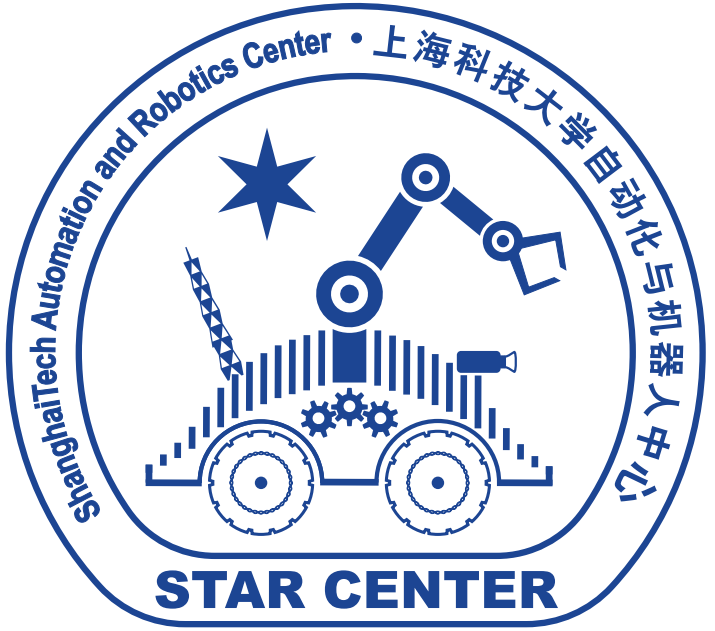
上海科技大学

ShanghaiTech University

CS289: **Mobile Manipulation Fall 2025**

Sören Schwertfeger

ShanghaiTech University



Outline

- Continue arm planning
- OMPL
- MoveIt
- Mobile Robotics

Rapidly Exploring Random Tree: RRT

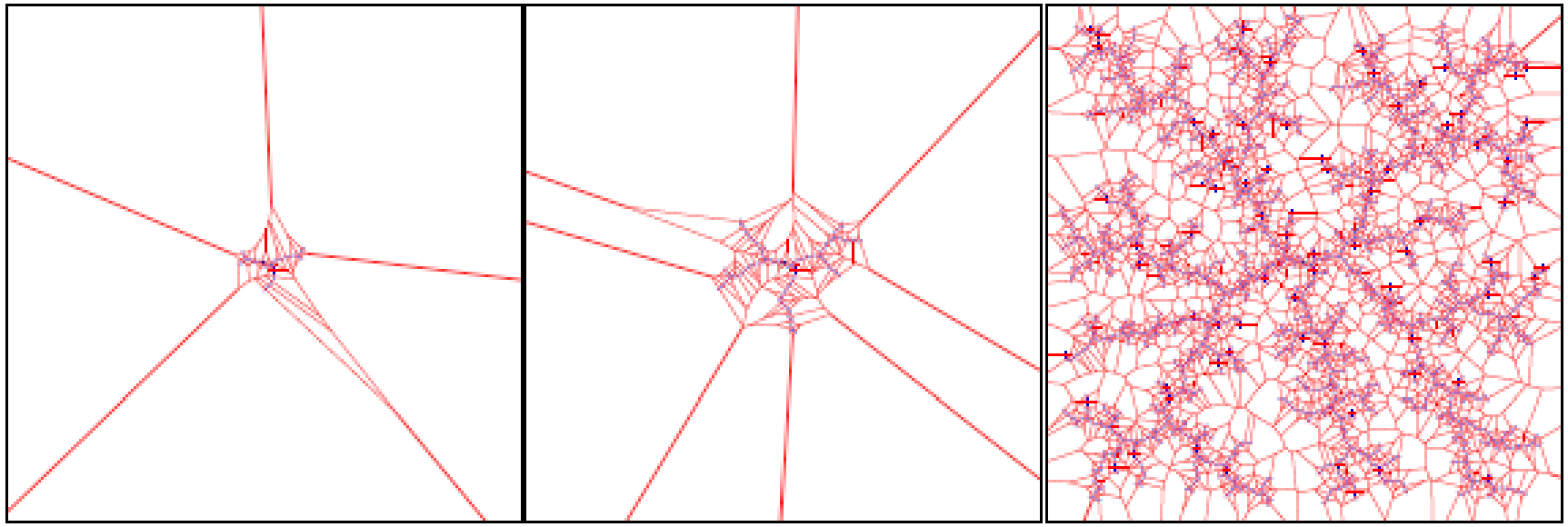
- Most famous/ widely used planner concept for Robot Manipulations

- Monte-Carlo based method with
- Bias towards “big empty space” (largest Voronoi region)

Kuffner, J. J., & LaValle, S. M. (2000, April). RRT-connect: An efficient approach to single-query path planning. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation.* IEEE.

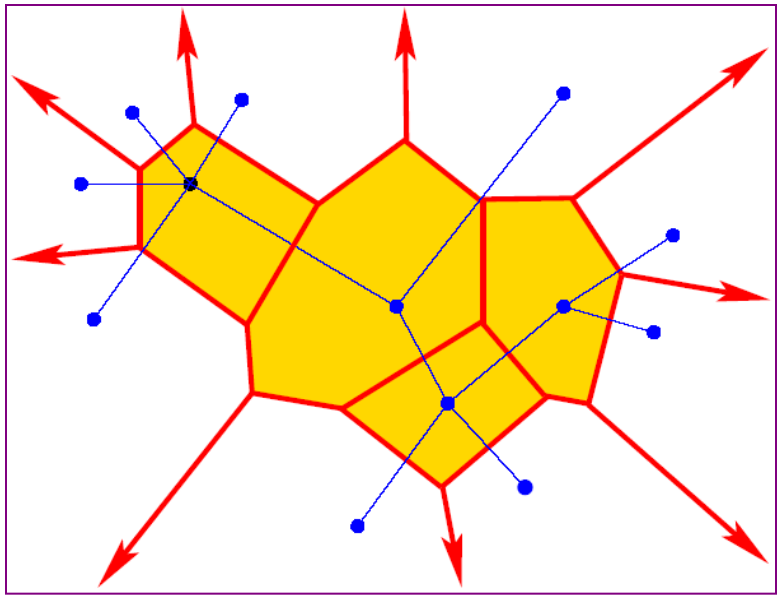
Animation from iteration 0 to 10,000

Why “Rapidly Exploring”?

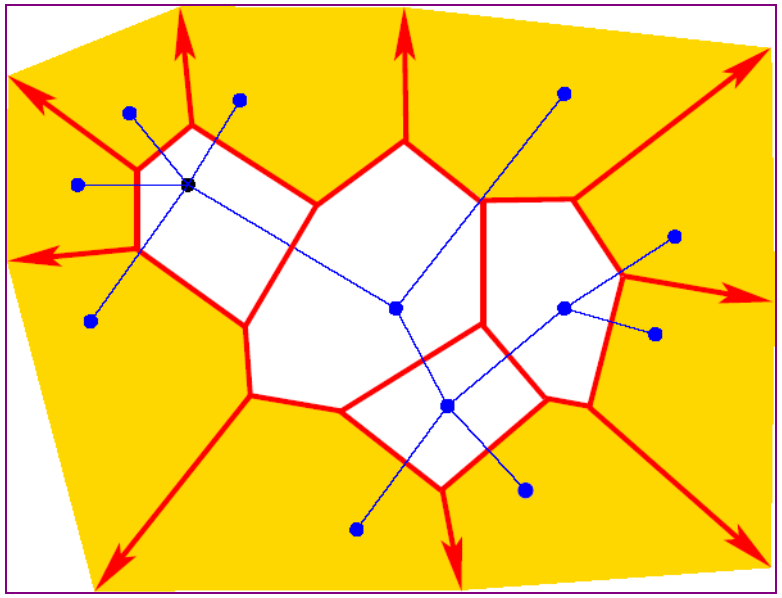


- What is the probability that a vertex will be extended?
  - Proportional to the area of its Voronoi region
- If just choose a vertex at random and extend, then it would act like random walk instead
  - Biased towards start vertex

Refinement vs. Expansion



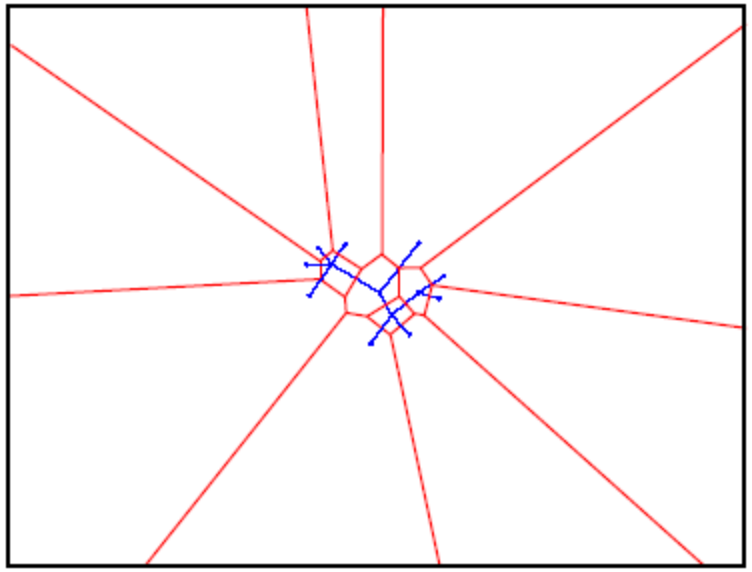
refinement



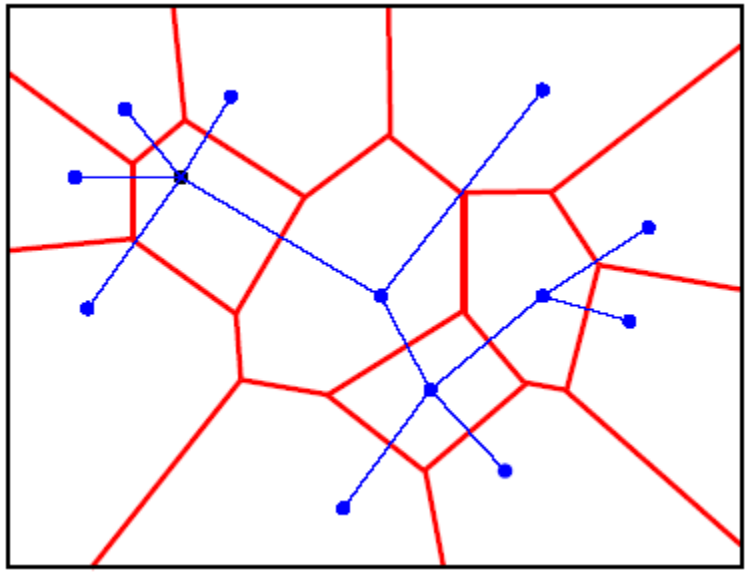
expansion

Where will the random sample fall? How to control the behavior of RRT?

Determining the Boundary



Expansion dominates



Balanced refinement and expansion

The tradeoff depends on the size of the bounding box

Problem

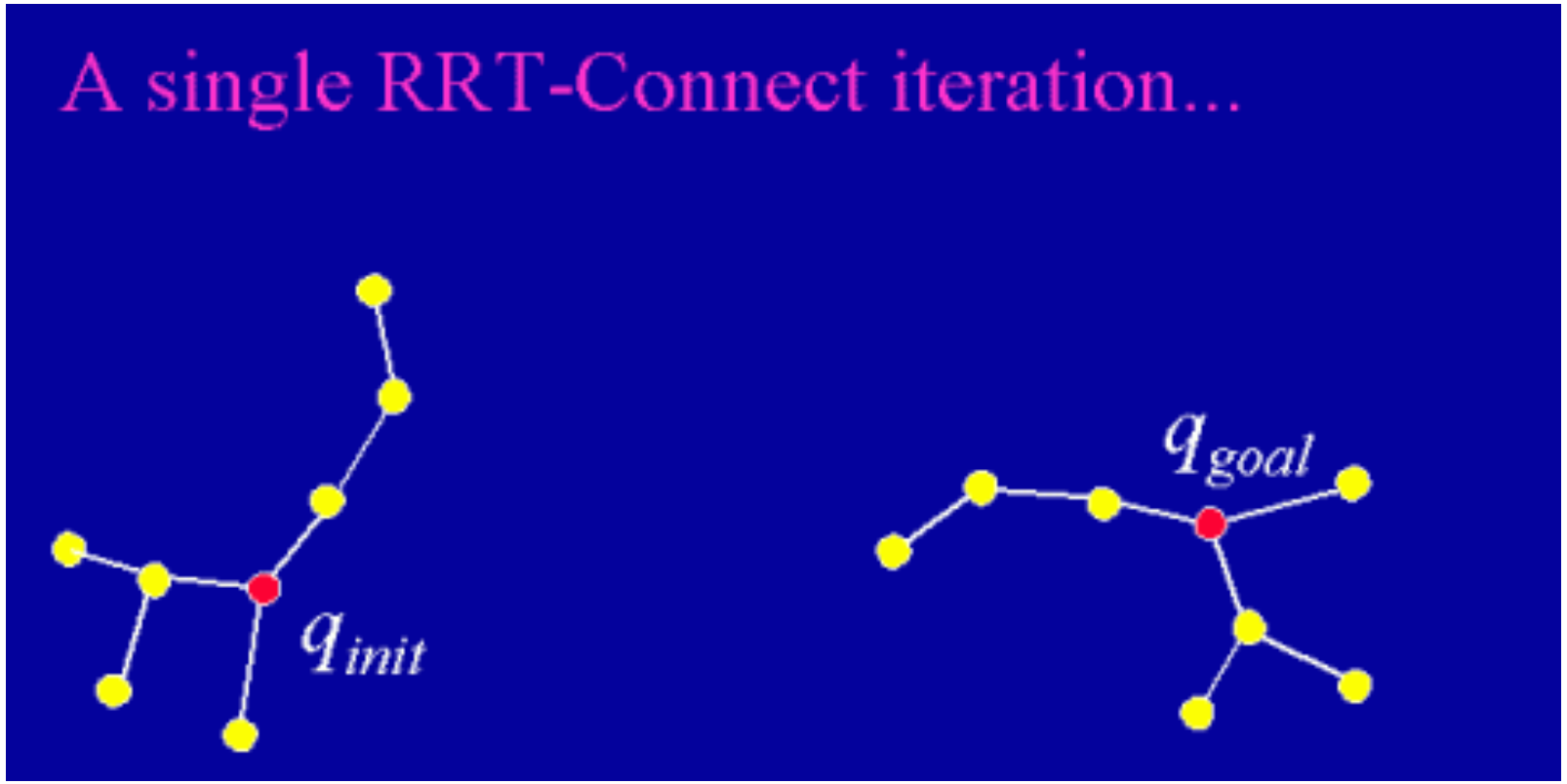
- Non-biased RRT explores in all directions
  - Not aimed at the goal
- Use a small percentage of targets to be the goal or its neighborhood

