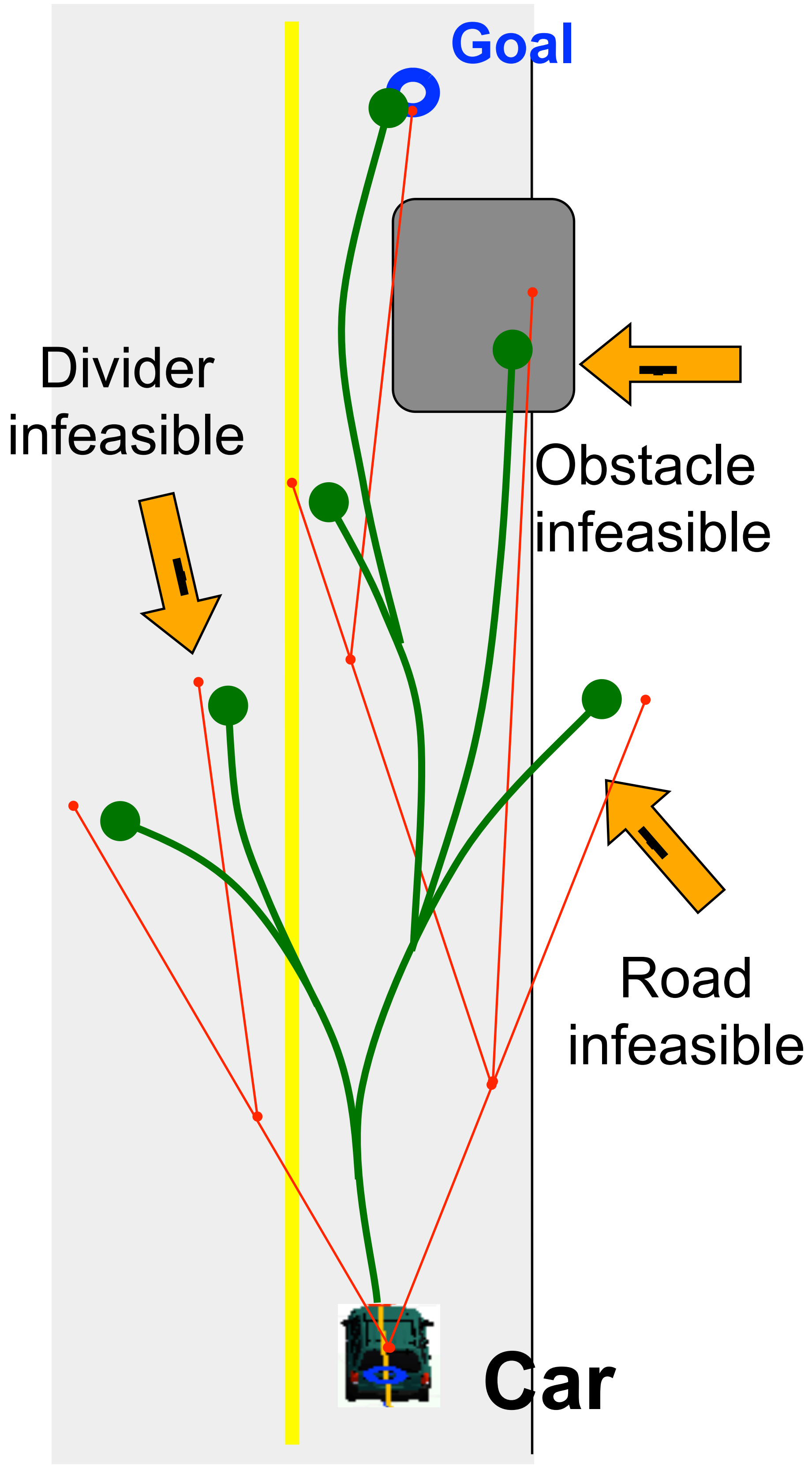


Navigator

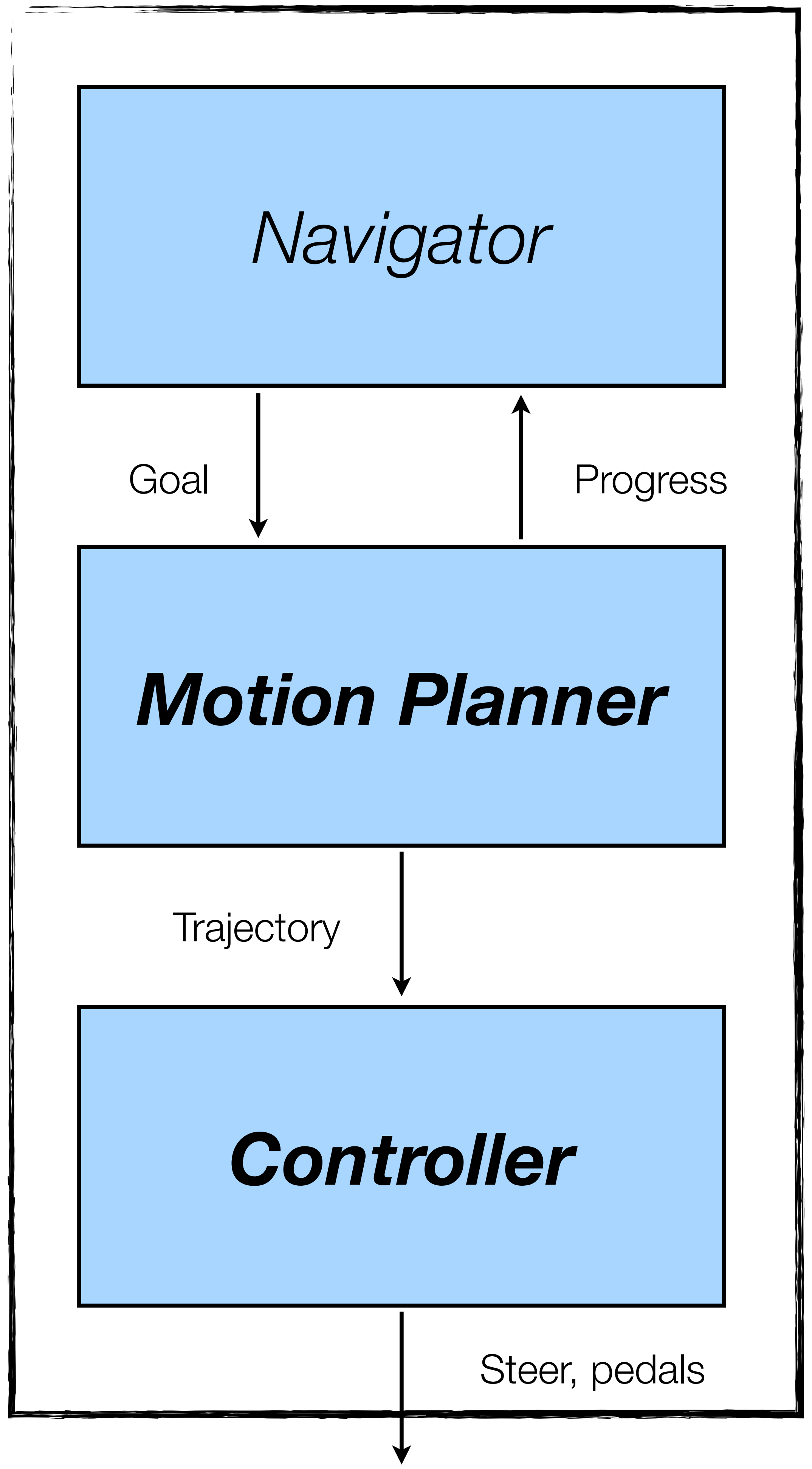


Rapidly-exploring Random Tree (RRT) for Motion Planning

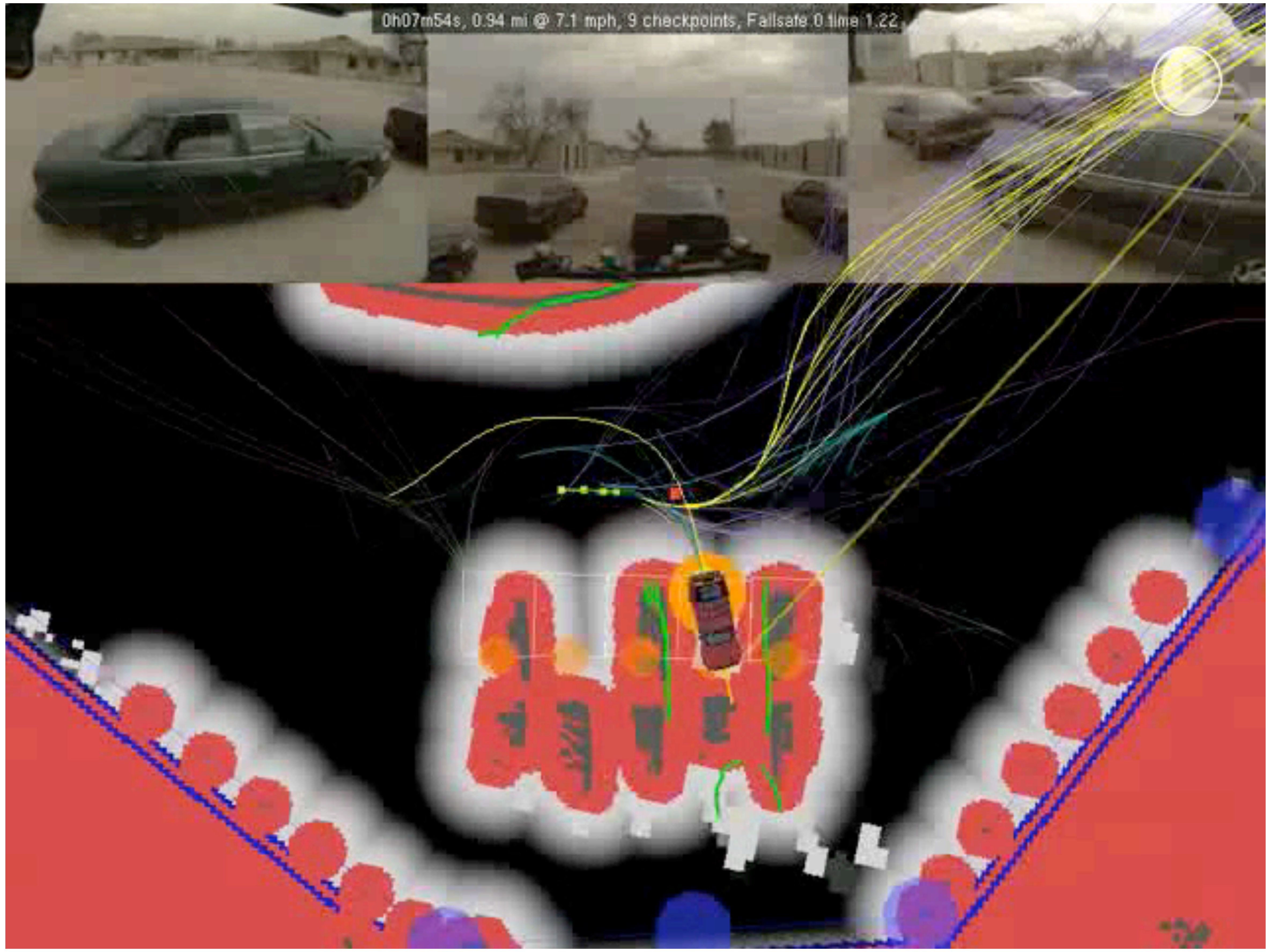


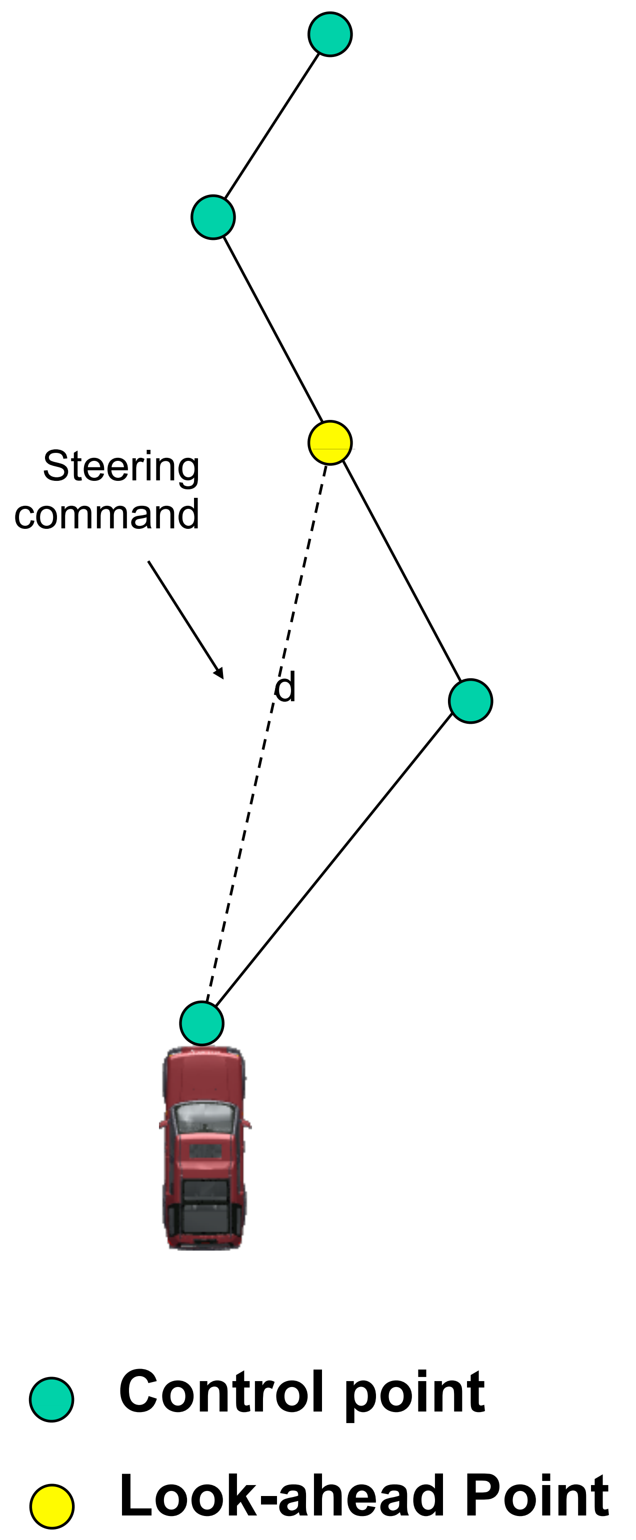


Navigator

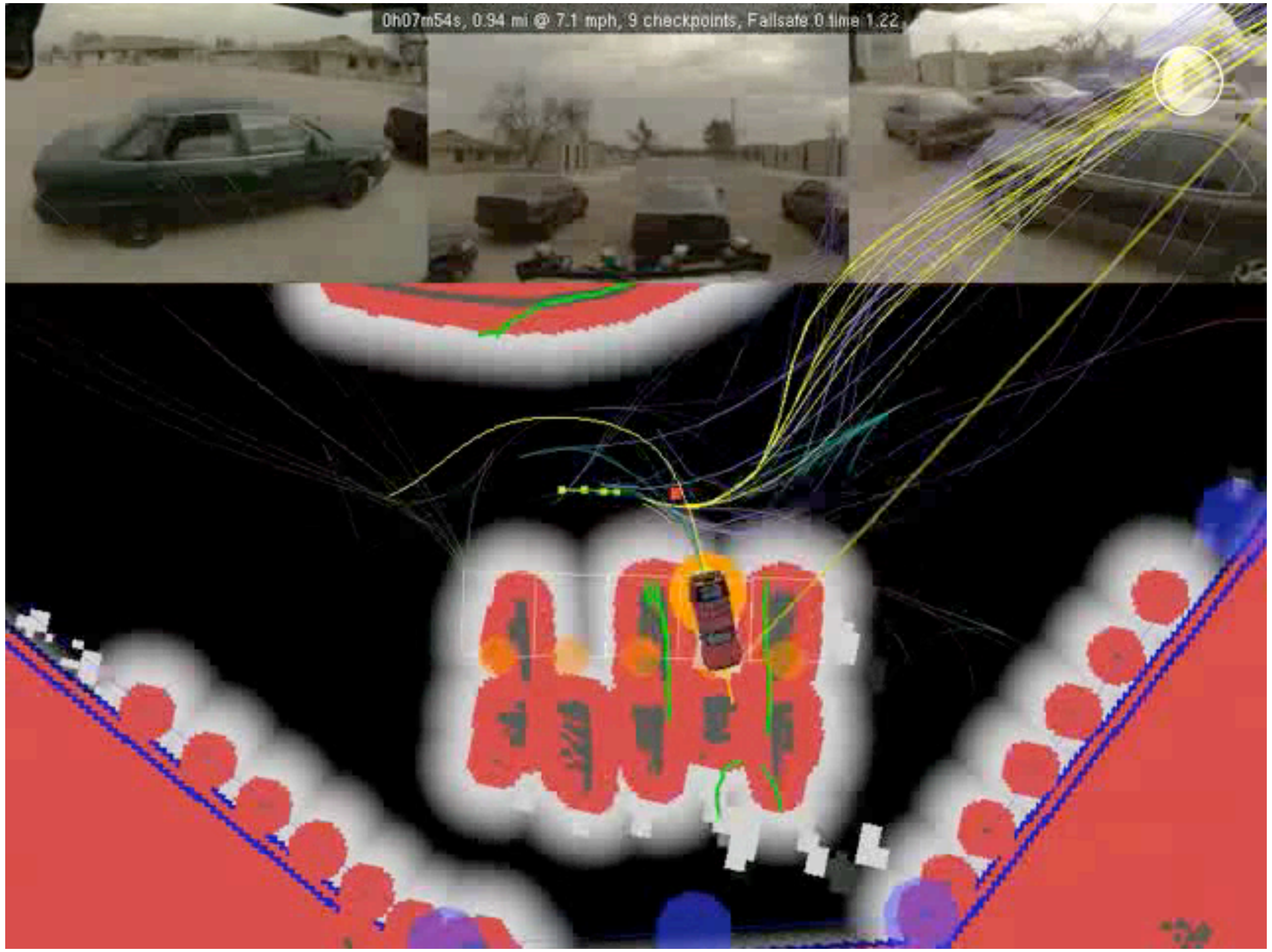


Rapidly-exploring Random Tree (RRT) for Motion Planning



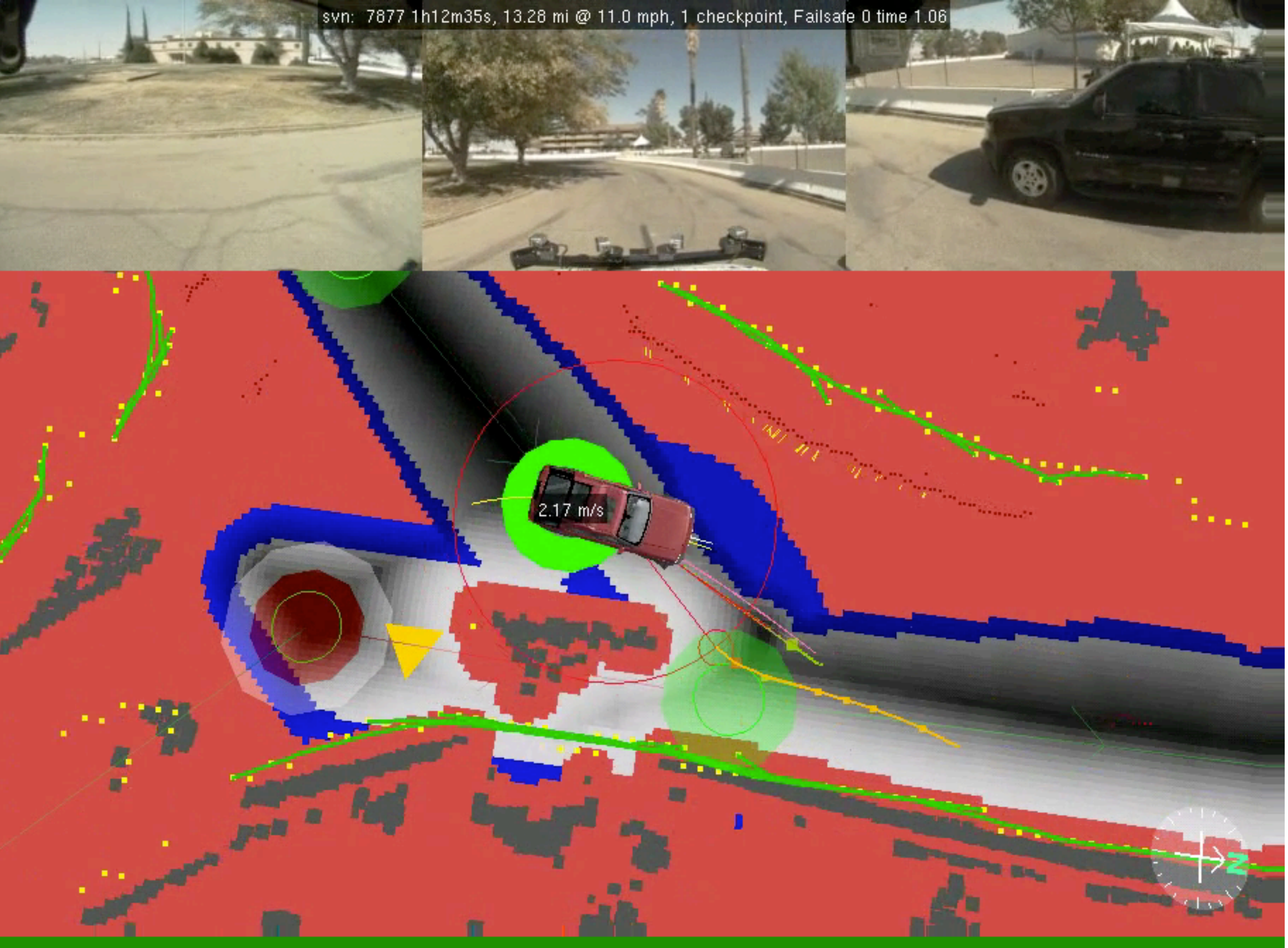


Results

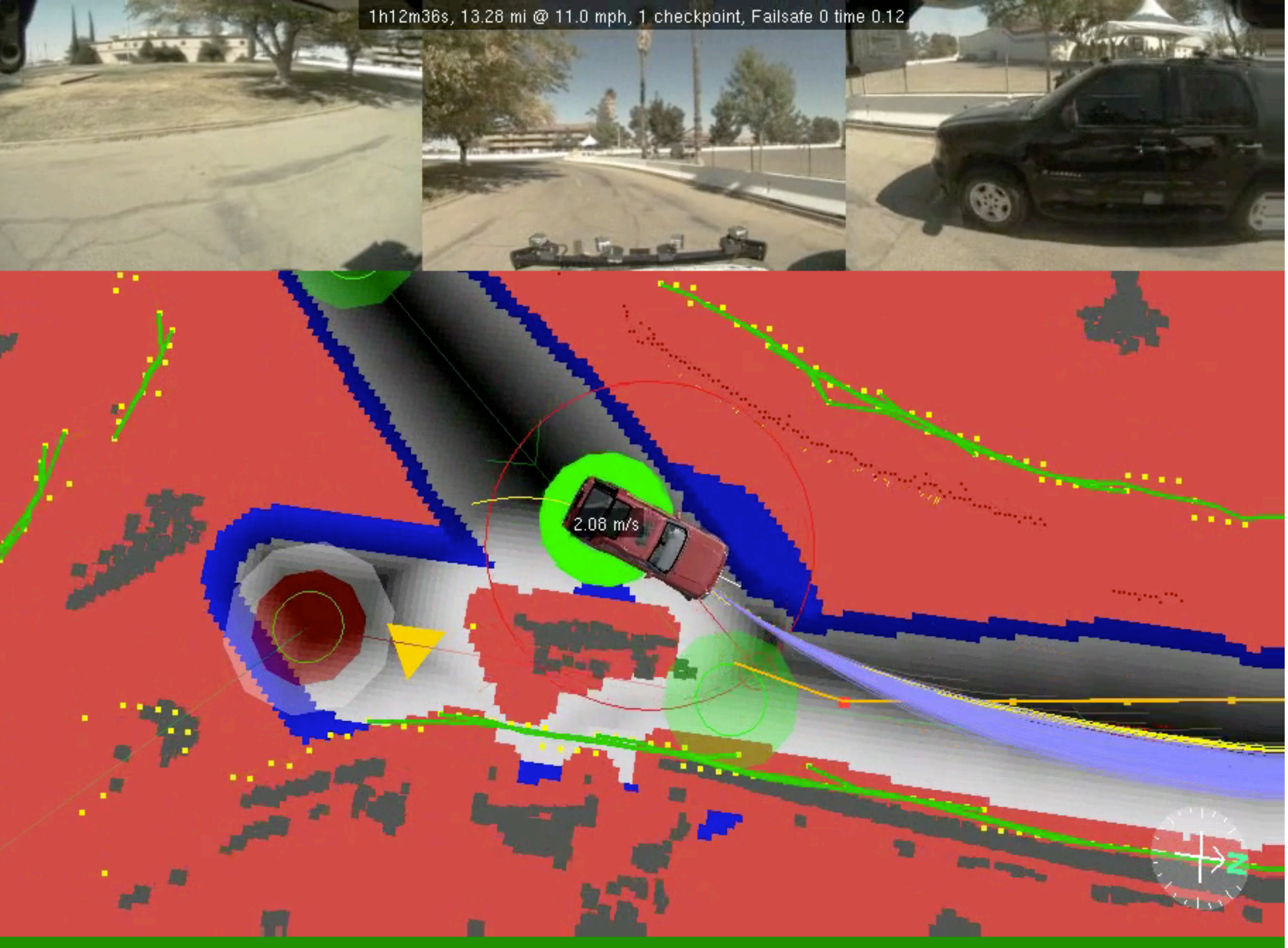

0h07m54s, 0.94 mi @ 7.1 mph, 9 checkpoints, Failsafe 0 time 1.22


28 remaining
0.00 m/s
PAUSE

DARPA Challenge (2007)