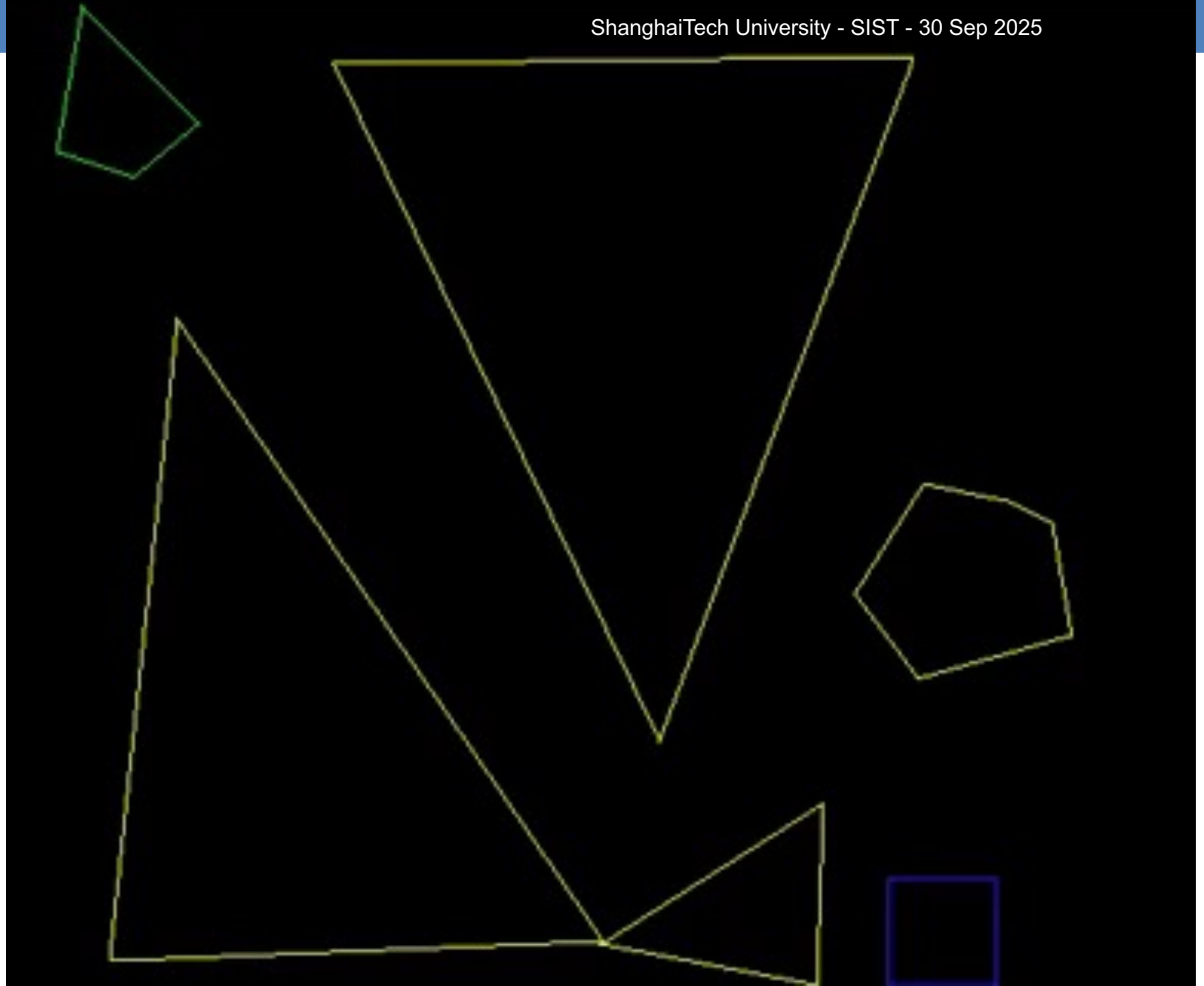
How Far to Extend?

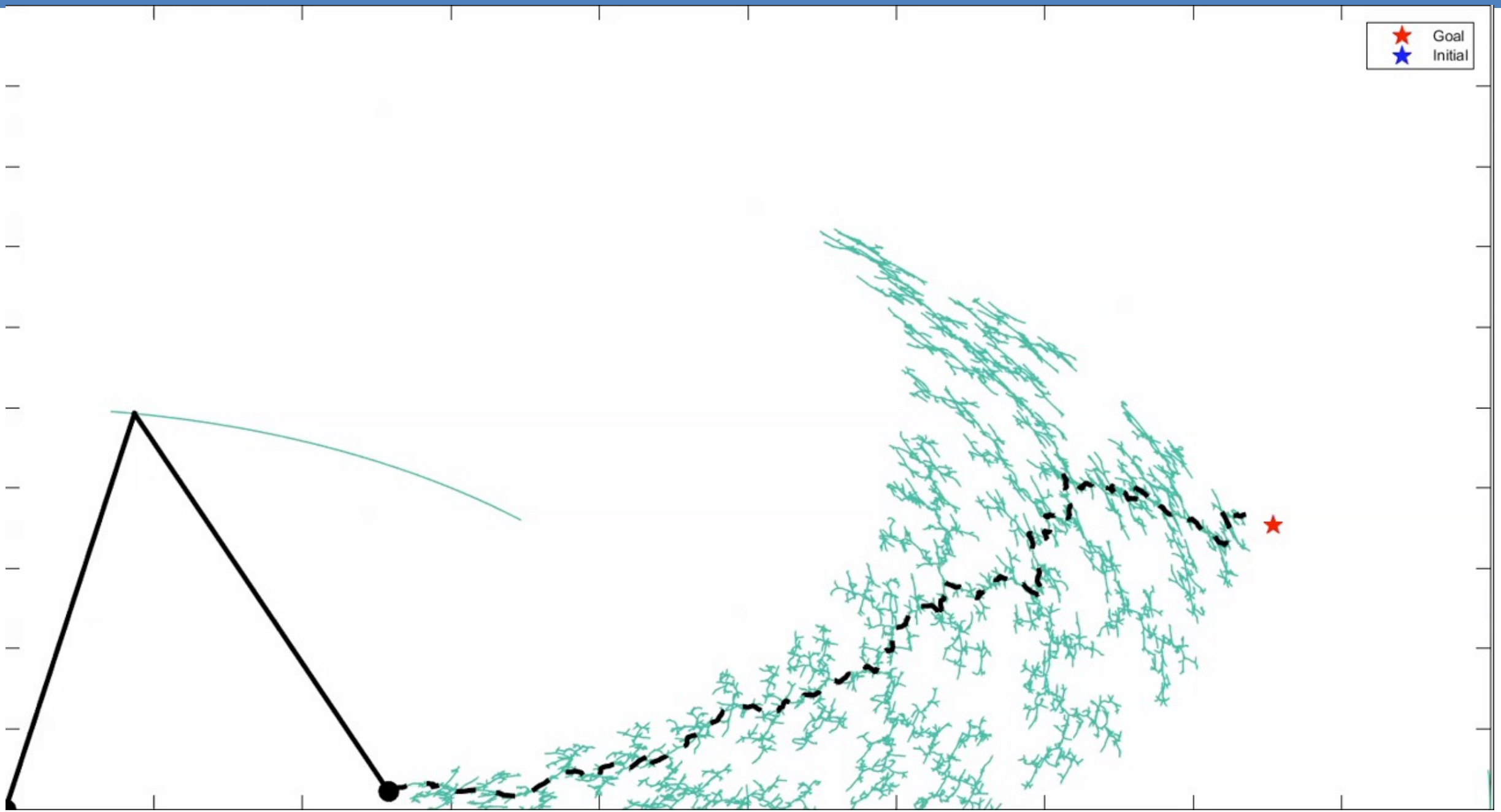
- Use distance measures from collision detection
  - Big steps when far away

RRT connect

- Grow two trees from start and goal – connect when close



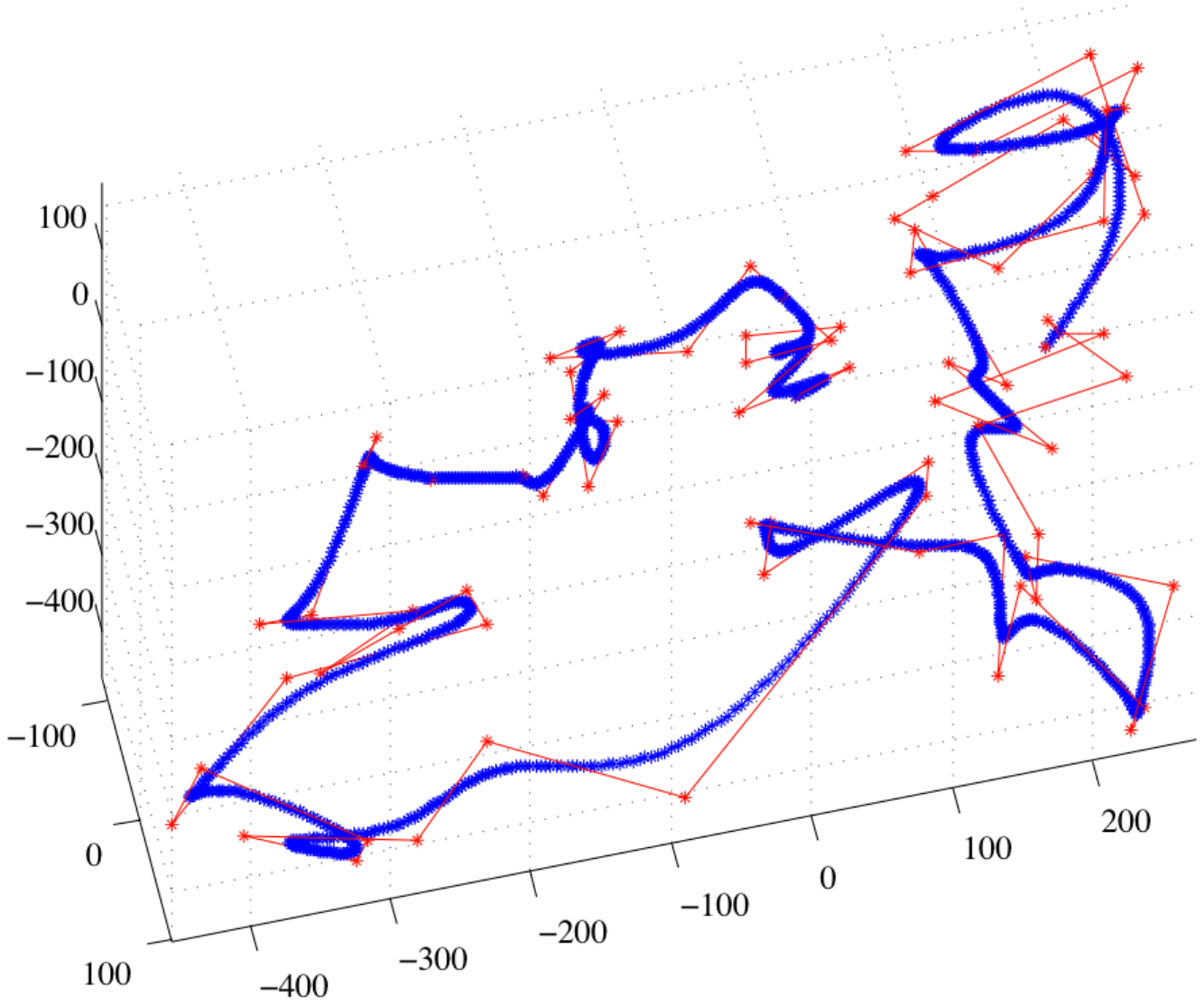



Goal
Initial

Path smoothing

- Path in tree not smooth ->
- Jerky motion and high forces ->
- Path smoothing needed

- E.g. use B-splines

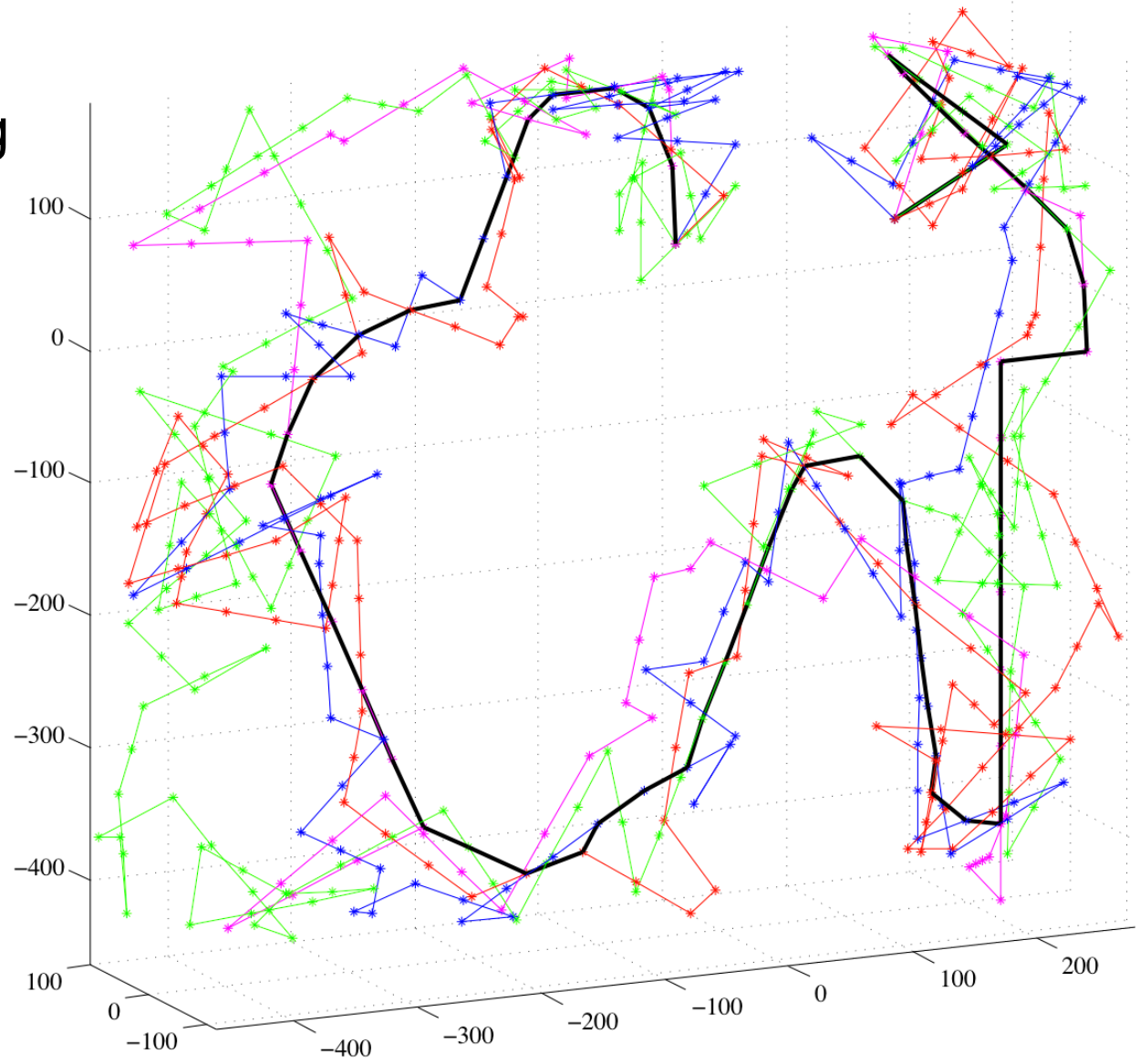


Images from this and next slide from:
https://ompl.kavrakilab.org/gallery.html#gallery_comp450