DARPA Challenge (2007)

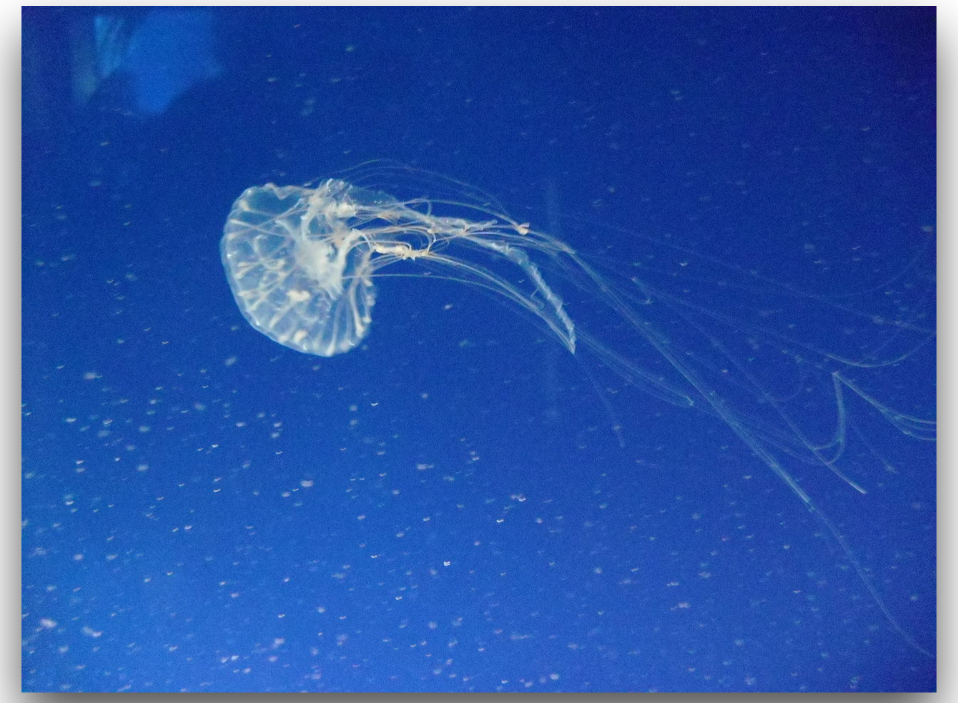


DARPA Challenge (2007)

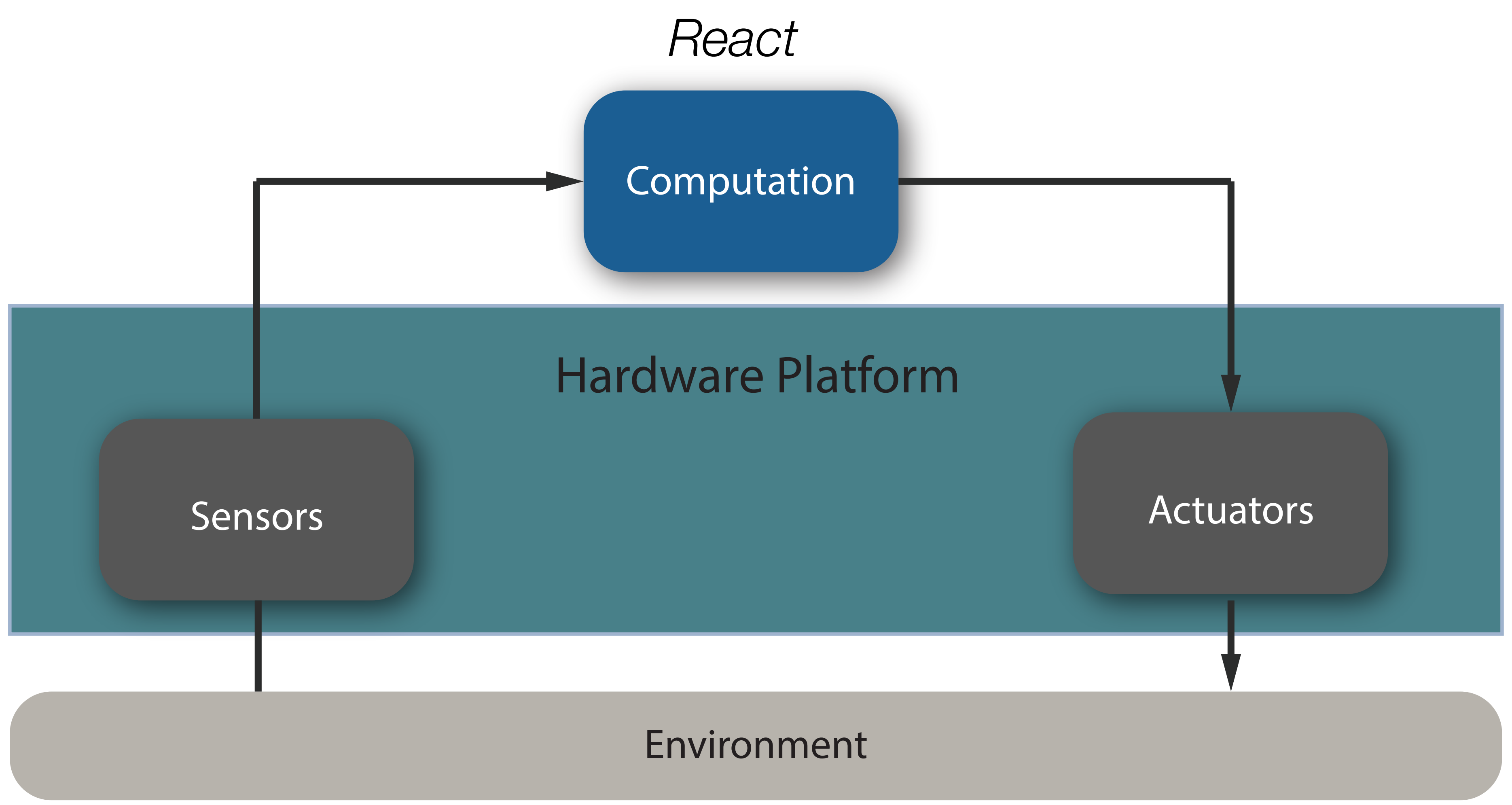
What do robots do exactly?

- Robots can be “defined” by the sensing-computation-actuation pipeline.

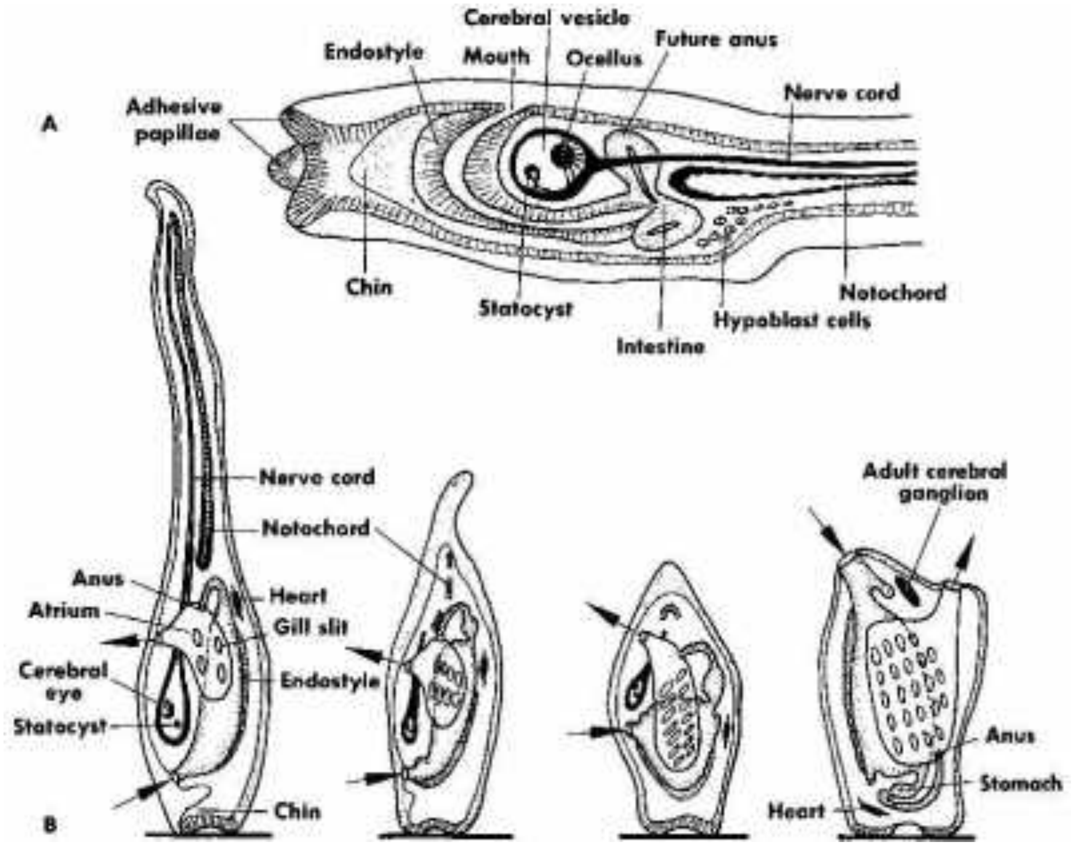
- **Simple computation: Reflexive control**
  - Most often a <u>direct</u> mapping from sensor data into actuation.



Jelly fish nerve net

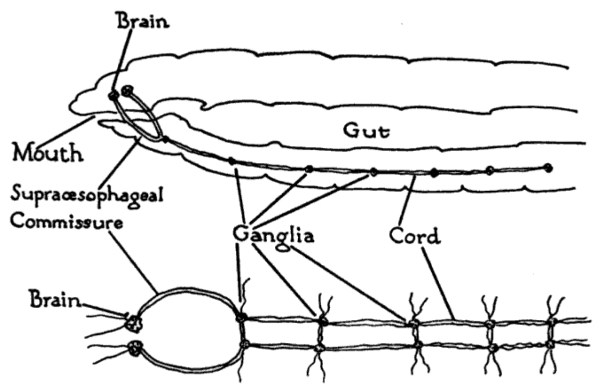


- **The tunicate (sea squirt) has an extreme life cycle:**
  - Starts out <u>mobile</u>, with a primitive eye and a nervous cord
  - Then, settles to a good spot, and *digests its own brain* once <u>stationary.</u>
- *Many scientists believe nervous systems evolved to satisfied the need to be mobile.*

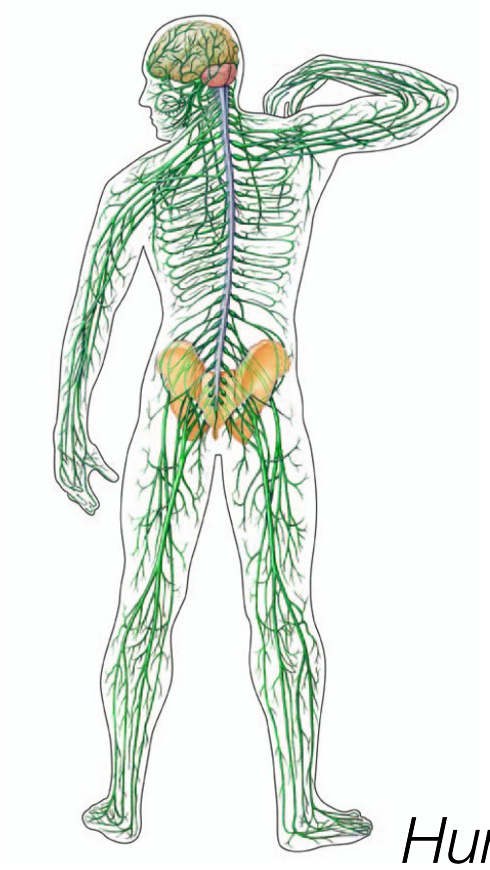


What do robots do exactly?

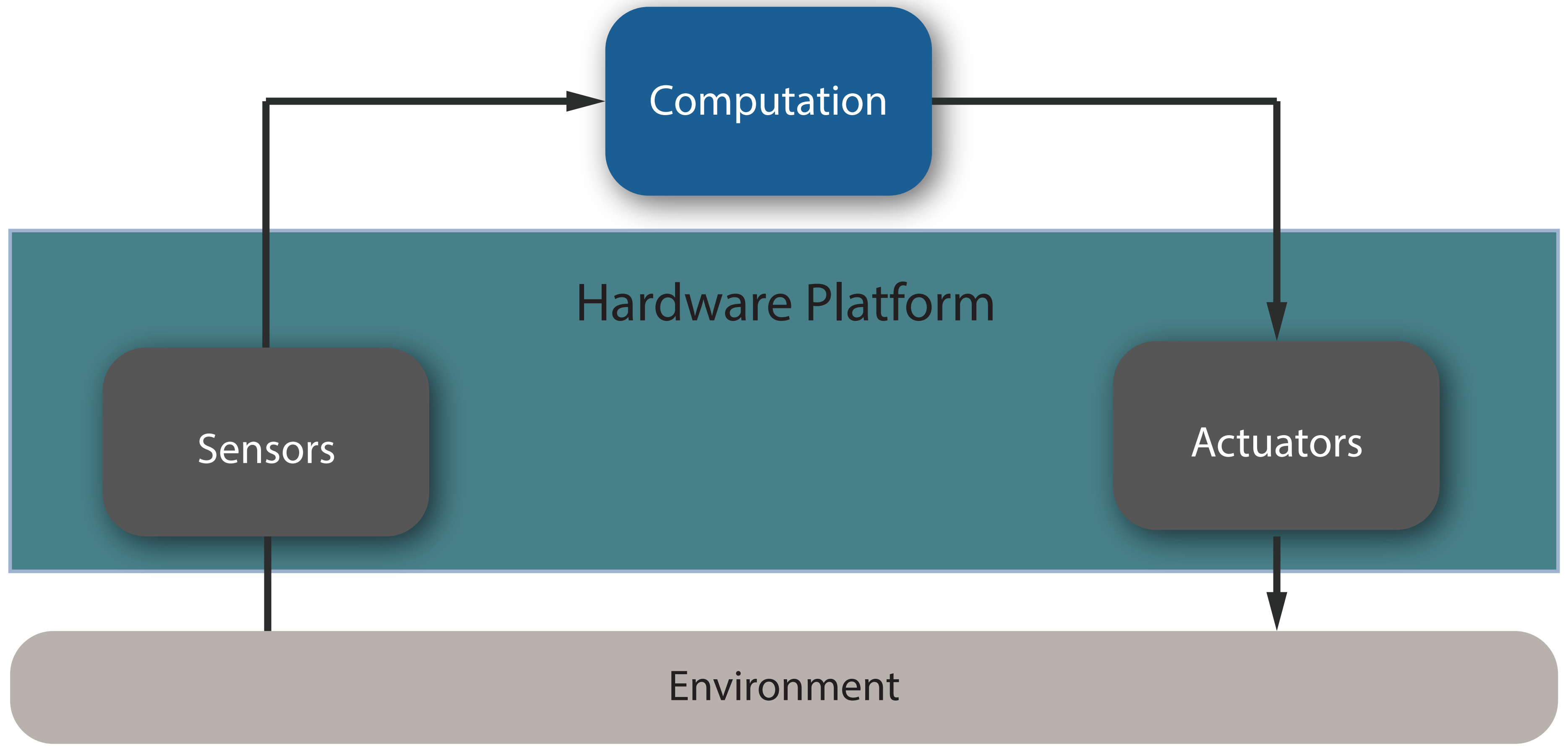
- **Horizontal breakdown of computation:**
  - *Perception and State Estimation:* Process the data to understand the environment and the state of the robot
  - *Planning and Control:* Given an understanding of the robot and its surroundings, make decisions to move the robot to accomplish the task



Earthworm nervous system

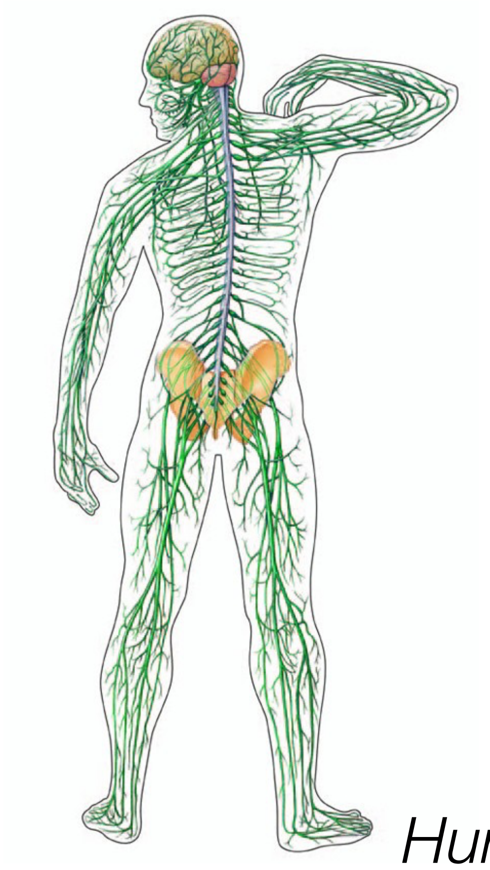


Human nervous system

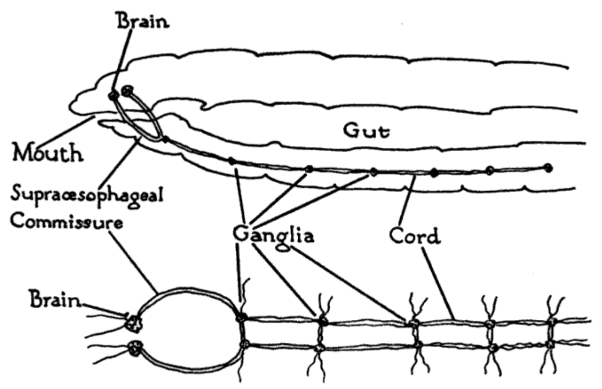


What do robots do exactly?

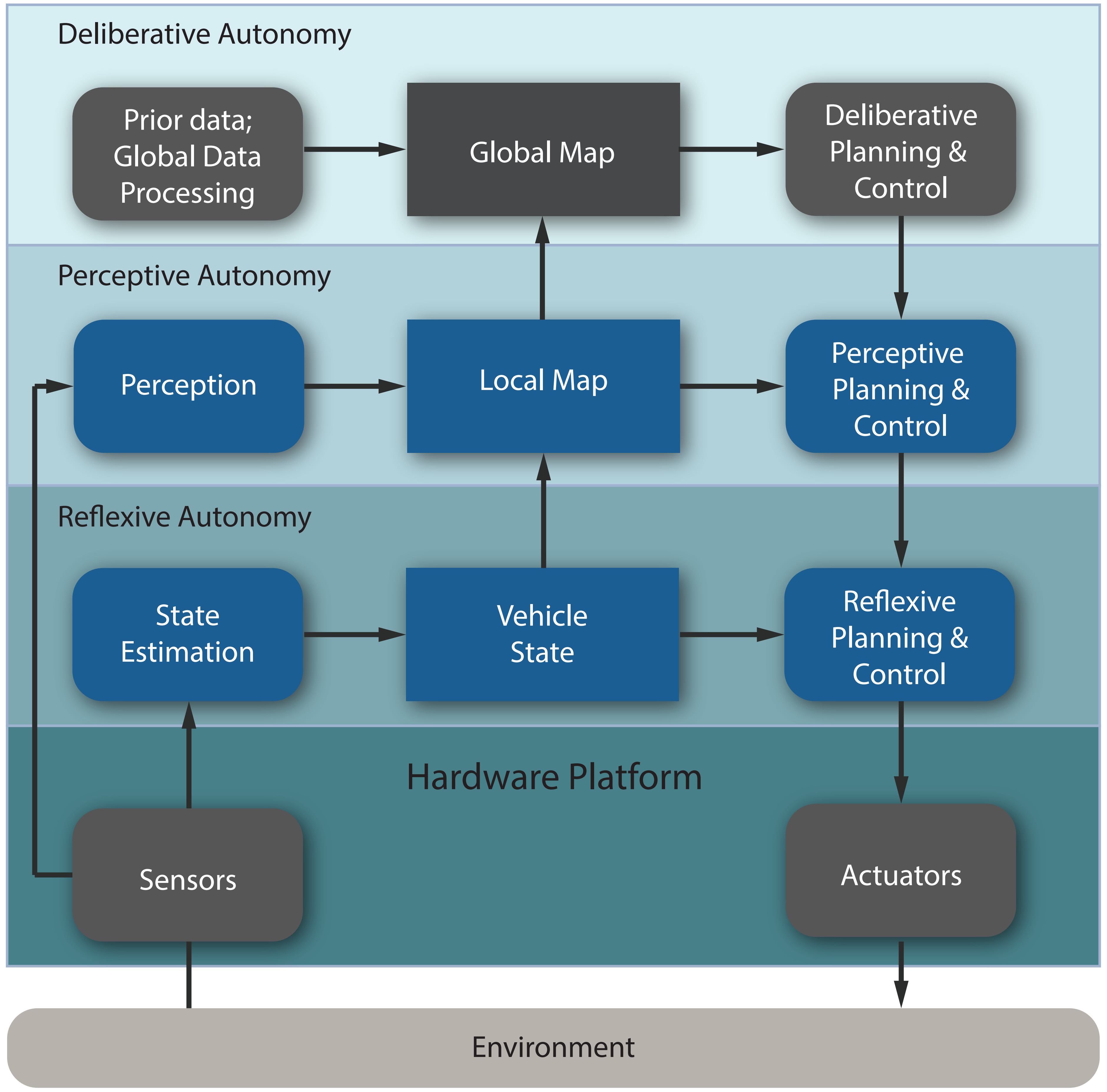
- **The vertical breakdown for computation:**
  - Most robotic systems rely on a three-layer software architecture.
  - The there layers can roughly be divided according to spatial- and temporal-scales.
  - The scales depend on the size/weight/task of the robot.



Human nervous system

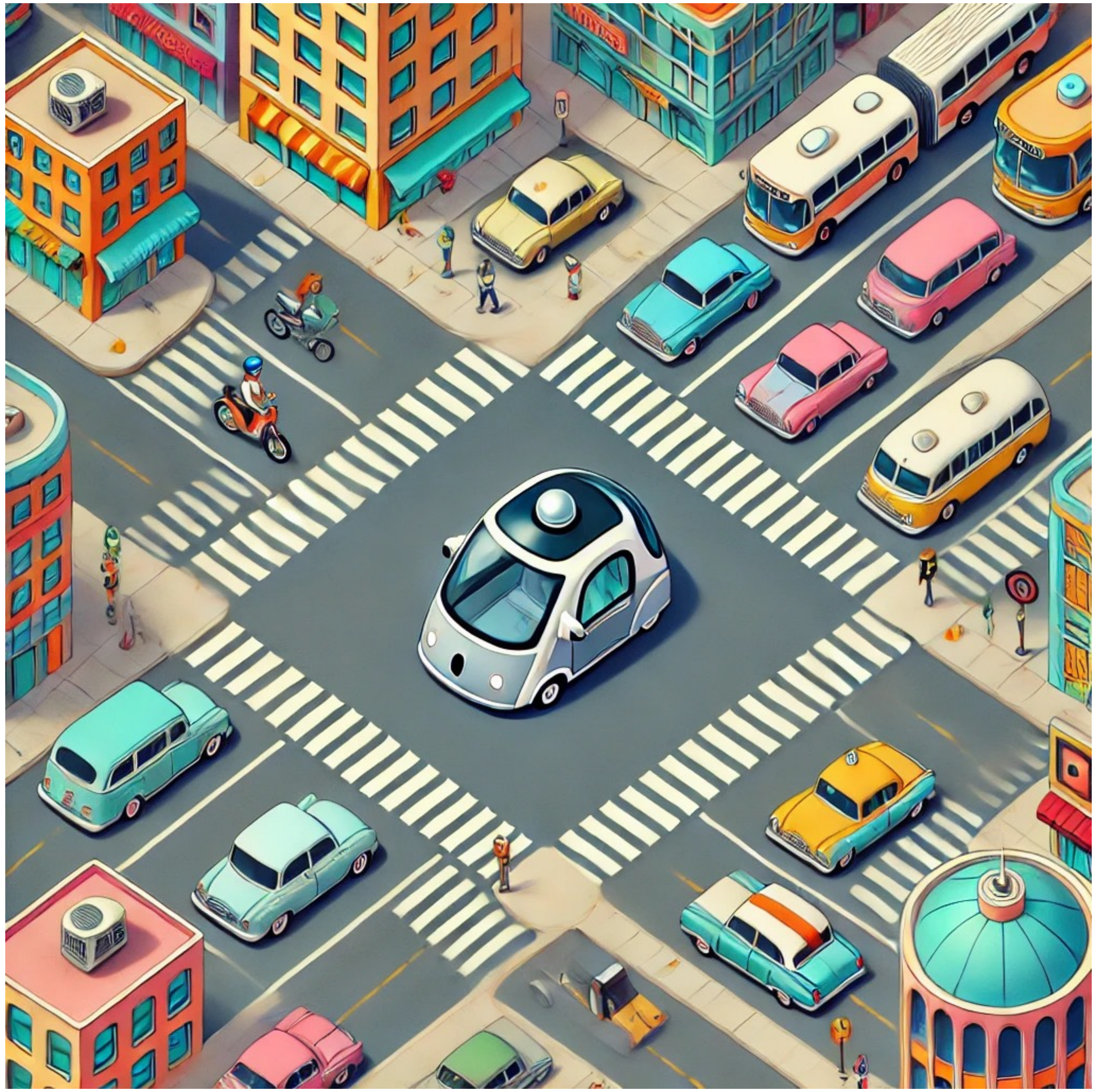


Earthworm nervous system



Planning and Reasoning

Self-driving Cars



- Can you write down a simple state representation for the self-driving car?
- What are key considerations?
- What makes this state representative?
- What are your underlying assumptions?
- How would these change for other similar examples:
  - Planetary rover
  - Delivery drones