Path Shortening

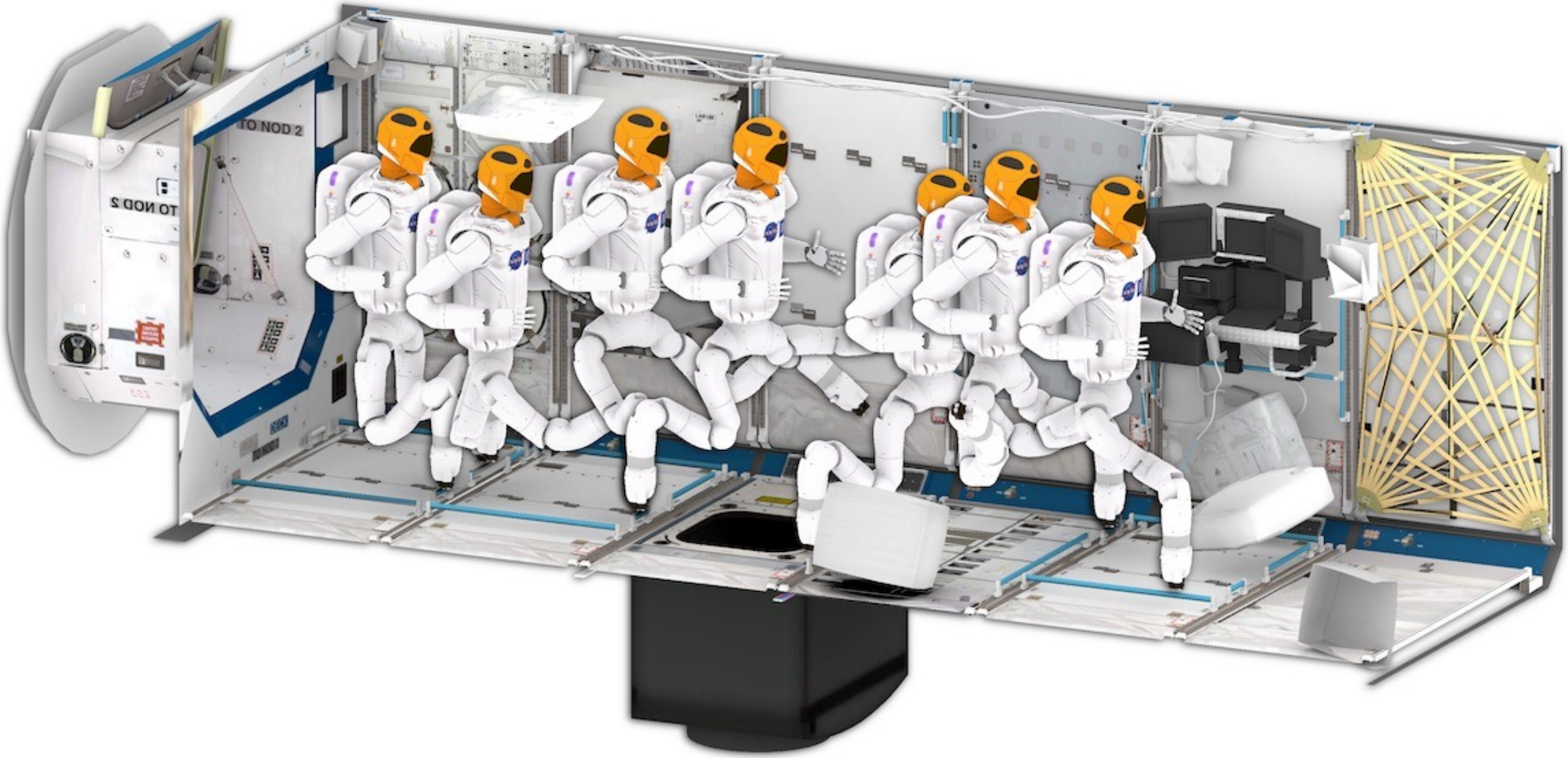
- Path generated by RRT often too long
- Optimizing/ shortening path desirable

- Example: path shortening using hybridization

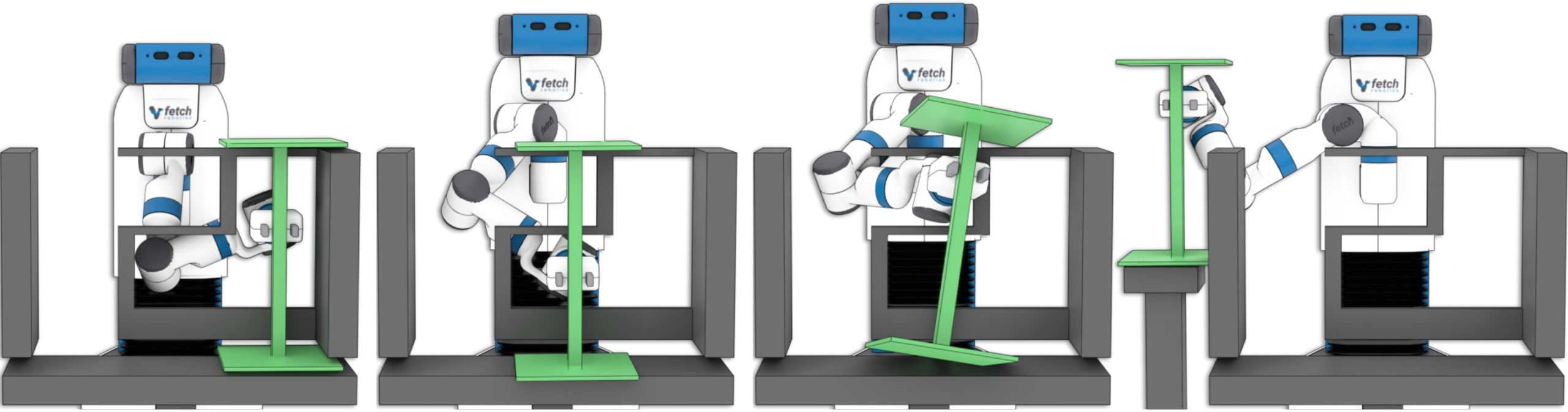


OMPL: Open Motion Planning Library

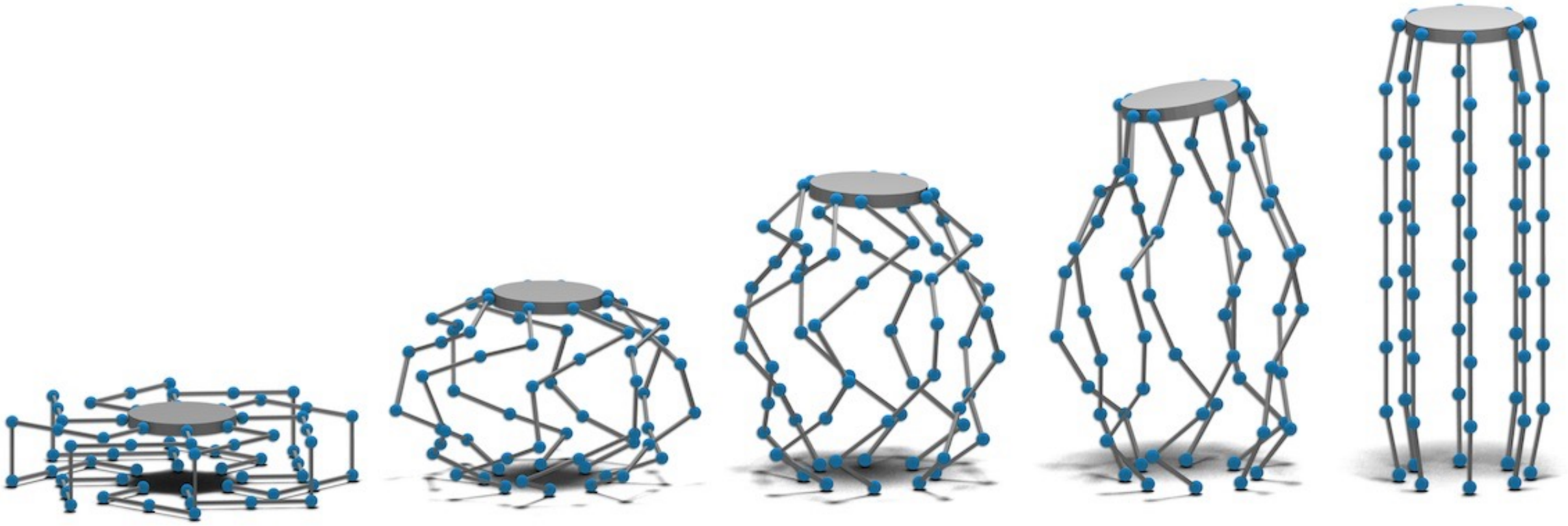
- https://ompl.kavrakilab.org/
- Used in many systems
  - Movit -> ROS; OpenRAVE; CoppeliaSim; MORSE; …
- Limited to motion planning algorithms ->
  - Collision detection; visualization; environment specification external, e.g. MoveIt!
- Written in C++ with python bindings
- Includes framework to Benchmark Planners -> select best planner for your problem/ robot! https://plannerarena.org/



OMPL is the default planning library in MoveIt and has been used for many robots. Here, OMPL is used to plan footsteps for NASA's Robonaut2 aboard the International Space Station.



Multi-modal Motion Planning
OMPL has been used in multi-modal planners to achieve complex robot behavior, such as removing this barbell object from a puzzle box.



Constrained Motion Planning

OMPL supports planning for kinematically constrained robots. This parallel manipulator has over 150 degrees of freedom, but feasible motions can still be computed in seconds.

OMPL Foundations (1)

- StateSampler
  - Uniform and Gaussian sampling of state space configurations
- NearestNeighbors
  - Perform nearest neighbor search along samples
  - E.g. Geometric Near-neighbor Access Trees (GNAT) (similar to kd-tree), linear searches
- StateValidityChecker
  - Check a single state for
    - Collision
    - Constraints

OMPL Foundations (2)

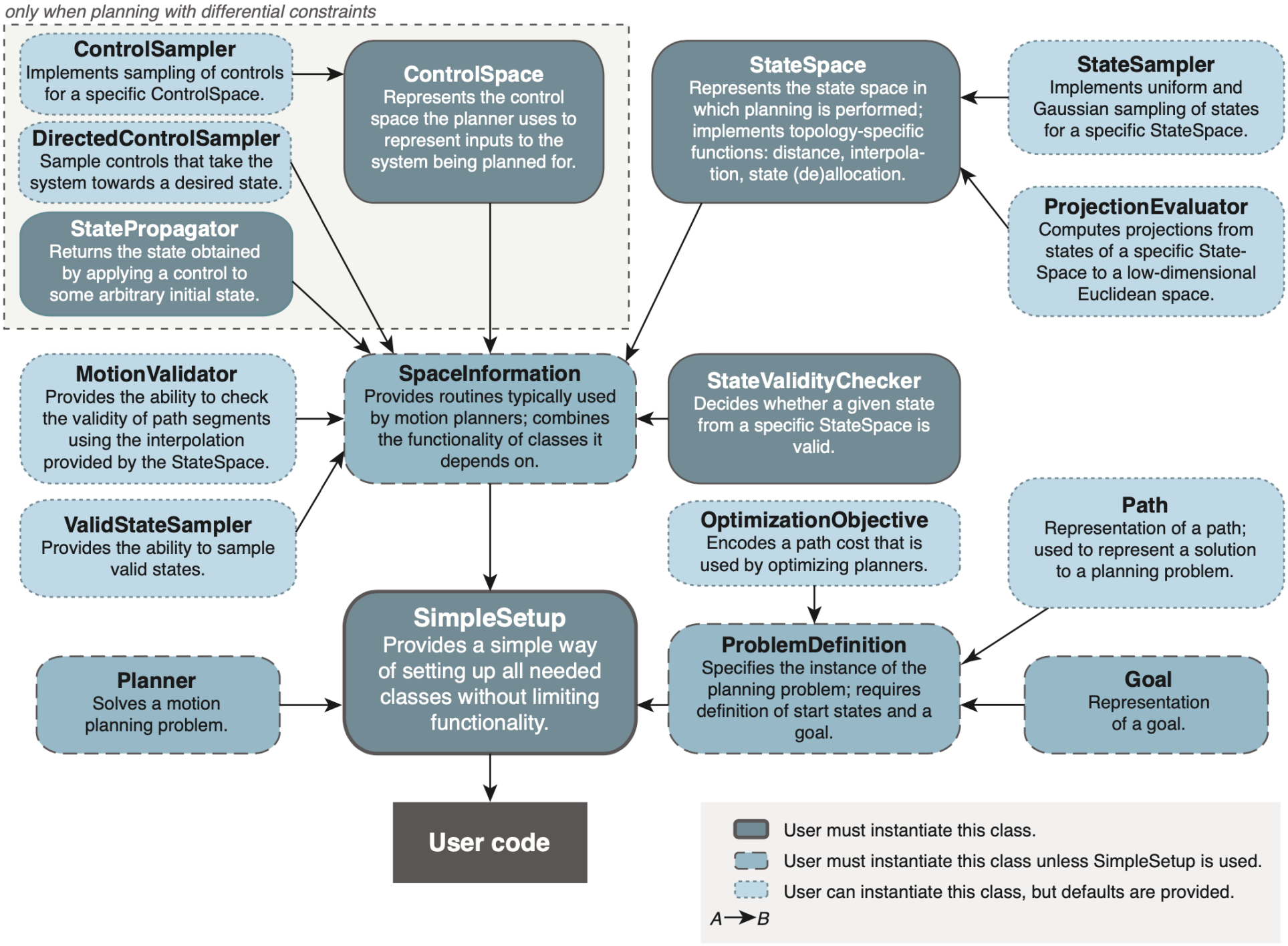
- MotionValidator
  - Check if motion between two states is valid
    - Collisions
    - Constraints
  - E.g.: DiscreteMotionValidator: interpolated movement between 2 states
    - Check finite number of states (using StateValidityChecker) between the 2 states
- OptimizationObjective
  - Some planners support optimization, e.g.: Minimize path length
- ProblemDefinition
  - Planning query with start, goal, optimizationObjectives
  - Goal can be single configuration or region surrounding a state

OMPL Foundations (3)

- StatePropagation
  - Given: state, control, duration
  - Calculate: new state
  - Used for control-based planning

- Steer
  - Also member of StatePropagator Class
  - Given: start and end state (close to each other!)
  - Calculate: needed control; duration; success
  - Used to save control commands in edge/ path/ trajectory