Warehouse order packing

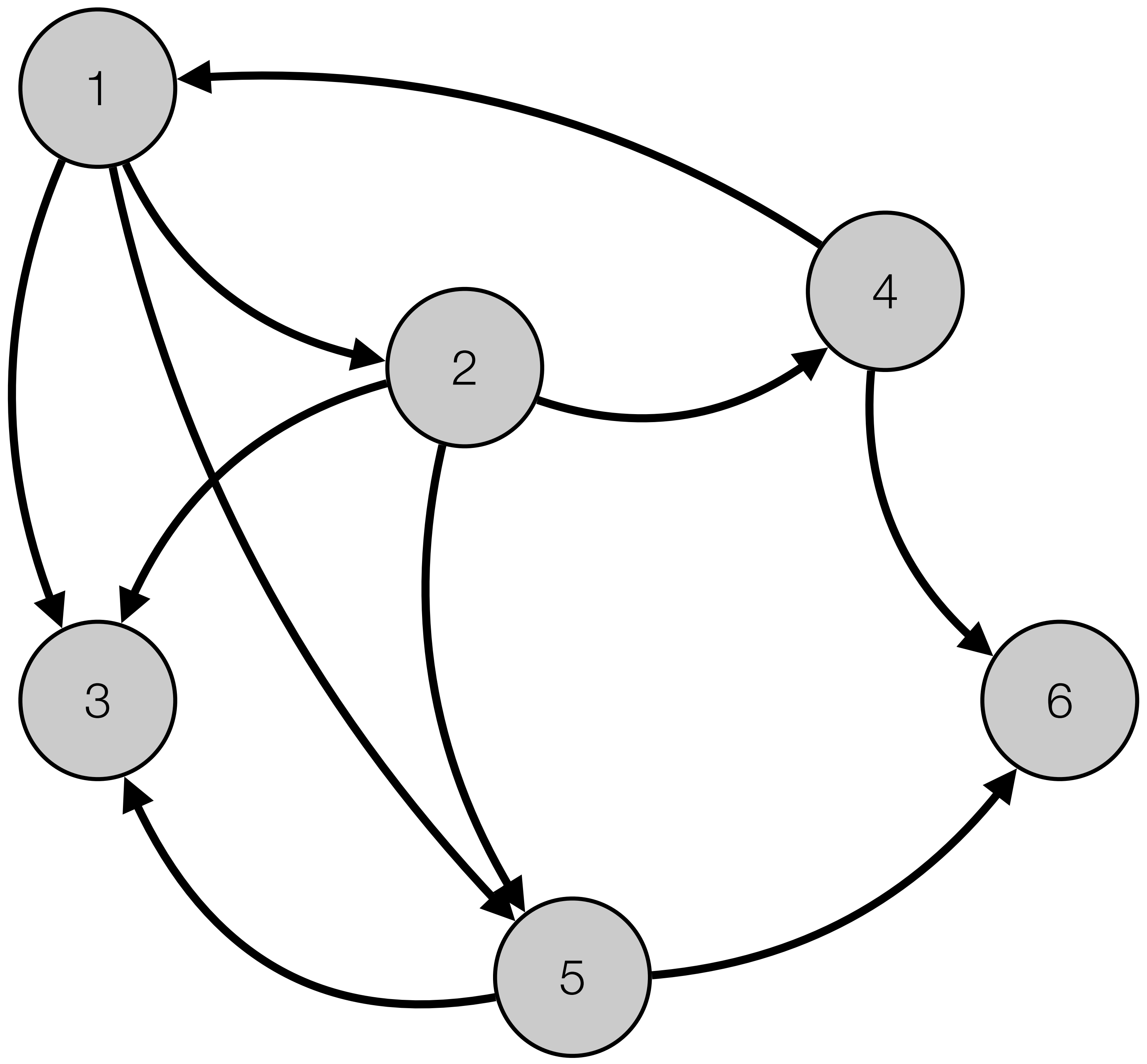
- Can you write down a simple state representation for the warehouse packing?

- What are key considerations?
- What makes this state representative?
- What are your underlying assumptions?

- How would these change for other similar examples:
  - Planetary rover
  - Delivery drones

On State Space Modeling

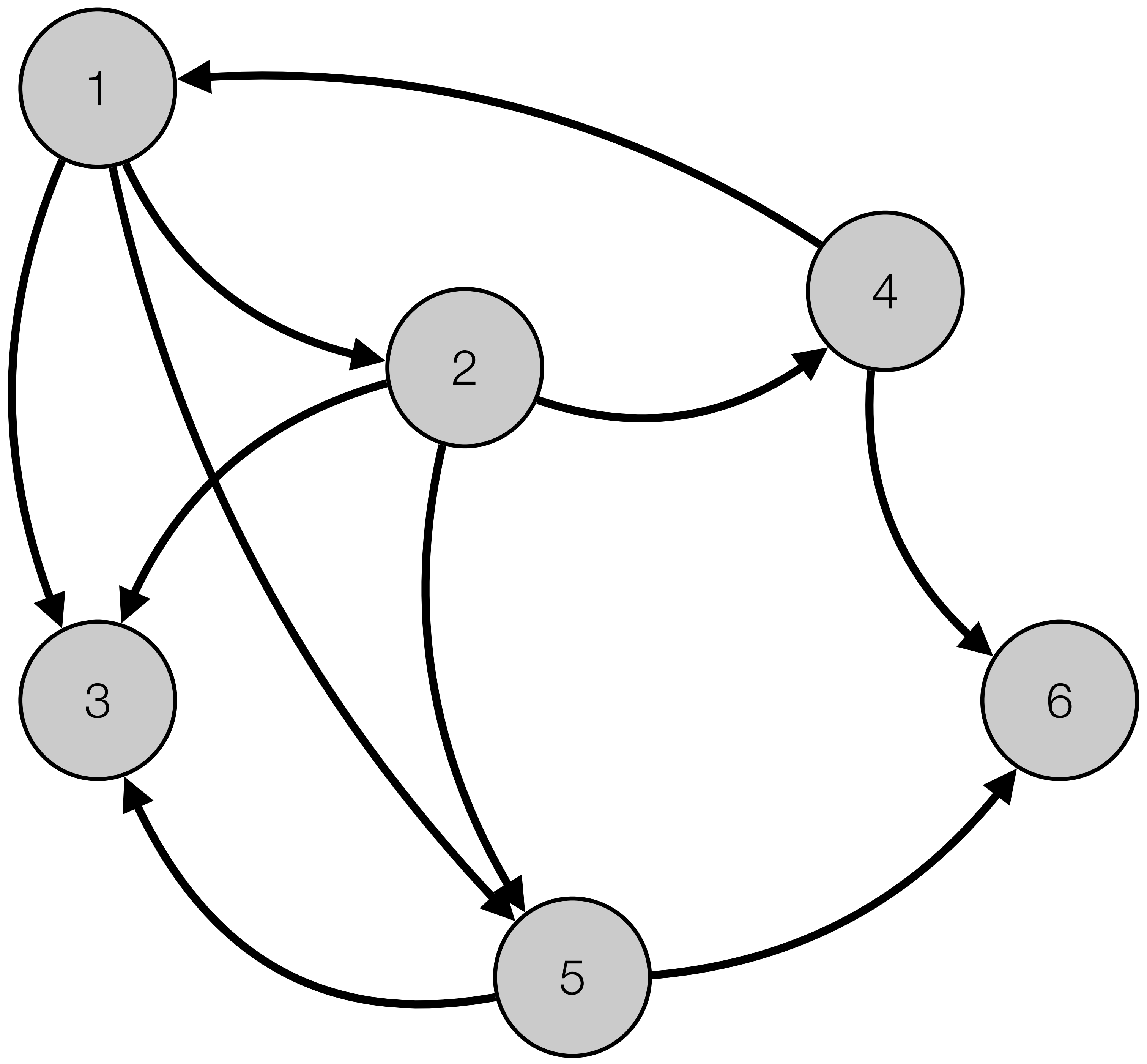
- **State-state models** involve:
  - Sates
  - Actions
  - Transition function



- **Planning problems** involve:
  - State-space model
  - Start state
  - Goal state
  - (Optional) cost/reward function

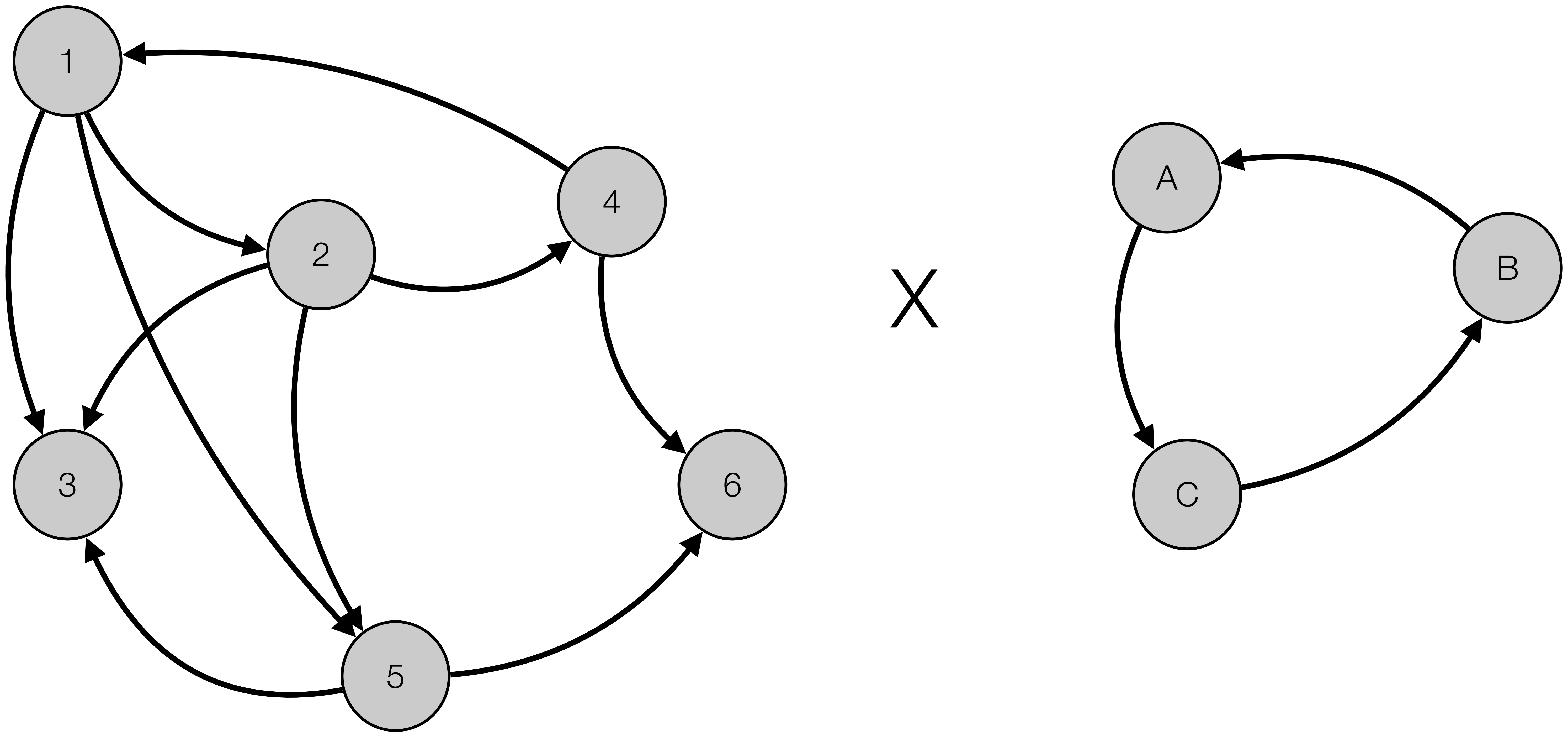
On The Representational Power of State-space Models

- **State-state models** can represent numerous elements of a problem by encoding it into the state, action and transition function:
  - Obstacles
  - Deadlines
  - Coordination
  - …



On Composing State-space Models

- **State-state models** for multiple entities can be composed to induce multi-system behavior:
  - States of composed model: Tuples of states
  - Actins of composed model: Tuples of actions
  - Transition function of composed model: Combines both entities taking actions at the same time



Representing State Space Models in Code

- **Use lists or dictionaries to represent states**
- **Use functions to represent transition functions**

```
states = [1, 2, 3, 4, 5, 6, 7, 8, 9]

actions = ["left", "right", "up", "down"]

def transition(state, action):
```

Representing State Space Models in Code

- **Use lists or dictionaries to represent states**

- **Use functions to represent transition functions**

```
def transition(state, action):
```

Try all combinations?