OMPL


only when planning with differential constraints
ControlSampler
Implements sampling of controls for a specific ControlSpace.
DirectedControlSampler
Sample controls that take the system towards a desired state.
StatePropagator
Returns the state obtained by applying a control to some arbitrary initial state.
ControlSpace
Represents the control space the planner uses to represent inputs to the system being planned for.
StateSpace
Represents the state space in which planning is performed; implements topology-specific functions: distance, interpolation, state (de)allocation.
StateSampler
Implements uniform and Gaussian sampling of states for a specific StateSpace.
ProjectionEvaluator
Computes projections from states of a specific StateSpace to a low-dimensional Euclidean space.
MotionValidator
Provides the ability to check the validity of path segments using the interpolation provided by the StateSpace.
SpaceInformation
Provides routines typically used by motion planners; combines the functionality of classes it depends on.
StateValidityChecker
Decides whether a given state from a specific StateSpace is valid.
ValidStateSampler
Provides the ability to sample valid states.
OptimizationObjective
Encodes a path cost that is used by optimizing planners.
Path
Representation of a path; used to represent a solution to a planning problem.
SimpleSetup
Provides a simple way of setting up all needed classes without limiting functionality.
Planner
Solves a motion planning problem.
ProblemDefinition
Specifies the instance of the planning problem; requires definition of start states and a goal.
Goal
Representation of a goal.
User code
User must instantiate this class.
User must instantiate this class unless SimpleSetup is used.
User can instantiate this class, but defaults are provided.
A→B

OMPL Available Planners: Geometric Planners (1)

- Only geometric and kinematic constraints

- Multi-query planners
  - First build roadmap, afterwards plan quickly for many queries
  - Probabilistic Roadmap Methods (PRM): LazyPRM; PRM*; LazyPRM*
  - SPArse Roadmap Spanner algorithm (SPARS); SPARS2

- Single-query planners
  - RRTs: RRT Connect; RRT*; Lower Bound Tree RRT; Parallel RRT; Lazy RRT; …
  - Search Tree with Resolution Independent Density Estimation (STRIDE)
  - Path-Directed Subdivision Trees (PDST)
  - Fast Marching Tree algorithm (FMT∗); Bidirectional FMT∗
  - …

OMPL Available Planners: Geometric Planners (2)

- Multi-level planners
  - Multiple levels of abstraction
  - Rapidly-exploring Random Quotient space Trees (QRRT); QRRT*
  - Quotient-Space Roadmap Planner (QMP): generalized RPM; QMP*

- Optimizing planners:
  - Have a star * after the name
  - Provide optimality guarantee, even though sampling based
  - Optimization objectives:
    - shortest path;
    - Minimum path clearance;
    - Minimize mechanical work;
    - General state cost integral; (user-based optimization)

OMPL Available Planners: Control-based Planners

- Differential constraints -> need control-based planner!
- State propagation instead of interpolation for motion/ edge

- RRT
- Sparse Stable RRT
- Expansive Space Trees (EST)
- Kinodynamic Planning by Interior-Exterior Cell Exploration (KPIECE)
- Path-Directed Subdivision Trees (PDST)

OMPL Available Planners: Multilevel-based Planners

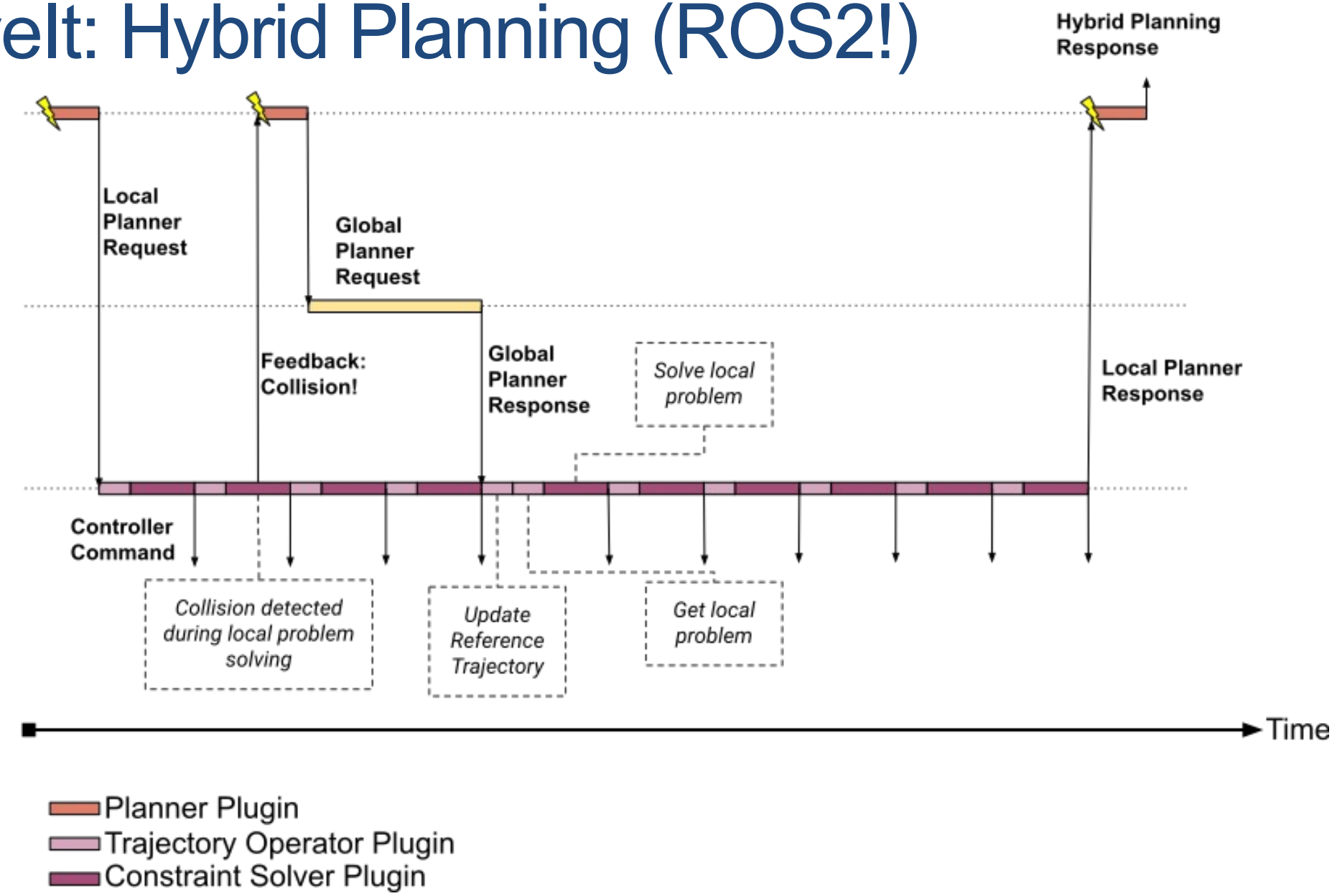
- Very high-dimensional state space ->
- Use Multi-level abstractions to simplify state space ->
- Find solutions faster

- Rapidly-exploring Random Quotient space Trees (QRRT)
- QRRT*
- Quotient-Space Roadmap Planner (QMP)
- QMP*
- Sparse Quotient space roadmap planner (SPQR)*

MoveIt ROS 2 Planners

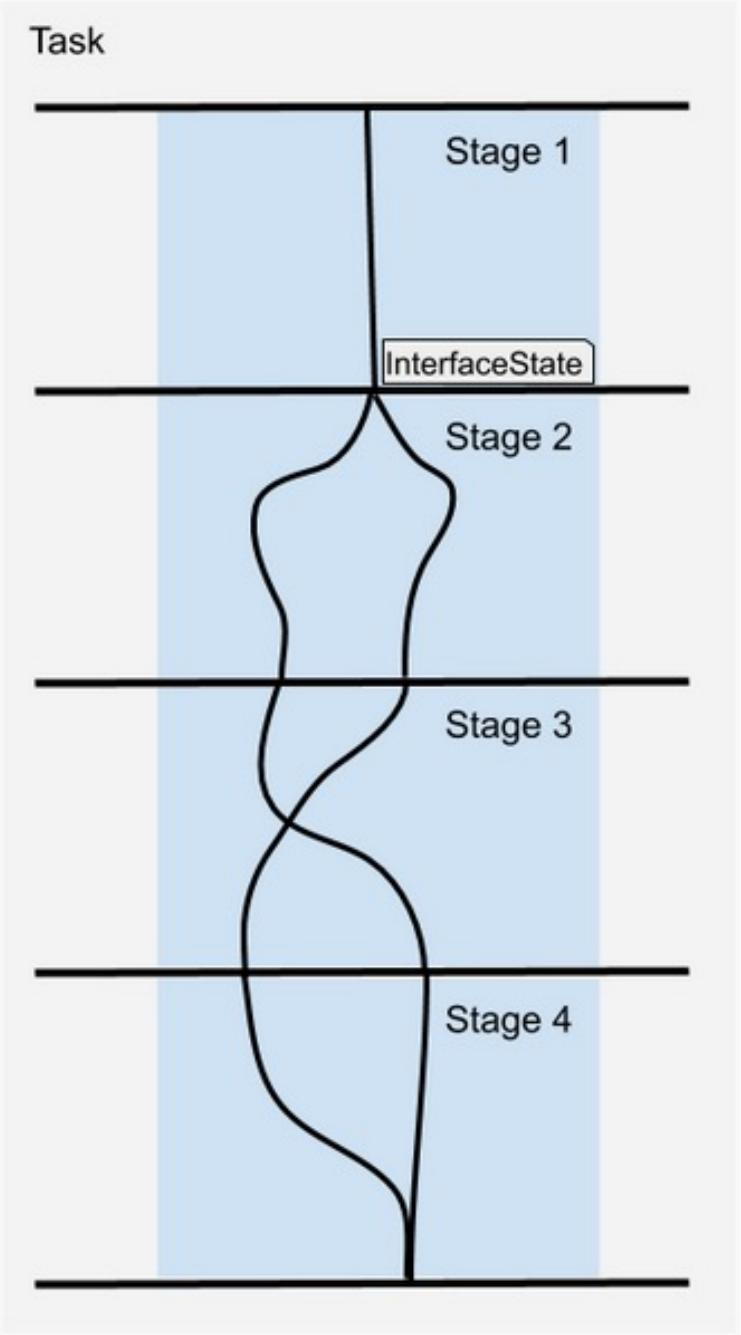
- OMPL
- Pilz Industrial Motion Planner
  - Deterministic generator for circular and linear motions
  - Supports blending multiple motion segments together using a MoveIt capability.
- Stochastic Trajectory Optimization for Motion Planning (STOMP)
  - Optimization-based planner based on “Policy Improvement with Path Integrals” (partially supported)
- Search-Based Planning Library (SBPL)
  - Search based planning in discrete space (work in progress)
- Covariant Hamiltonian Optimization for Motion Planning (CHOMP)
  - gradient-based trajectory optimization
  - covariant gradient and functional gradient approaches
  - motion planning algorithm based entirely on trajectory optimization
  - Start with infeasible, naïve trajectory -> use optimization to pull out of collision (work in progress)

MoveIt: Hybrid Planning (ROS2!)