Representing State Space Models in Code

```
def transition(state, action):
    # Define grid dimensions
    grid = [
        [1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]
    ]

    # Map state to (row, col)
    for row in range(3):
        for col in range(3):
            if grid[row][col] == state:
                r, c = row, col
                break
        else:
            continue
        break
    else:
        return None  # state not found
```

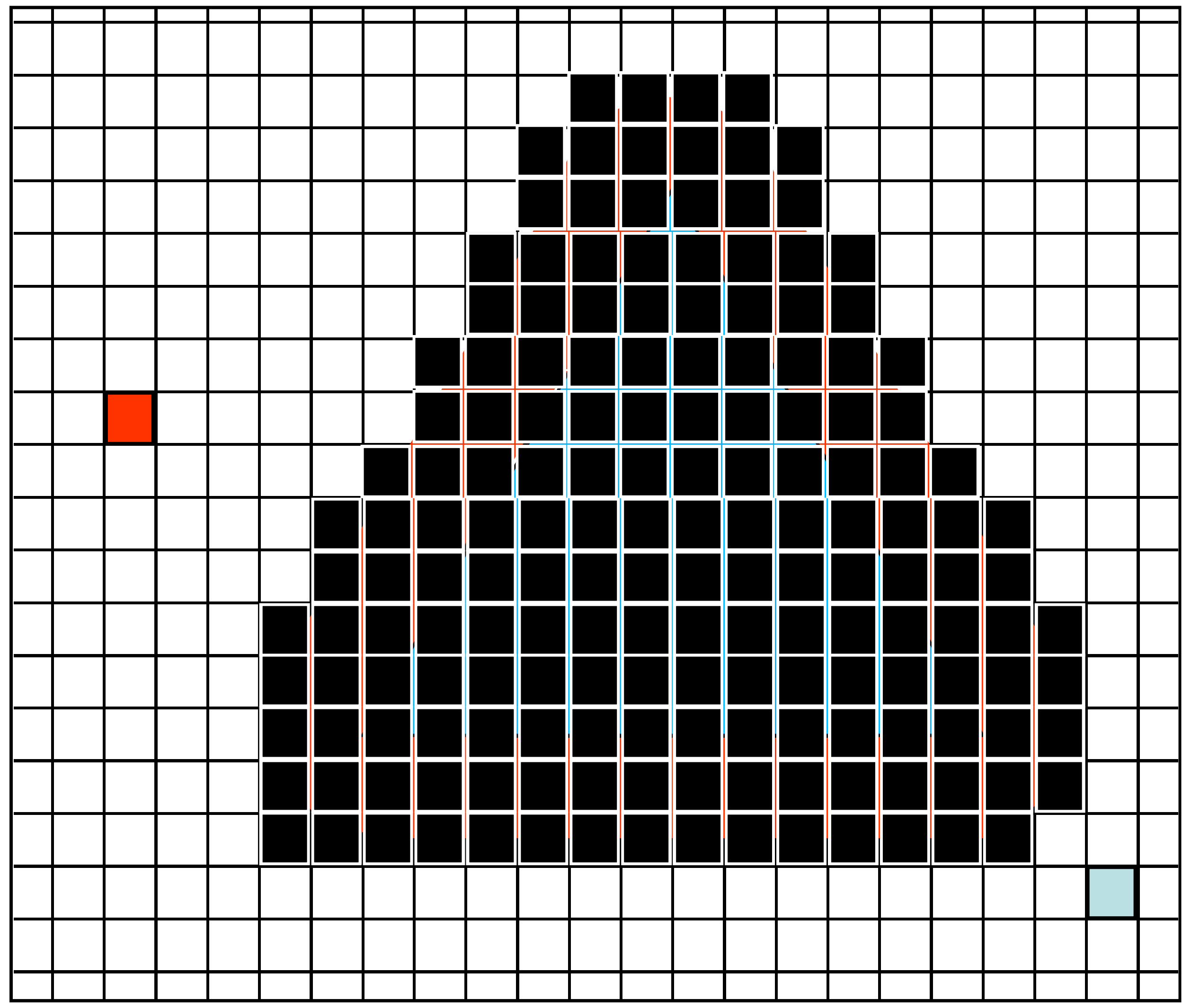
```
    # Compute new position based on action
    if action == "up":
        r -= 1
    elif action == "down":
        r += 1
    elif action == "left":
        c -= 1
    elif action == "right":
        c += 1
    else:
        return None  # Invalid action

    # Check bounds
    if 0 <= r < 3 and 0 <= c < 3:
        return grid[r][c]
    else:
        return None  # Action not valid from this state
```

State-space Search Methods

- State-space search amounts to finding a “Path” from the start state to the goal state.

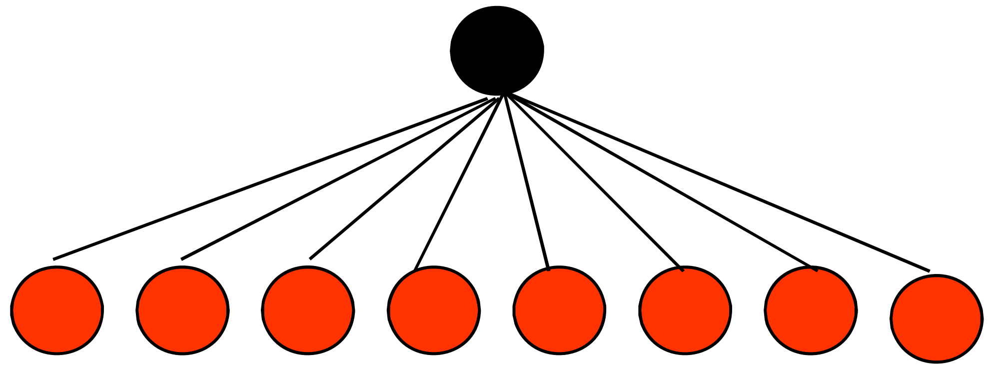
- Optionally this can involve an “objective” function as well: minimize cost or maximize reward

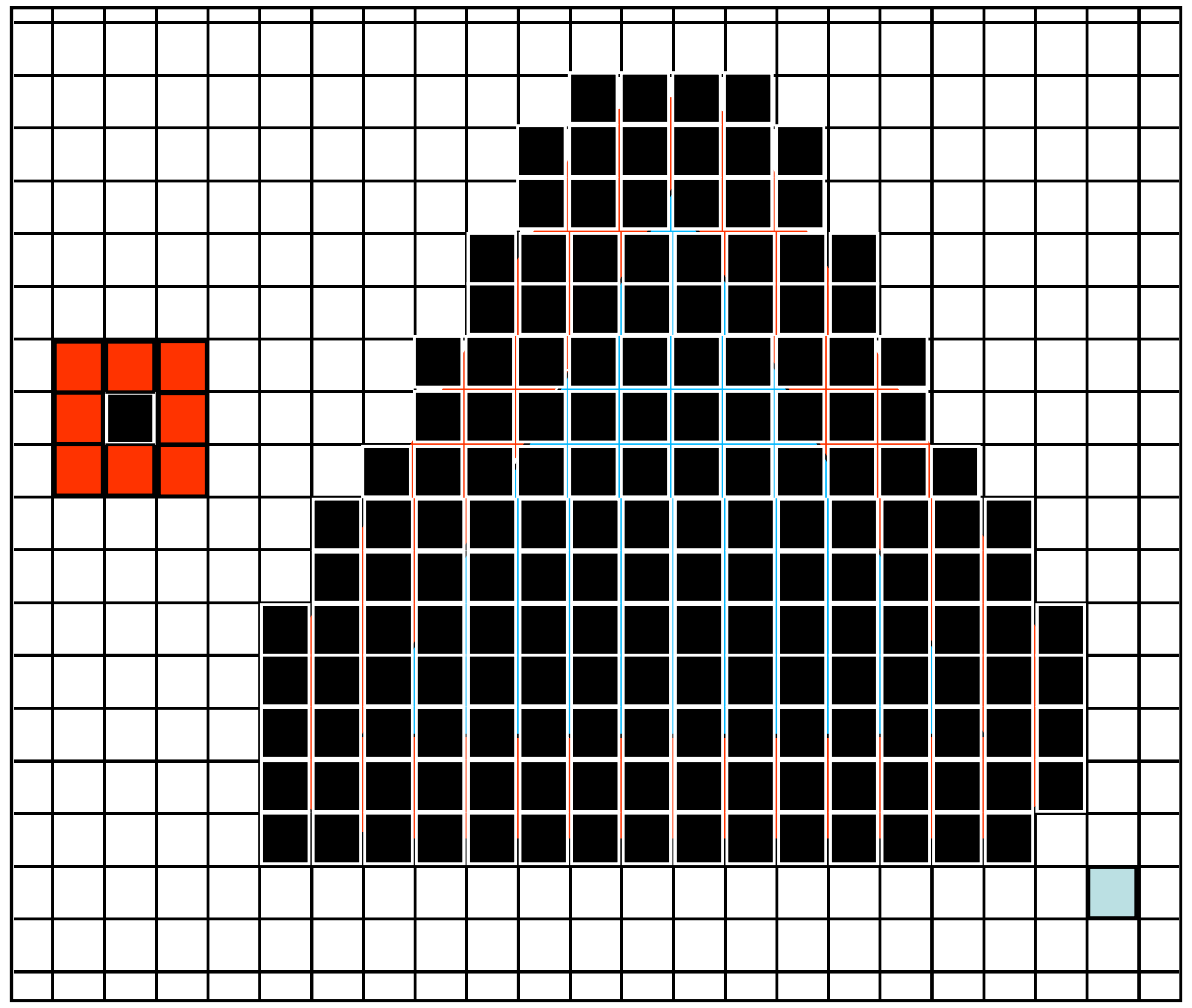


State-space Search Methods

- State-space search amounts to finding a “Path” from the start state to the goal state.