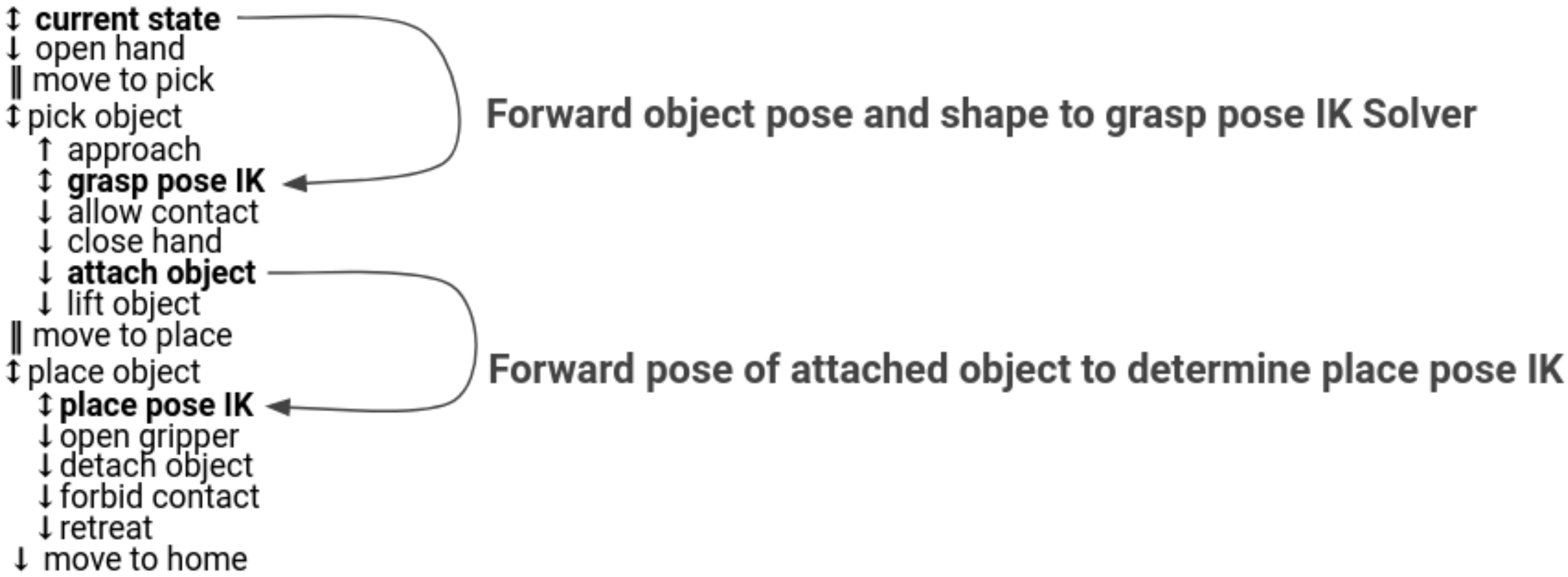


MoveIt Task Constructor MTC in ROS2

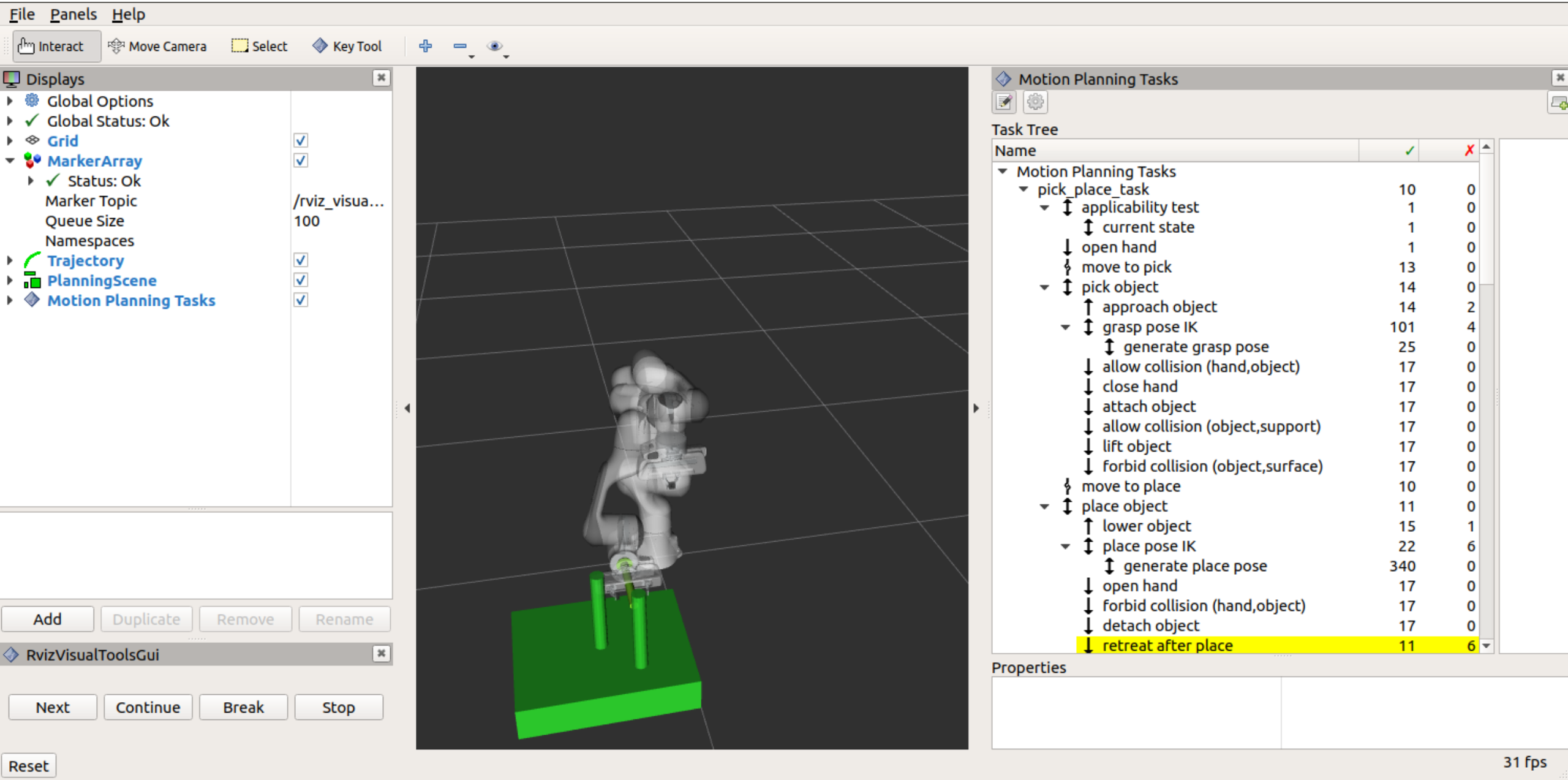
- Break down complex planning tasks to multiple interdependent subtasks
- MTC uses Moveit to solve the subtasks
- Stages:
  - Generators: no input; compute results and send in both directions; e.g. GeneratePose (e.g. current or goal)
  - Propagators: get solution, do something, transmit result; e.g. Move Relative
  - Connectors: get inputs but do not provide output: solve for feasible trajectory between the start and goal states
- Wrappers: encapsulate another stage to modify or filter the results
- Parallel containers: combine a set of stages to allow planning alternate solutions



MTC: Pick and Place



MTC: Pick and Place



/root/ws_moveit/install/moveit2_tutorials/share/moveit2_tutorials/launch/mtc.rviz* - RViz

File Panels Help

Interact | Move Camera | Select | Focus Camera | Measure | 2D Pose Estimate | 2D Goal Pose | Publish Point

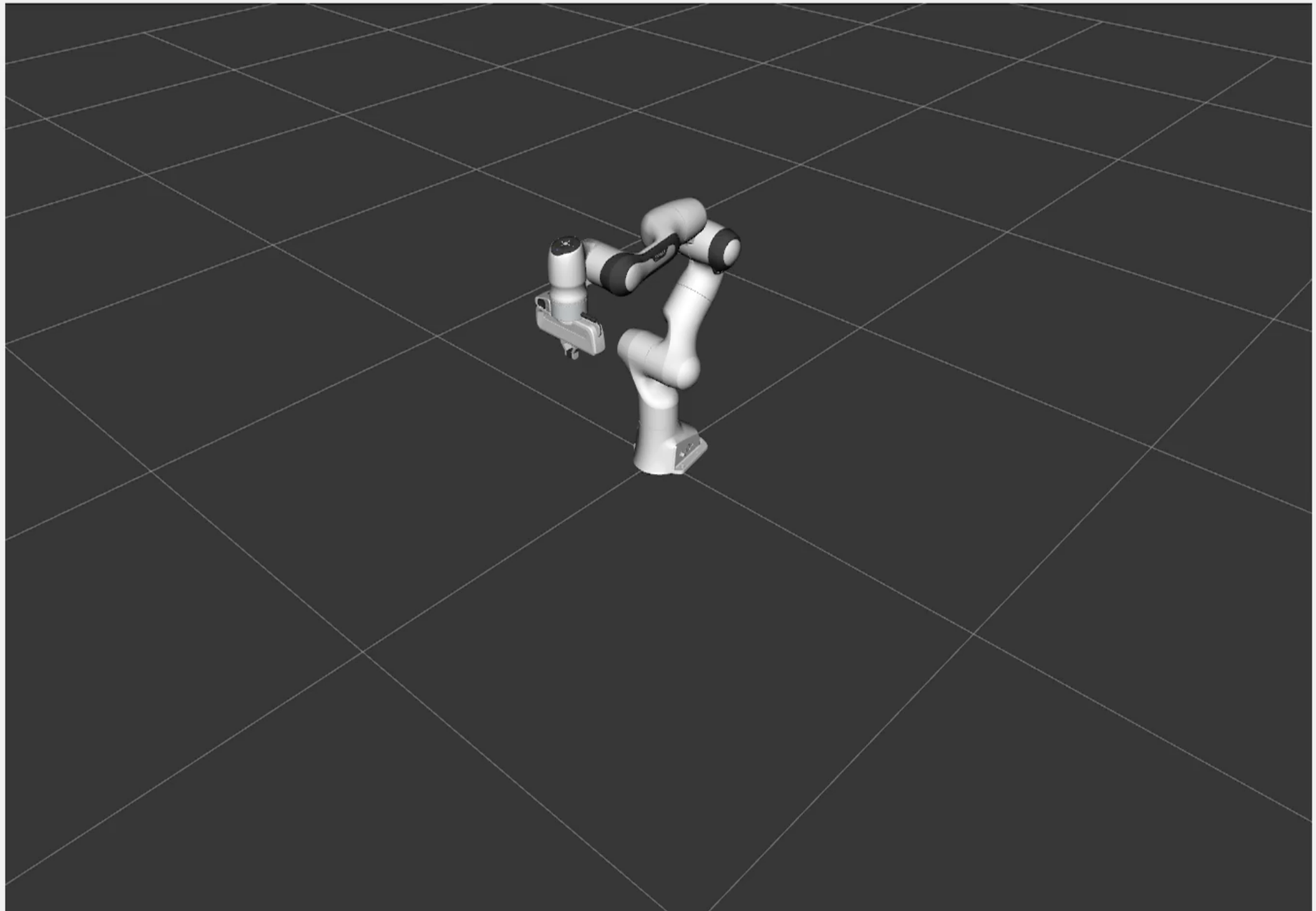
Motion Planning Tasks

Exec

Task Tree

name	✓	✗	time
Motion Planning Tasks			

Properties



Learning-based Motion Planning

- Not (by default) available in MoveIt (yet)

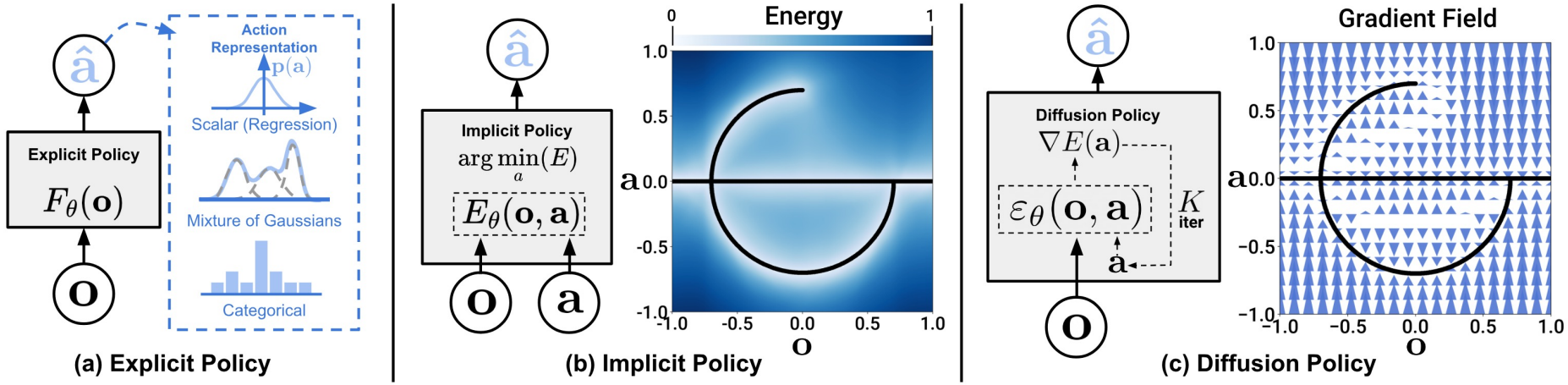
- Supervised methods (30+ cited)
  - End-to-end: sensor data to motion …
  - Module replacement: integrate with/ parallel to classical methods
    - E.g. improve RRT; Predict trajectory; improve collision checking

- Reinforcement Learning based Motion Planning (30+ cited)
  - value-based, policy-based, and actor-critic (AC)
  - Also end-to-end and module-replacement

Wang, J., Zhang, T., Ma, N., Li, Z., Ma, H., Meng, F., & Meng, M. Q. H. (2021). A survey of learning-based robot motion planning. *IET Cyber-Systems and Robotics*, *3*(4), 302-314. https://ietresearch.onlinelibrary.wiley.com/doi/pdfdirect/10.1049/csy2.12020

Learning-based Approaches

- Use Neural Networks to control/ plan arm motion E.g.:
- Visuomotor Policy Learning via Action Diffusion
  - generate robot behavior by representing a robot's visuomotor policy as a conditional denoising diffusion process https://diffusion-policy.cs.columbia.edu/
  - Transformer-based or CNN-based
  - predict robot arm's end-effector pose trajectory



Learning-based Approaches

- Use Neural Networks to control/ plan arm motion E.g.:

- HumanPlus Humanoid Shadowing and Imitation from Humans

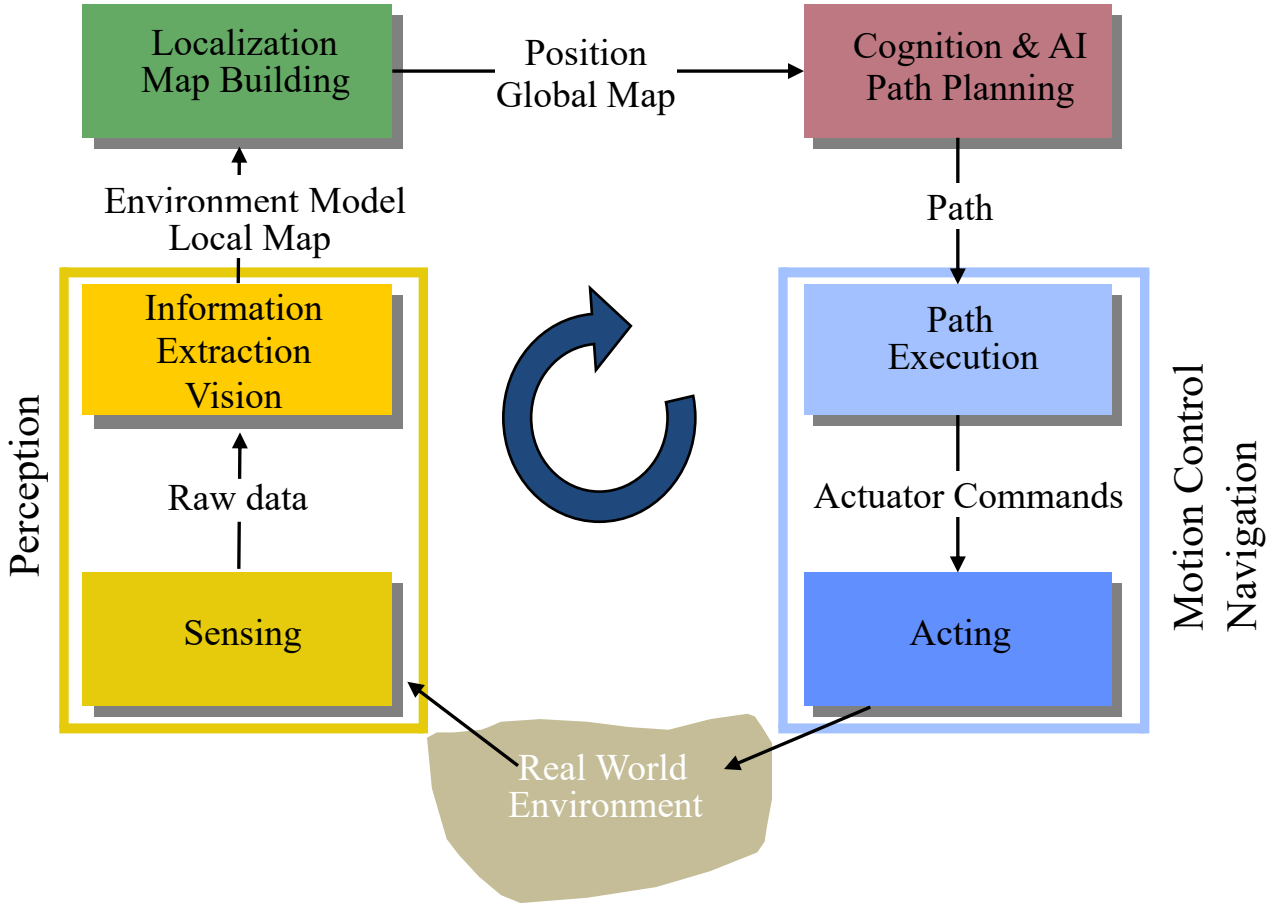
  - input: RGB & last action & robot proprioception info;
  - output: next action
  - using transformer-based network to predict the trajectory of each joint's pose.
  - Train via shadowing human operators

  - https://humanoid-ai.github.io/


HumanPlus: Autonomous
Stanford
University

MOBILE ROBOTICS

General Control Scheme for Mobile Robot Systems



With material from Roland Siegwart and Davide Scaramuzza, ETH Zurich

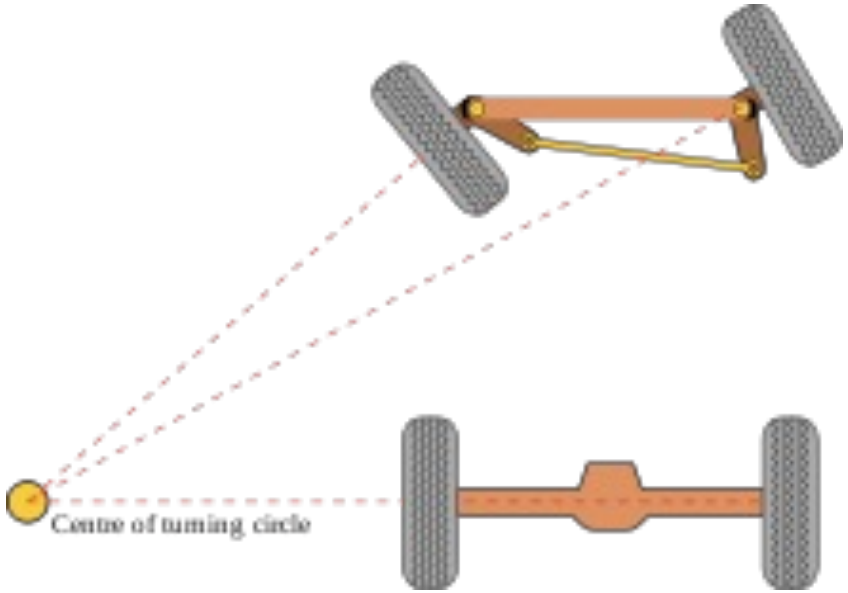
Mobile Robot Kinematics

- Aim
  - Description of mechanical behavior of the robot for *design* and *control*
  - Similar to robot manipulator kinematics
  - However, mobile robots can move unbound with respect to its environment
    - there is no direct way to measure the robot's position
    - Position must be integrated over time
    - Leads to inaccuracies of the position (motion) estimate
      -> the number 1 challenge in mobile robotics

Differential Drive Robots

Ackermann Robot

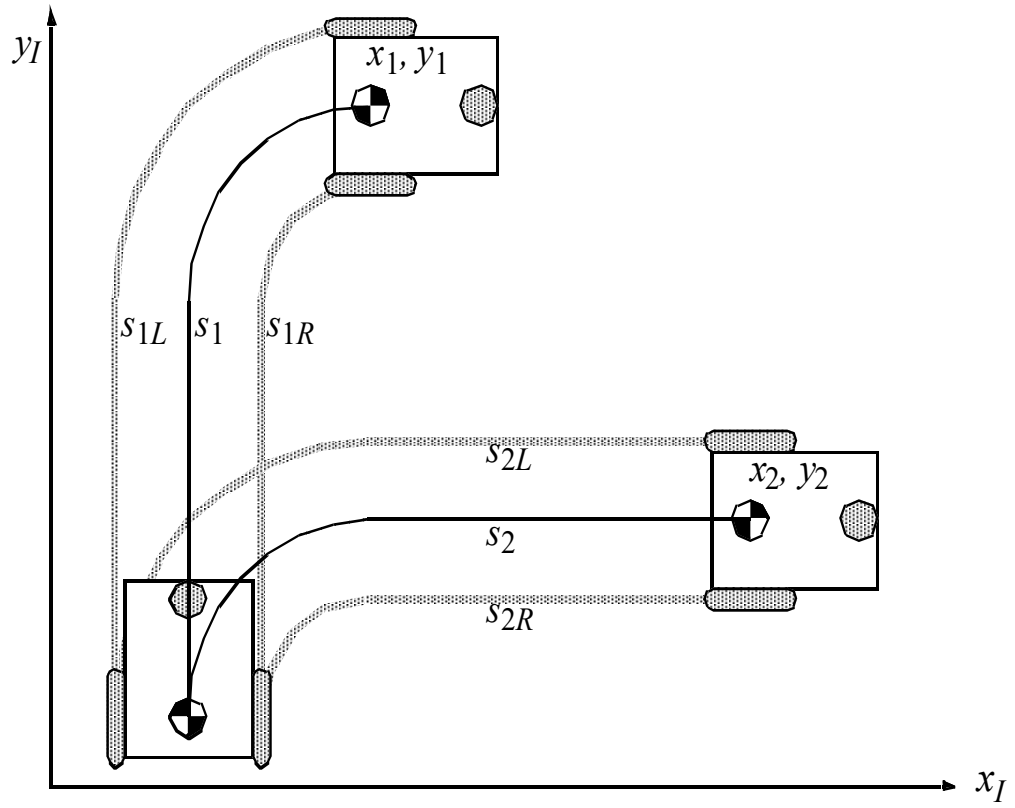
- No sideways slip than differential drive during turning ☺
- Cannot turn on the spot ☹



Mobile Robot Kinematics: Non-Holonomic Systems

$s_1 = s_2$; $s_{1R} = s_{2R}$; $s_{1L} = s_{2L}$

but: $x_1 \neq x_2$; $y_1 \neq y_2$



- Non-holonomic systems
  - differential equations are not integrable to the final pose.
  - the measure of the traveled distance of each wheel is not sufficient to calculate the final position of the robot. One has also to know how this movement was executed as a function of time.

MAPS & MAPPING

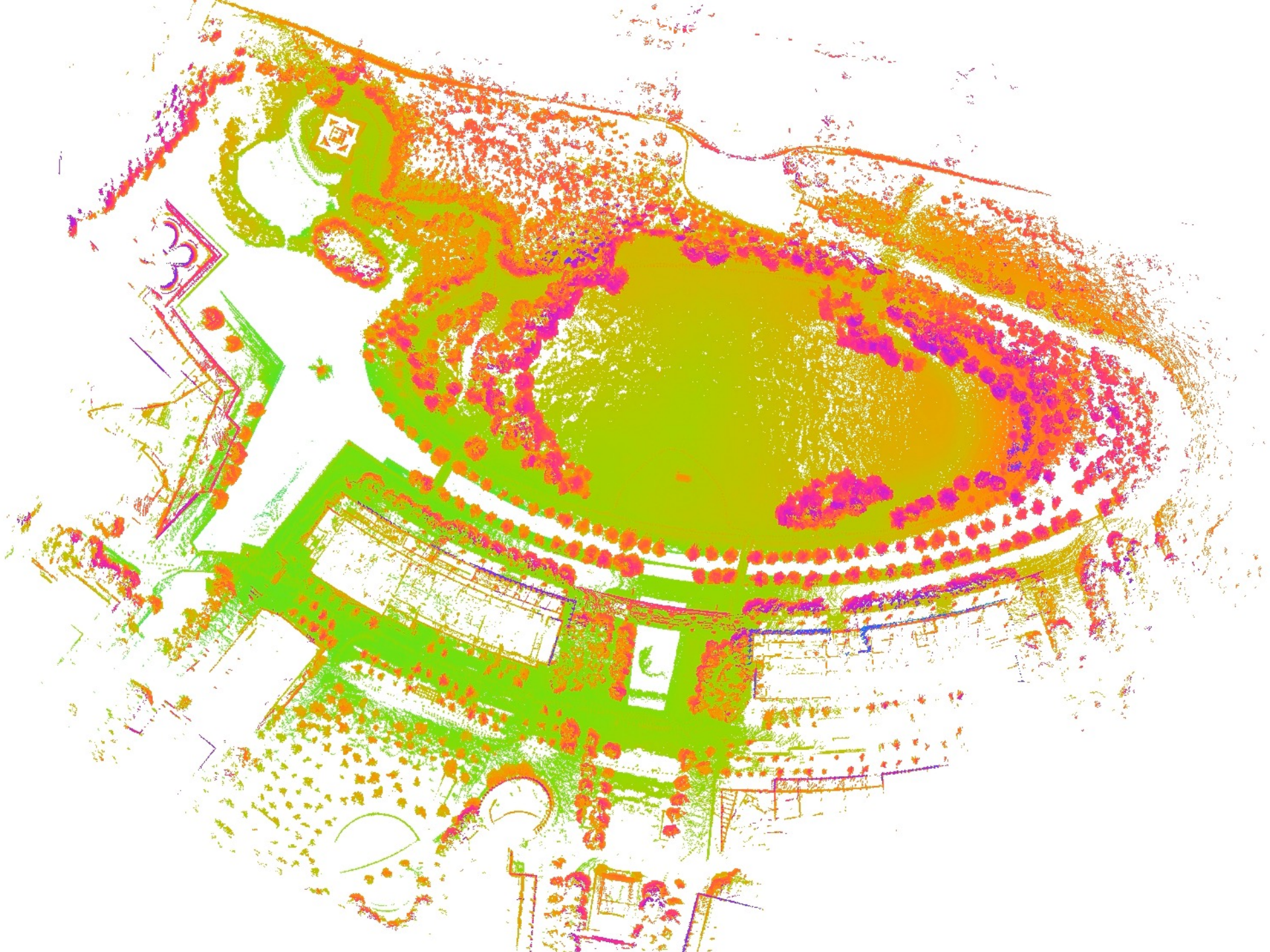
Map Representation: what is “saved” in the map

- Points (surface of objects, buildings): 2D or 3D
  - What: x,y or x,y,z coordinates;
    Optional: intensity; maybe RGB; maybe descriptor; temperature; …
  - From range sensors (laser, ultrasound, stereo, RGB-D): dense
  - From cameras (structure from motion; feature points): sparse
  - Variant: kd-tree
- Grid-map: 2D or 3D
  - Option: probabilistic grid map
  - Option: elevation map
  - Option: cost map
  - Option: Truncated Signed Distance Field
  - Option: Normal Distributions Transform (NDT)
  - Variant: Quad-tree; Oct-tree
- Higher-level Abstractions
  - Lines; Planes; Mesh
  - Curved: splines; Superquadrics
- Semantic Map
  - Assign semantic meaning to entities of a map representation from above
  - E.g. wall, ceiling, door, furniture, car, human, tree, …
- Topologic Map
  - High-level abstraction: places and connections between them
- Hierarchical Map
  - Combine Maps of different scales. E.g.:
  - Campus, building, floor
- Pose-Graph Based Map
  - Save (raw) sensor data in graph, annotated with the poses; generate maps on the fly
- Dynamic Map
  - Capture changing environment
- Hybrid Map
  - Combination of the above

RGBD-Inertial Trajectory Estimation and Mapping for Ground Robots

Zeyong Shan, Ruijian Li and Sören Schwertfeger

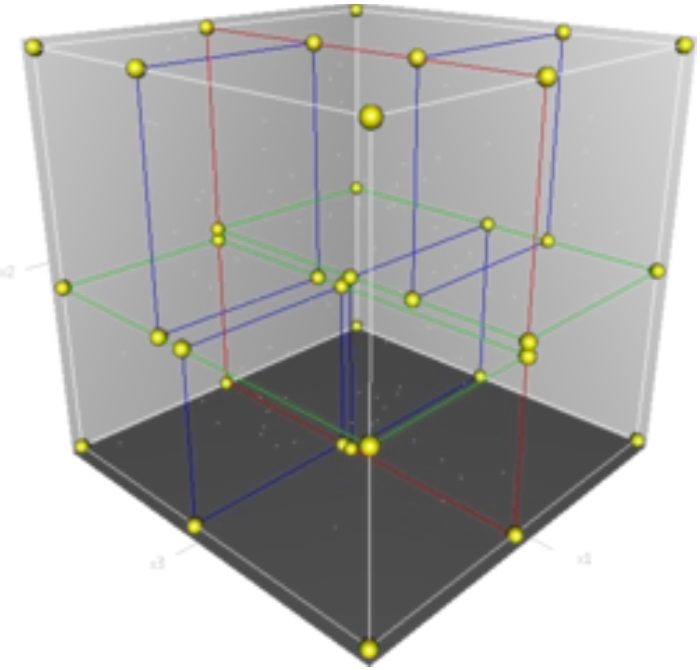
Source code: https://github.com/STAR-Center/VINS-RGBD

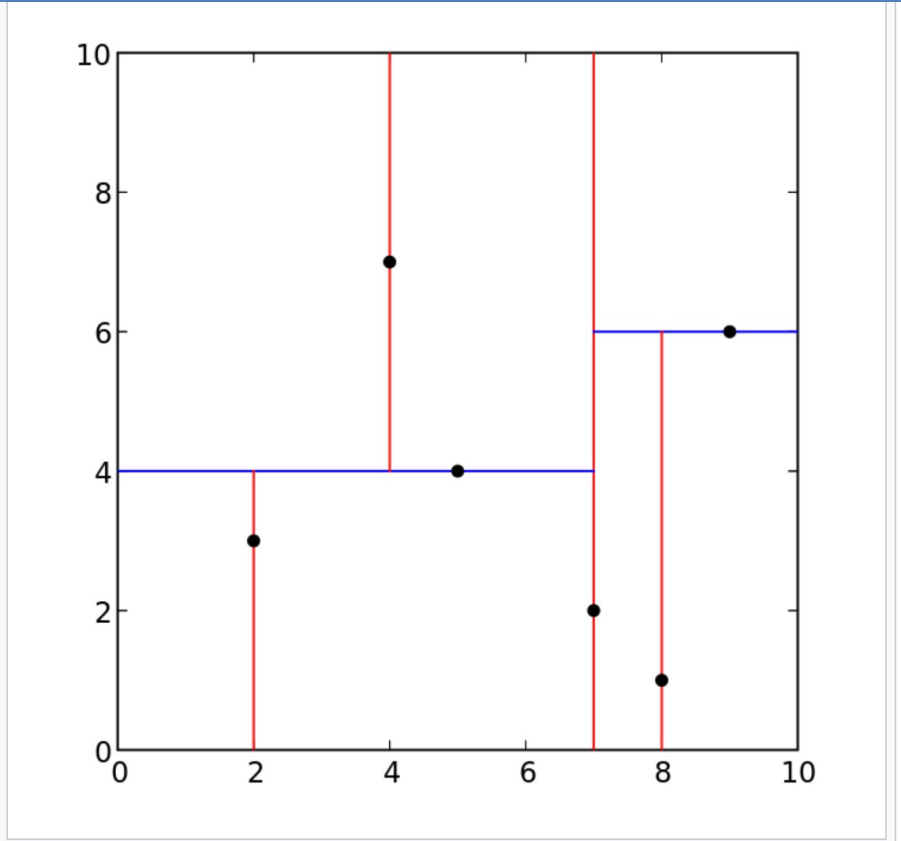


k-d tree

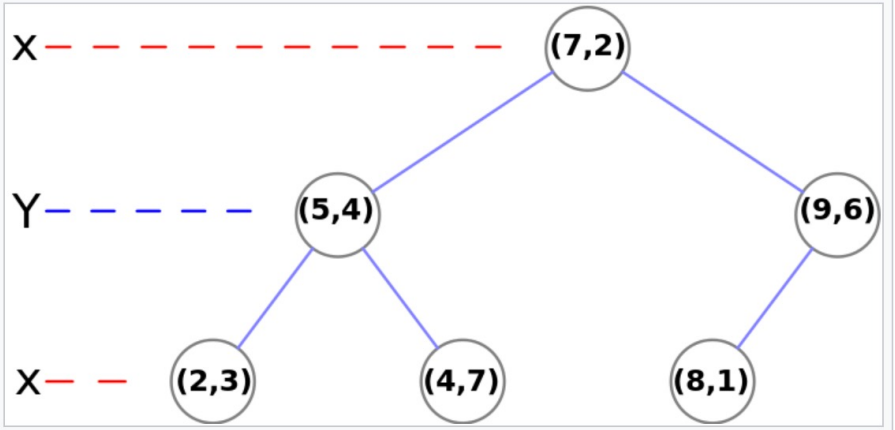
- k-dimensional binary search tree
- Robotics: typically 3D or 2D
- Every level of tree:
  - For a different axis (e.g. x,y,z,x,y,z,x,y,z) (=> split space with planes)
  - Put points in left or right side based on median point (w.r.t. its value of on the current axis) =>
  - Balanced tree
- Fast neighbor search -> ICP!