- Optionally this can involve an “objective” function as well: minimize cost or maximize reward

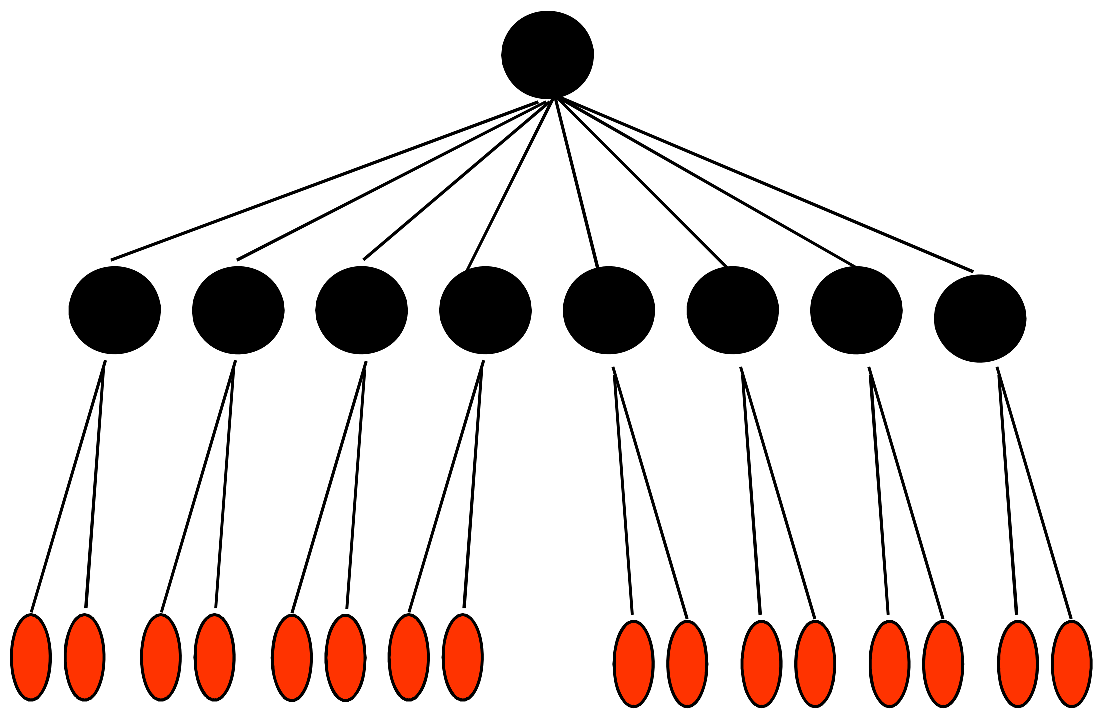


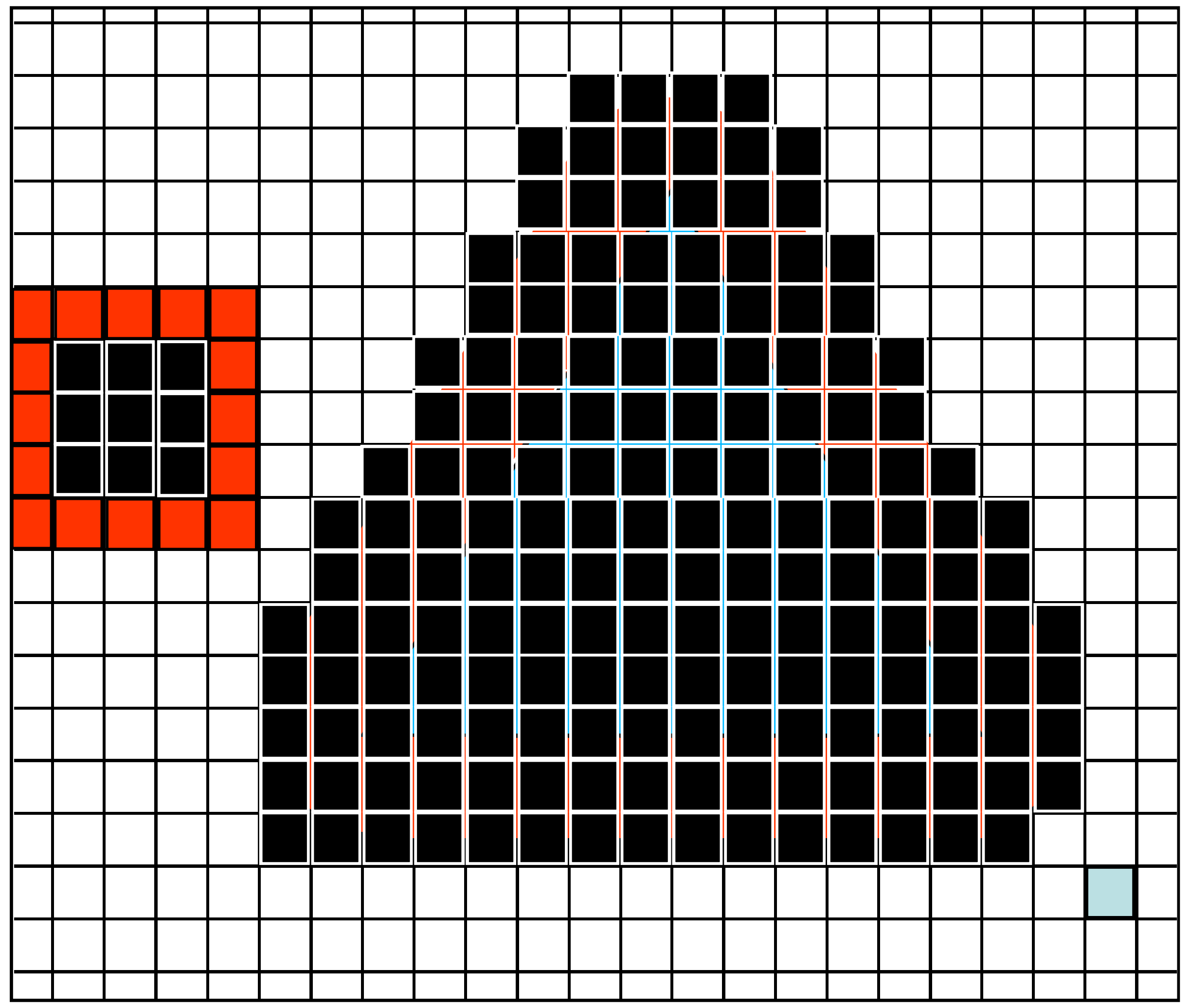


State-space Search Methods

- State-space search amounts to finding a "Path" from the start state to the goal state.

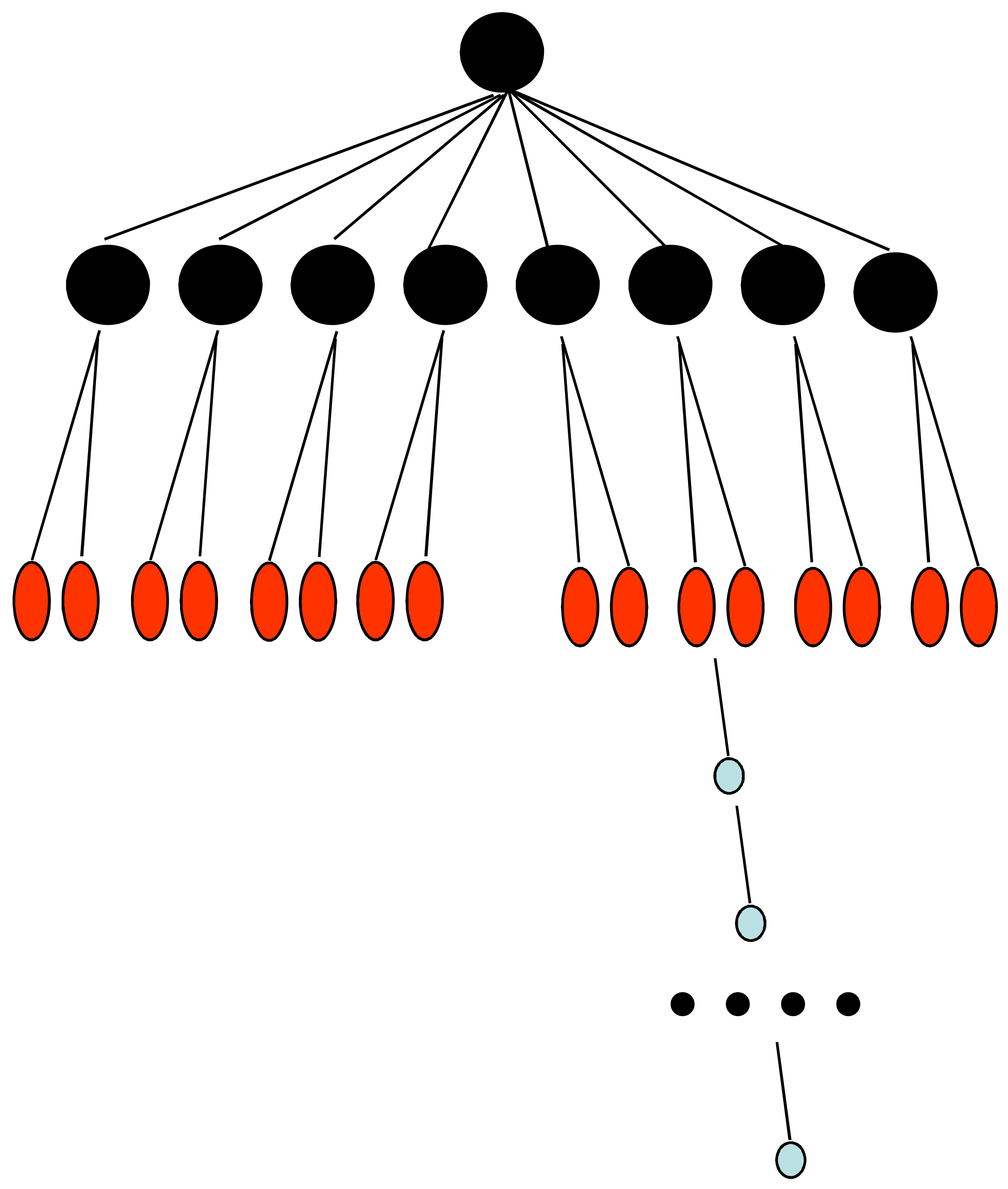
- Optionally this can involve an "objective" function as well: minimize cost or maximize reward