Algorithm	Average	Worst case
Space	$O(n)$	$O(n)$
Search	$O(\log n)$	$O(n)$
Insert	$O(\log n)$	$O(n)$
Delete	$O(\log n)$	$O(n)$





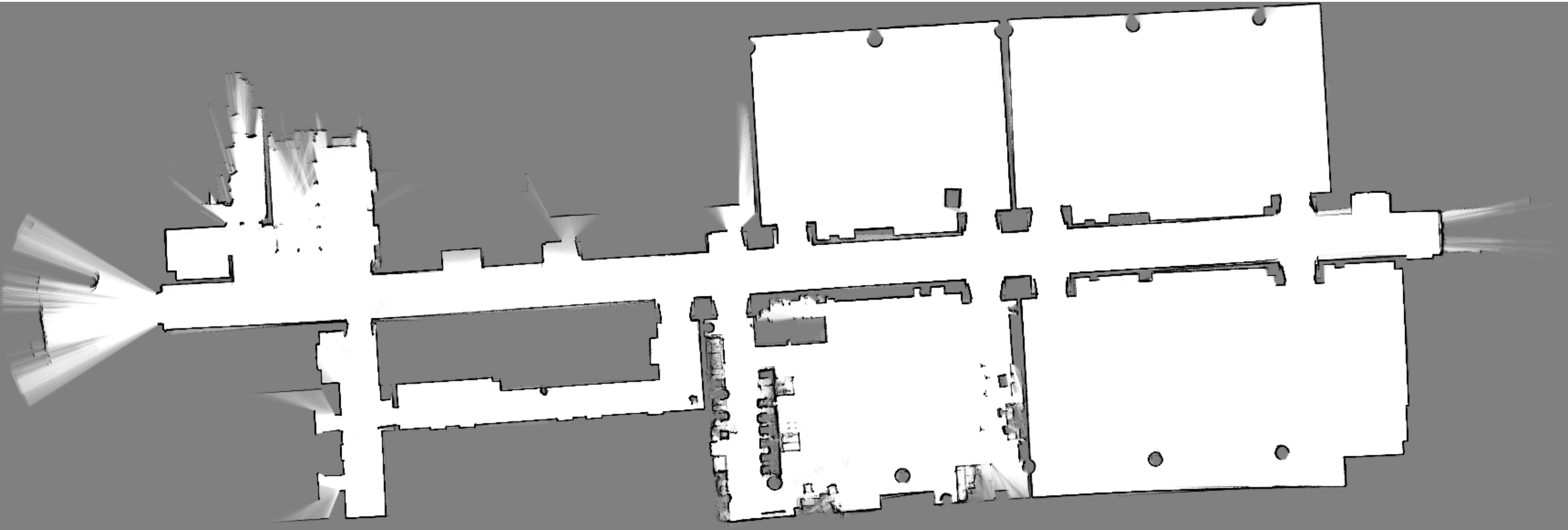
k-d tree decomposition for the point set `(2,3), (5,4), (9,6), (4,7), (8,1), (7,2)`.



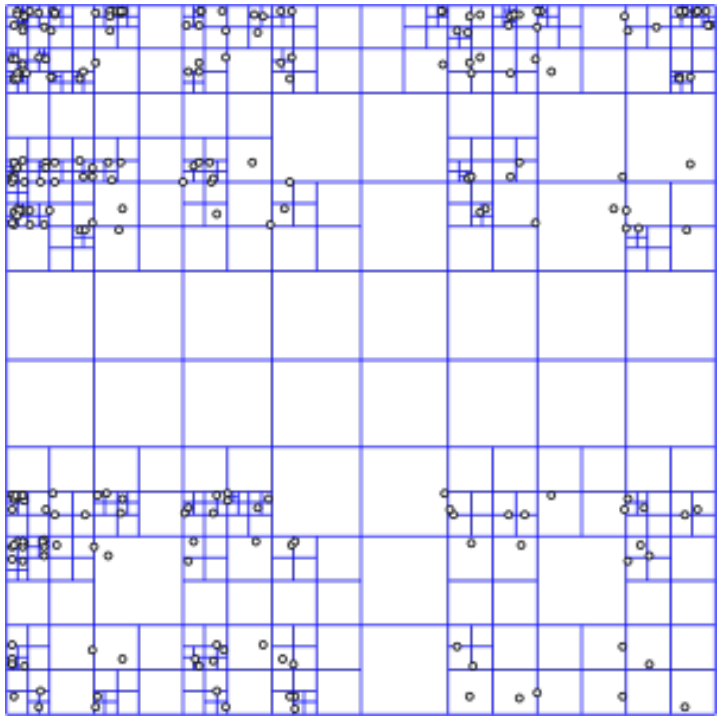
The resulting *k*-d tree.

Probabilistic grid map

- 1: occupied; 0: free; 0.5: unknown
- Need error model of sensor (and of localization) to properly update cells with a scan
- Can remove dynamic (moving) objects (by observing the free space multiple times)



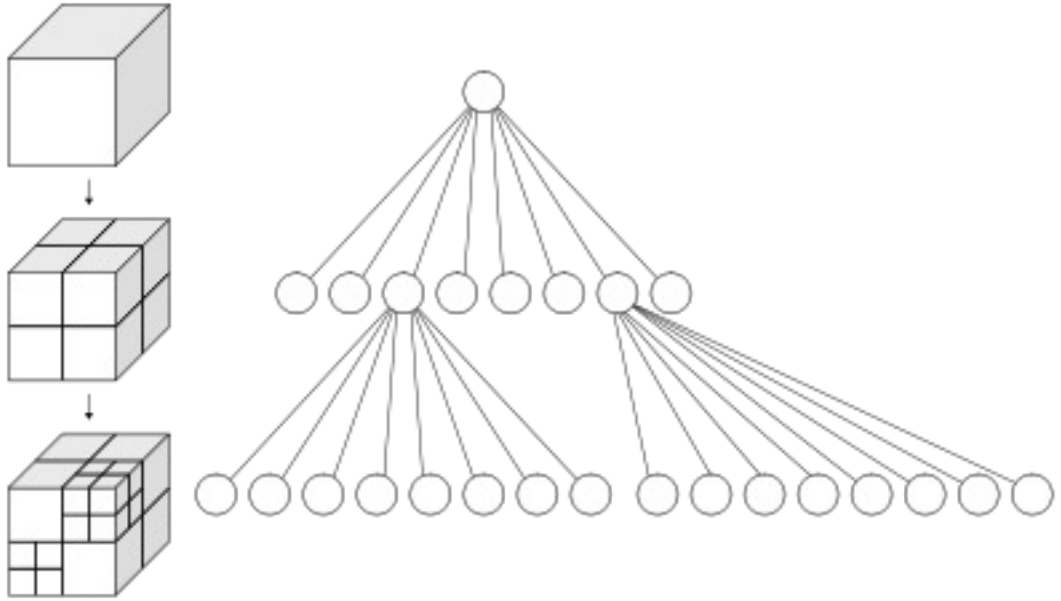
Adaptive Cell Decomposition

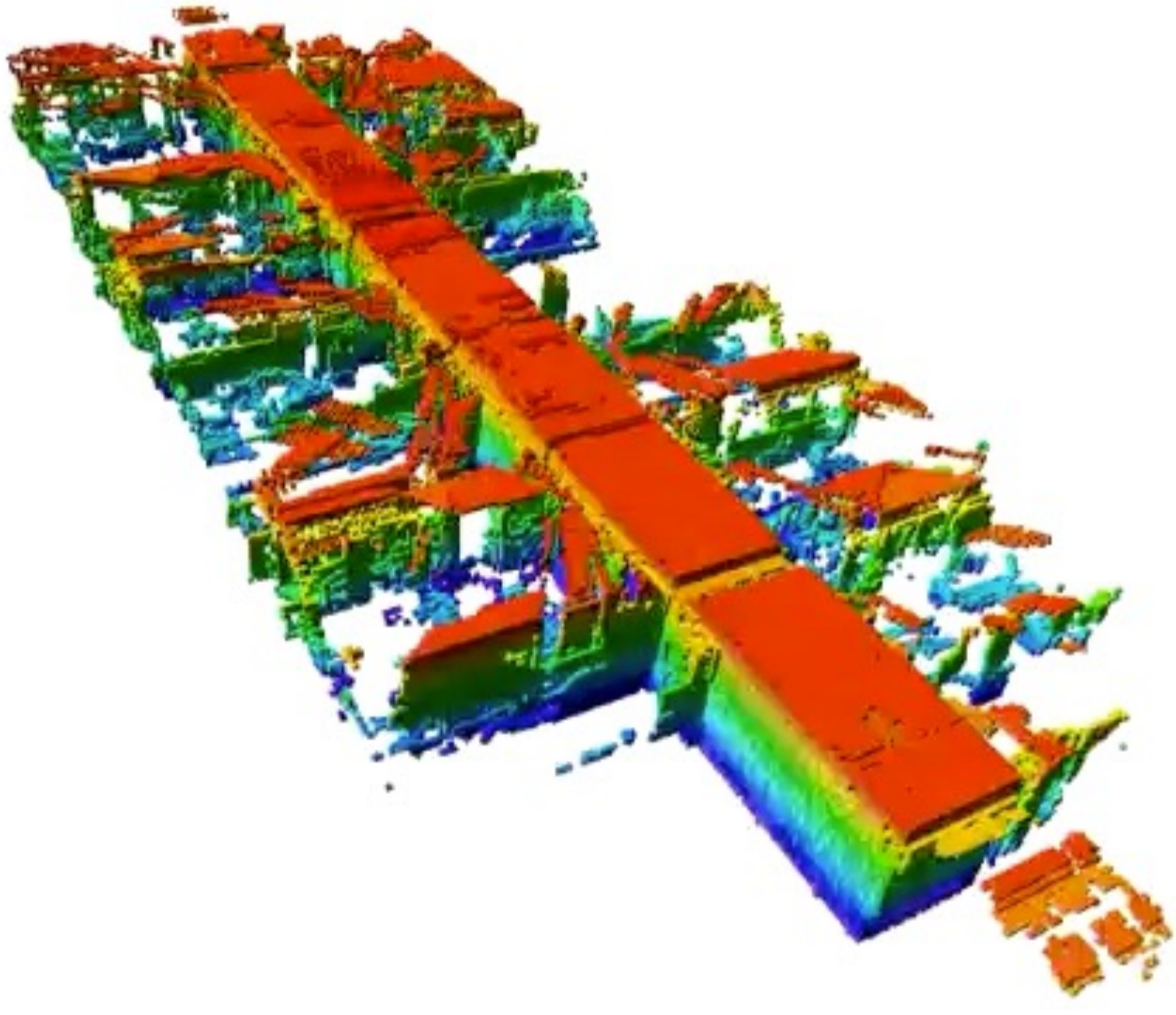


- Quad-tree
  - 2D map/ grid is recursively divided into 4 smaller cells
  - Only cells with different values/ points get divided further =>
  - Compact representation of big space (if many cells stay merged)

- Oct-tree
  - 3D grid divided into 8 smaller cells
  - Very compact! (There is lots of free space!)

- OctoMap: probabilistic oct-tree!
  - http://octomap.github.io
  - Good support in ROS (e.g. MoveIt!)





Semantic Information

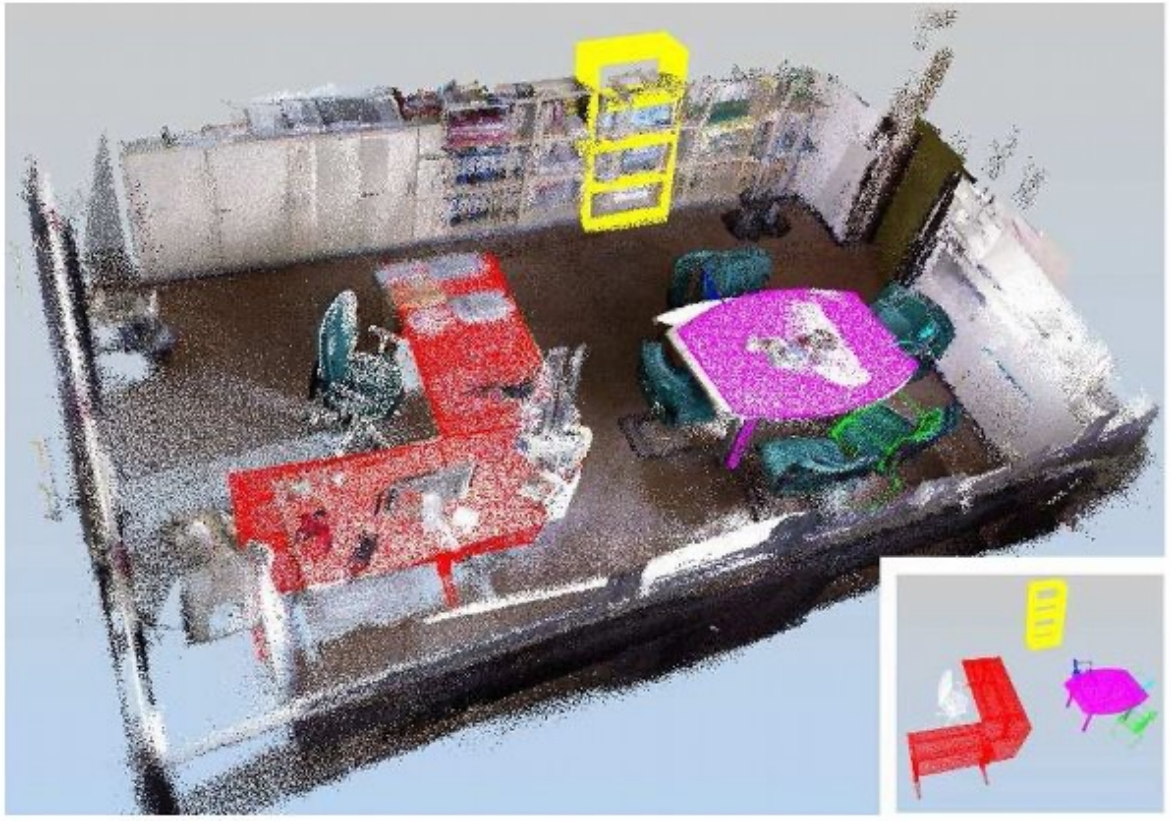
- Assign labels to data
- Segmentation: automatically group data (e.g. points) according to their semantic class