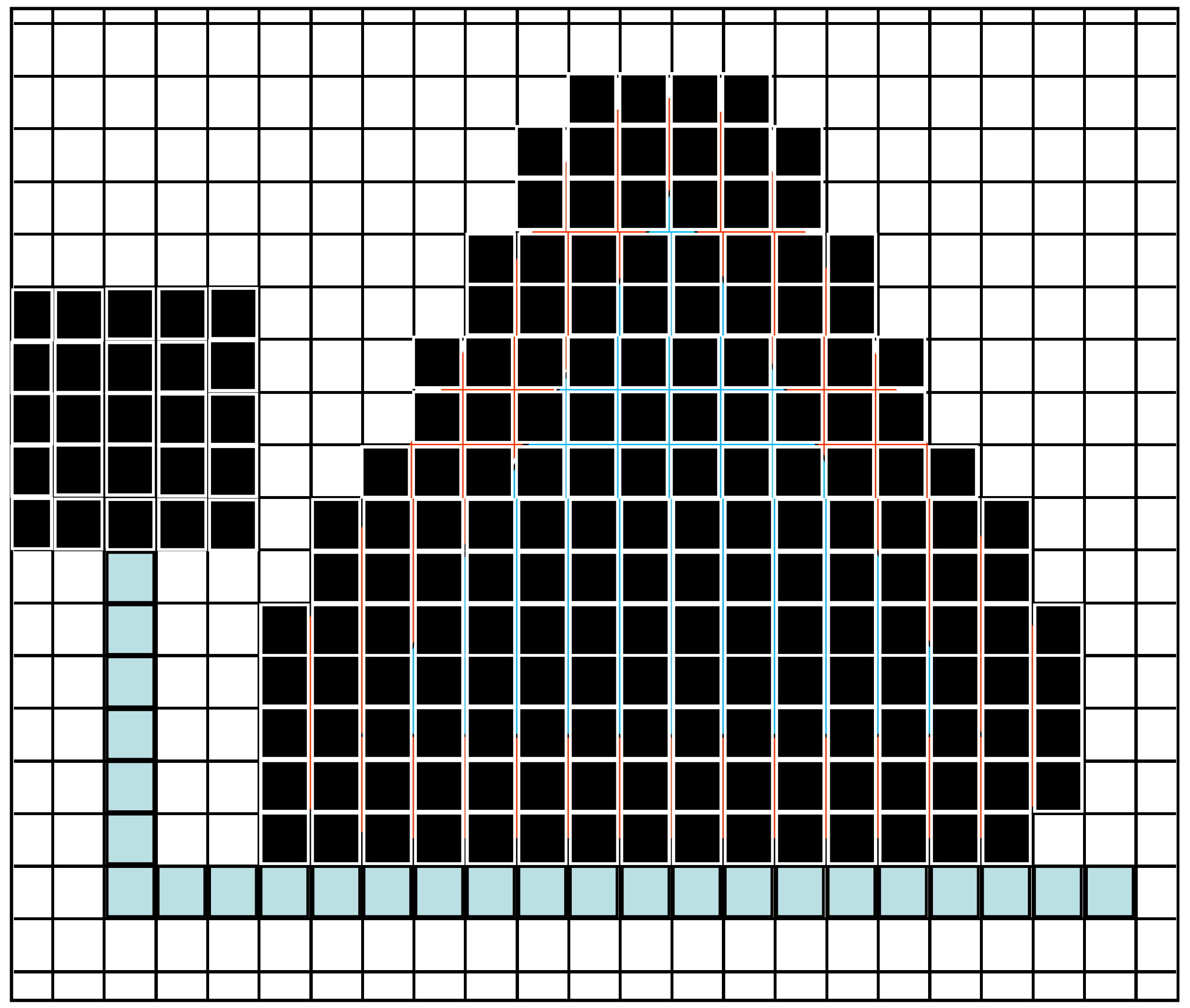




State-space Search Methods

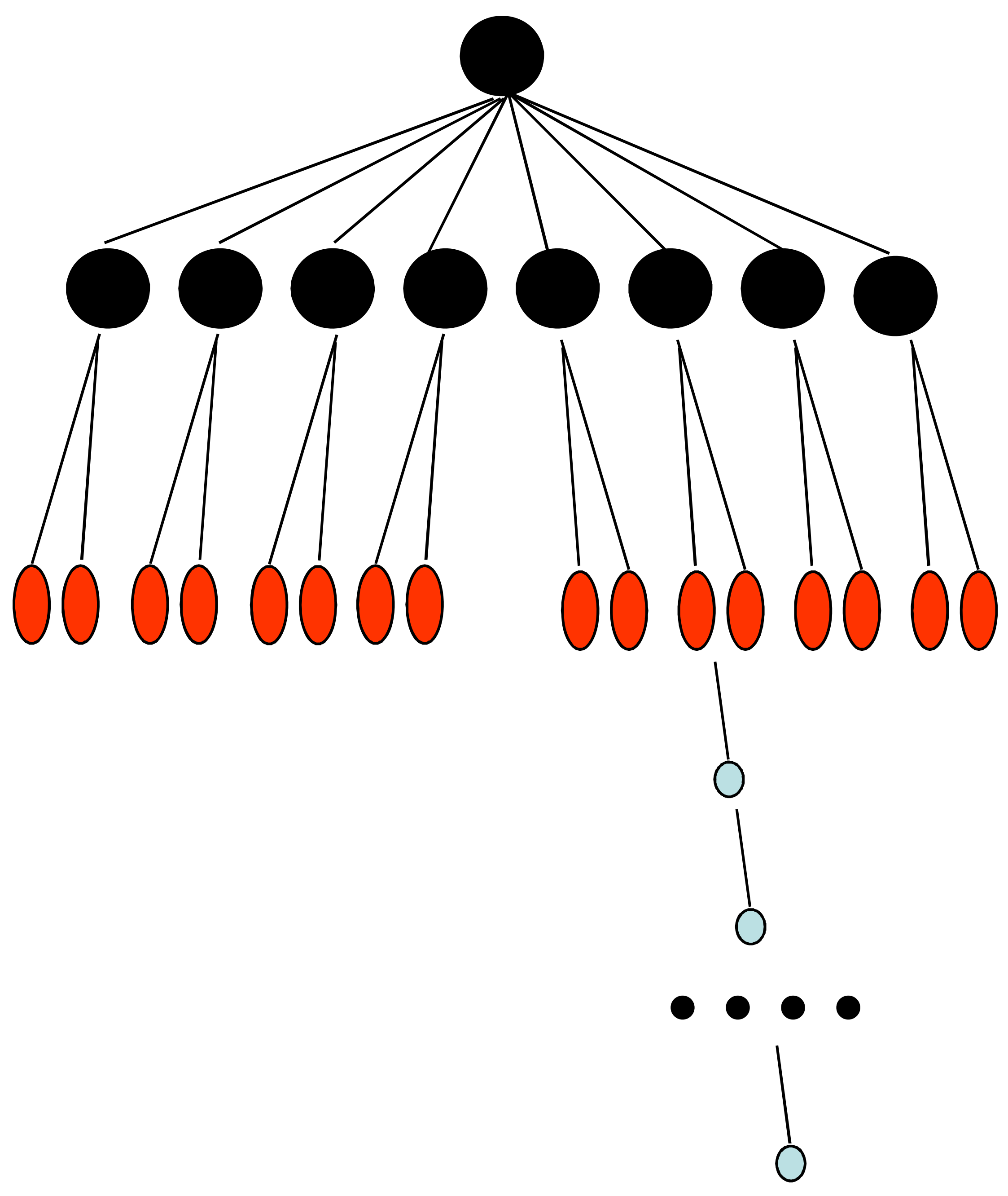
- State-space search amounts to finding a "Path" from the start state to the goal state.
- Optionally this can involve an "objective" function as well: minimize cost or maximize reward

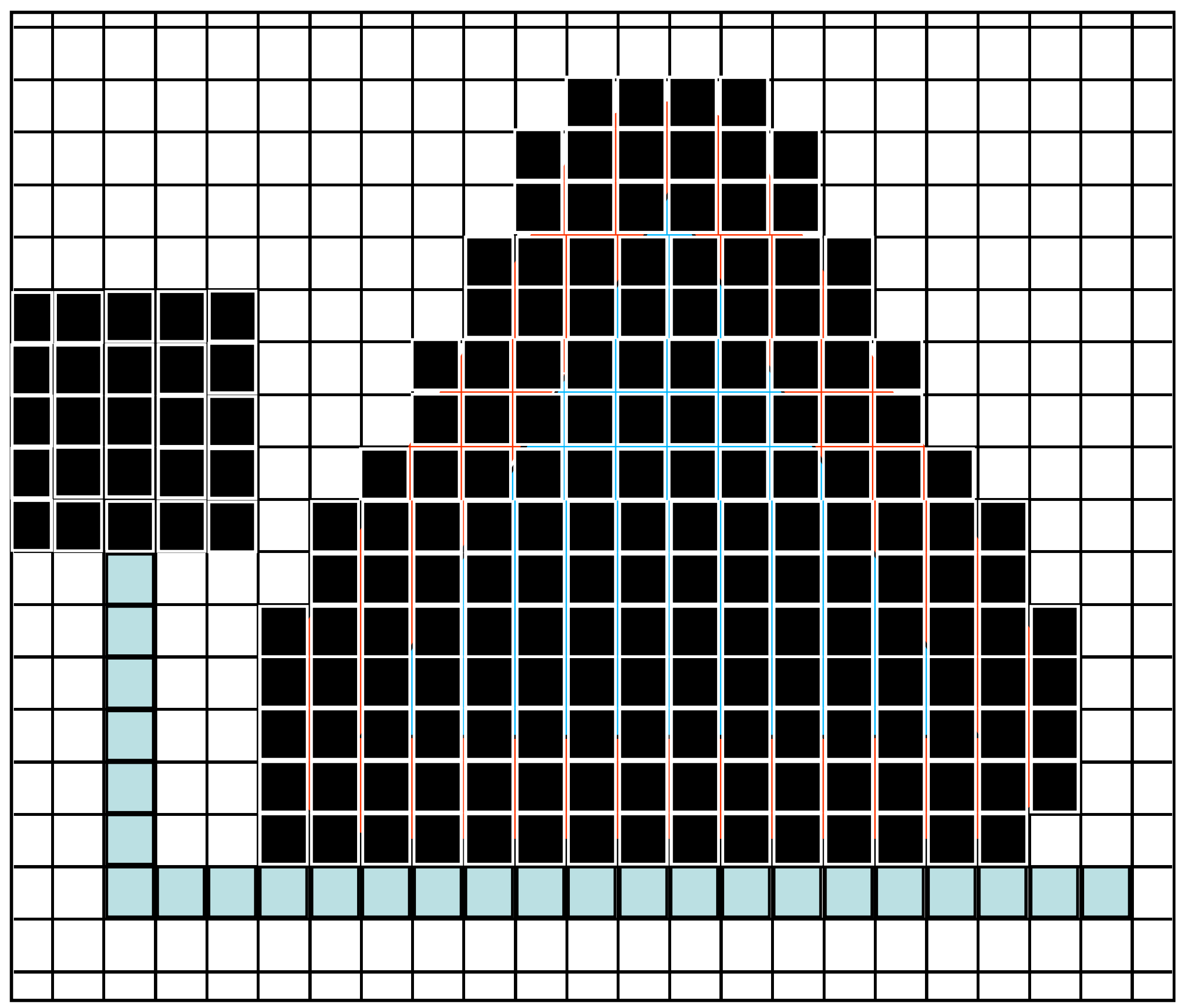




State-space Search Methods

- Which state-action pair to consider next?
  - **Shallowest next (Breadth-first search)**
    - Guarantees shortest path
    - but: storage intensive
  - **Deepest next (Depth-first search)**
    - Can use minimal storage
    - But: no optimality guarantee

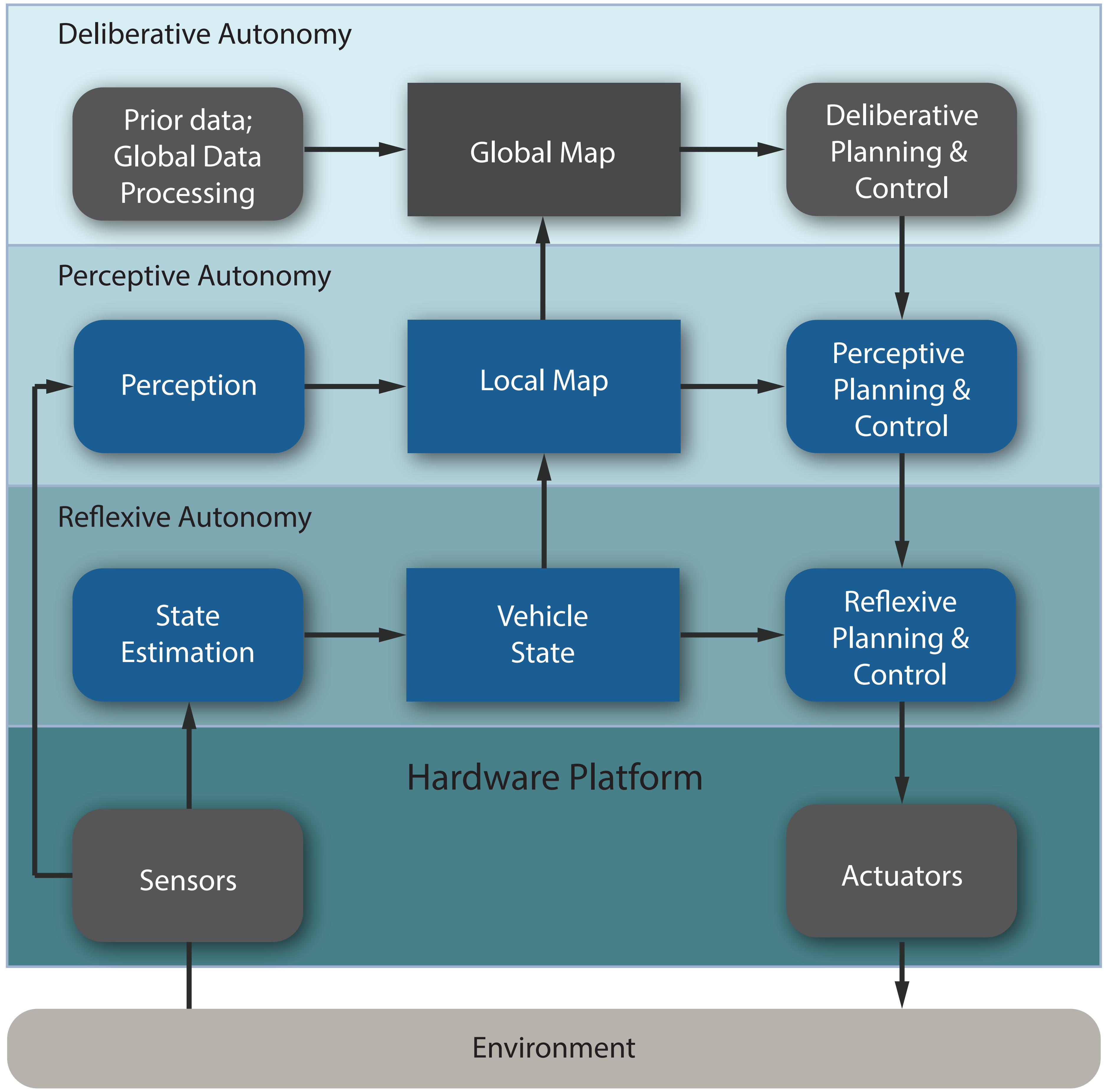




Open-loop vs Closed-loop Execution

- **Planning in a static world:** Planning provides state-action pairs to reach a goal, but typically does not provide a means to ensure the state-action pairs succeed in reaching the consecutive state.
  - **Closed-loop (low-level) controllers:** A closed-loop controller typically monitors the state of the system to ensure the state transitions are implemented as expected.

- **Planning in a dynamic world:** In many planning problems, the environment is not static - it changes unpredictably as the agent moves in the environment
  - **Policy/control design:** In those cases, we will design policies, as opposed to plans, that govern the entire planning process.



Key Takeaways

- **Autonomy** is at the core of AI: How can computers make decisions in the physical/social world interacting with an environment and/or other (AI/human) agents?
- **Planning** is the ability to find a "path" from a "start" configuration to a "goal" configuration, potentially optimizing an objective.
- **State-space models** allow the representation of (many) planning problems, where planning is reduced to "search" of a sequence of "state-action pairs" starting from a "start state" and reaching a "goal state".
- **Depth-first search** and **breadth-first search** are two simplest search methods, in principle allows us to solve any planning problem described by a state-space model.