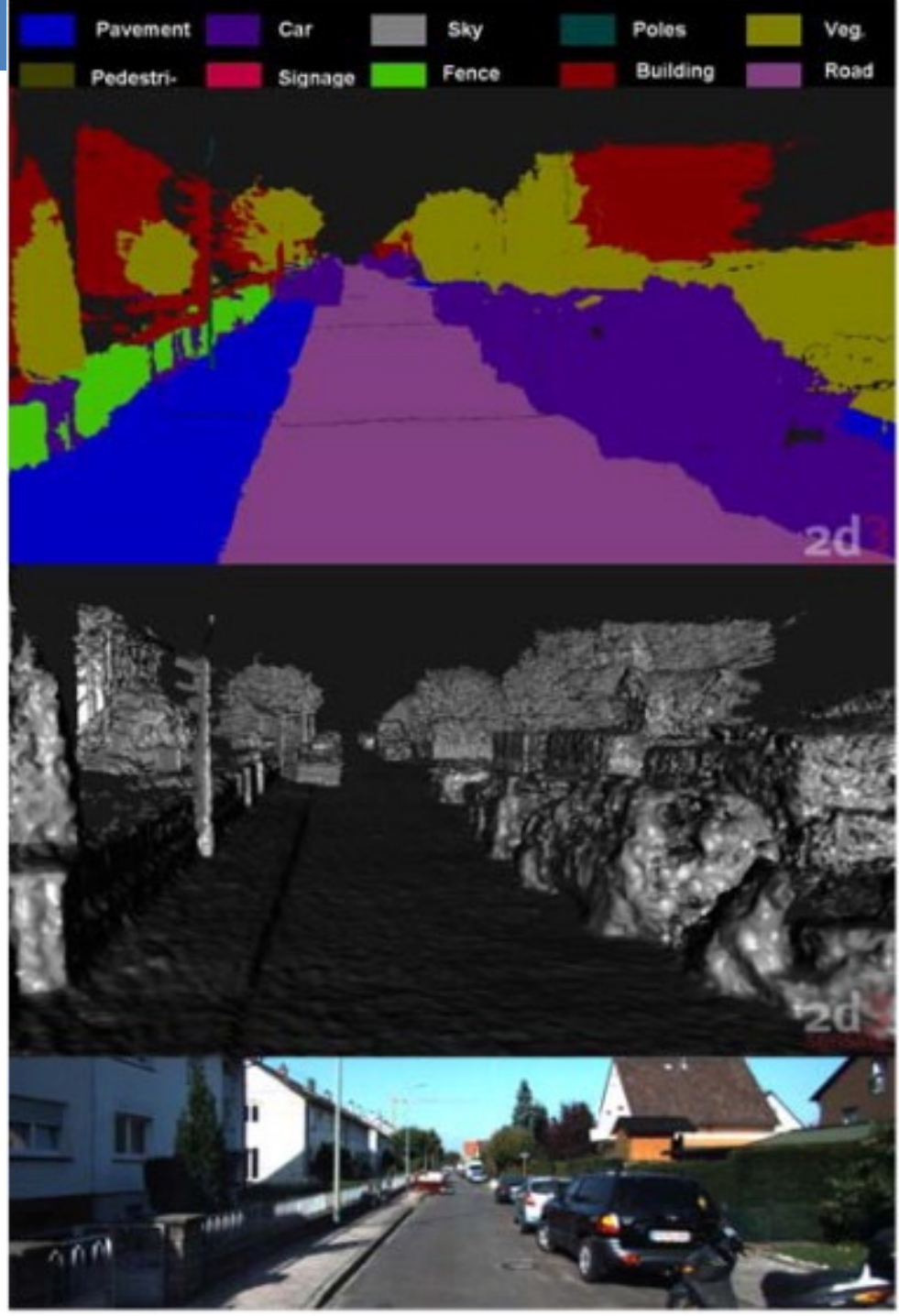
- Even save just very high level data; e.g. room at (x,y); Eiffel tower; …

- Applications:
  - Human Robot Interaction (“go to kitchen”)
  - Scene understanding
  - Navigation (detect road; detect door)
  - Localization
  - …

Semantic Map

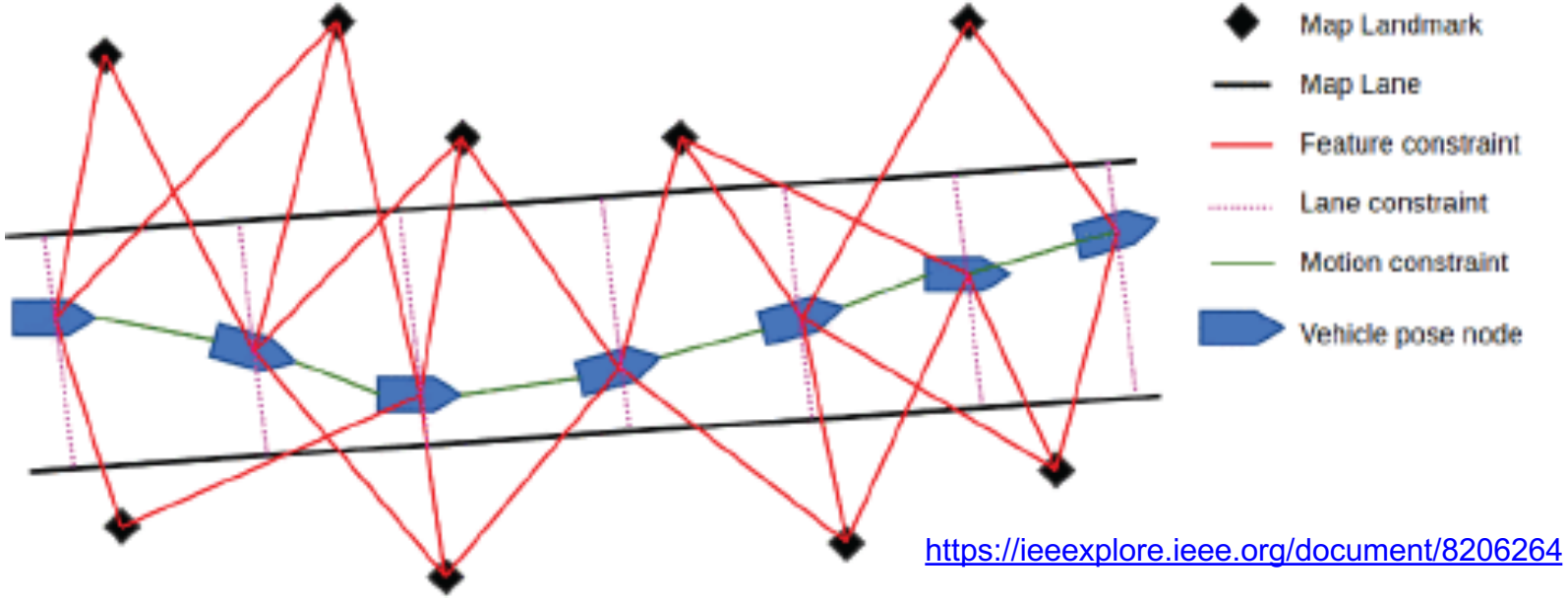
- Semantic Segmentation
  - In room (e.g. detect furniture & objects)
  - Outdoors





Pose Graph

- Graph structure
- Nodes are:
  - Robot
  - Landmarks/ observations



https://ieeexplore.ieee.org/document/8206264

- Used for Simultaneous Localization and Mapping (some more details later today)

- Typically saves (raw) sensor data in robot nodes =>
- For most applications: needs to be rendered before using it:
  put all sensor data in common frame in a point cloud or grid map or plane map or ...

Mapping

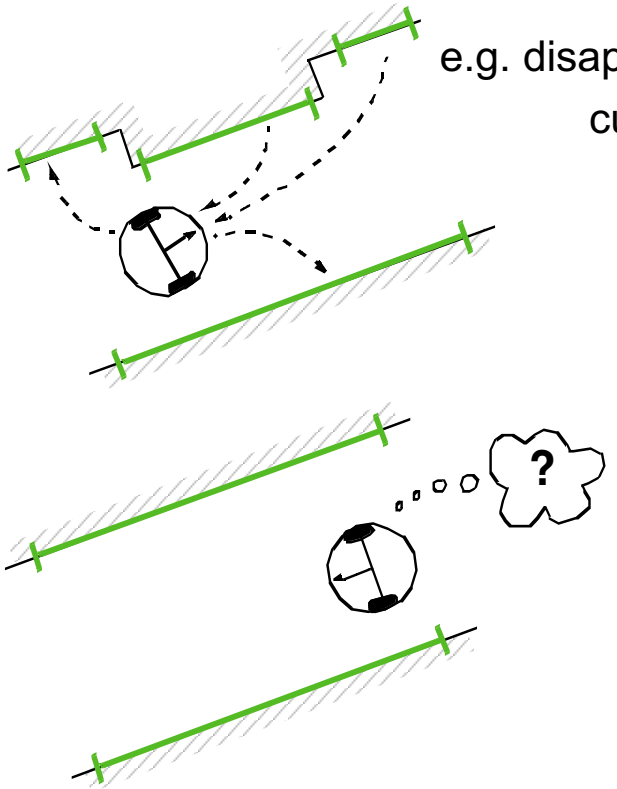
- Process of building a map
- Basic principle:
  1. Initialize the map with unknown or free
  2. Take a sensor scan
  3. Maybe pre-process it (e.g. plane detection)
  4. Localize the robot w.r.t. the map frame (maybe difficult!)
  5. Transform the (processed) sensor scan to the global frame
  6. “Merge” the new data with the old map data, e.g.:
     - Add scanned points to map point cloud
     - Update cells in a probabilistic occupancy grid
  7. Sometimes: Also do ray-casting to mark all cells from sensor to obstacle as free
  8. Repeat for every new sensor scan

- Localization step may need the map (e.g. matching the scan against the map) => both should be done at the same time =>
- Simultaneous Localization and Mapping : SLAM

SIMULTANEOUS LOCALIZATION AND MAPPING - SLAM

Map Building: The Problems

1. Map Maintaining: Keeping track of changes in the environment

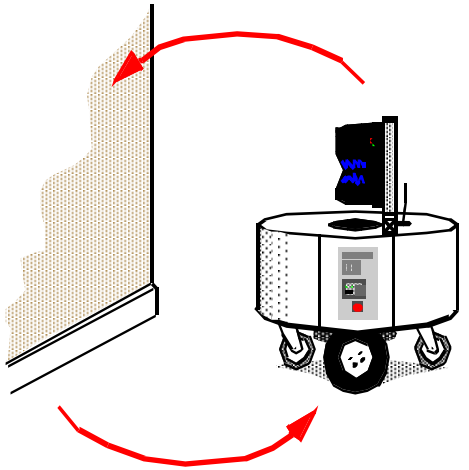


e.g. disappearing cupboard

- e.g. measure of belief of each environment feature

2. Representation and Reduction of Uncertainty

position of robot -> position of wall

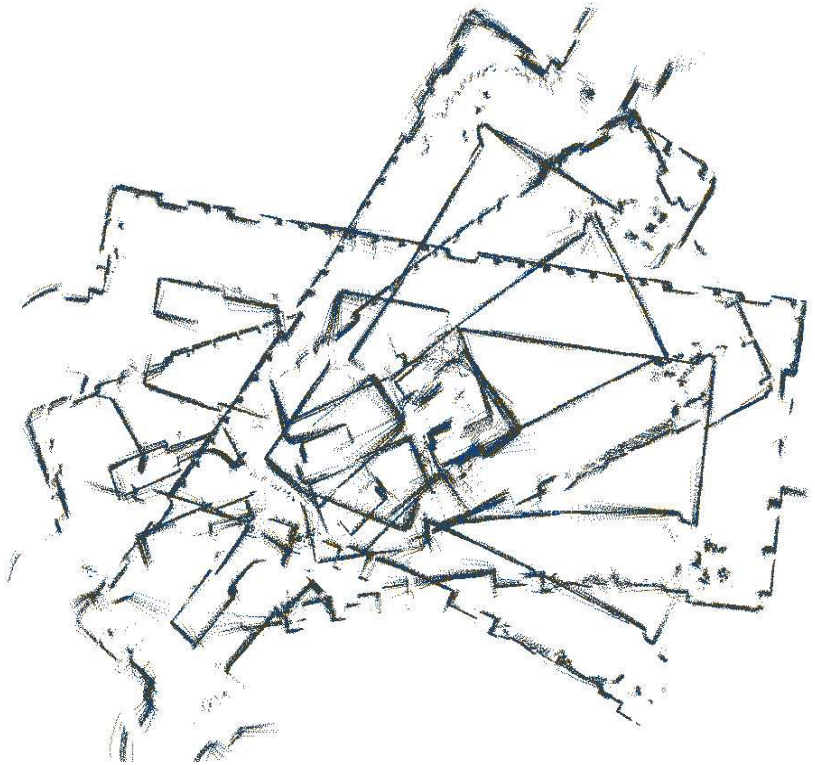


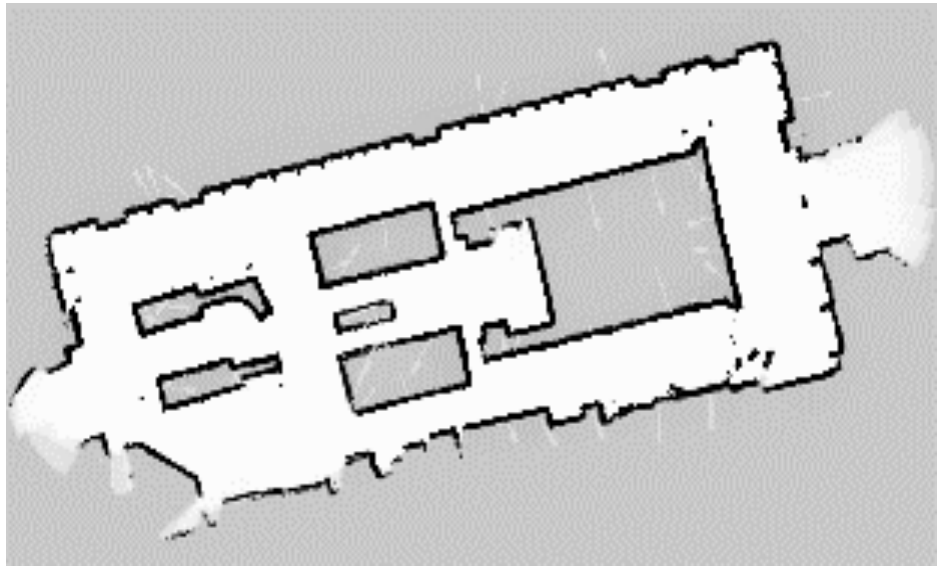
position of wall -> position of robot

- probability densities for feature positions
- Inconsistent map due to motion drift

Cyclic Environments

- Small local error accumulate to arbitrary large global errors!
- This is usually irrelevant for navigation
- However, when closing loops, global error does matter





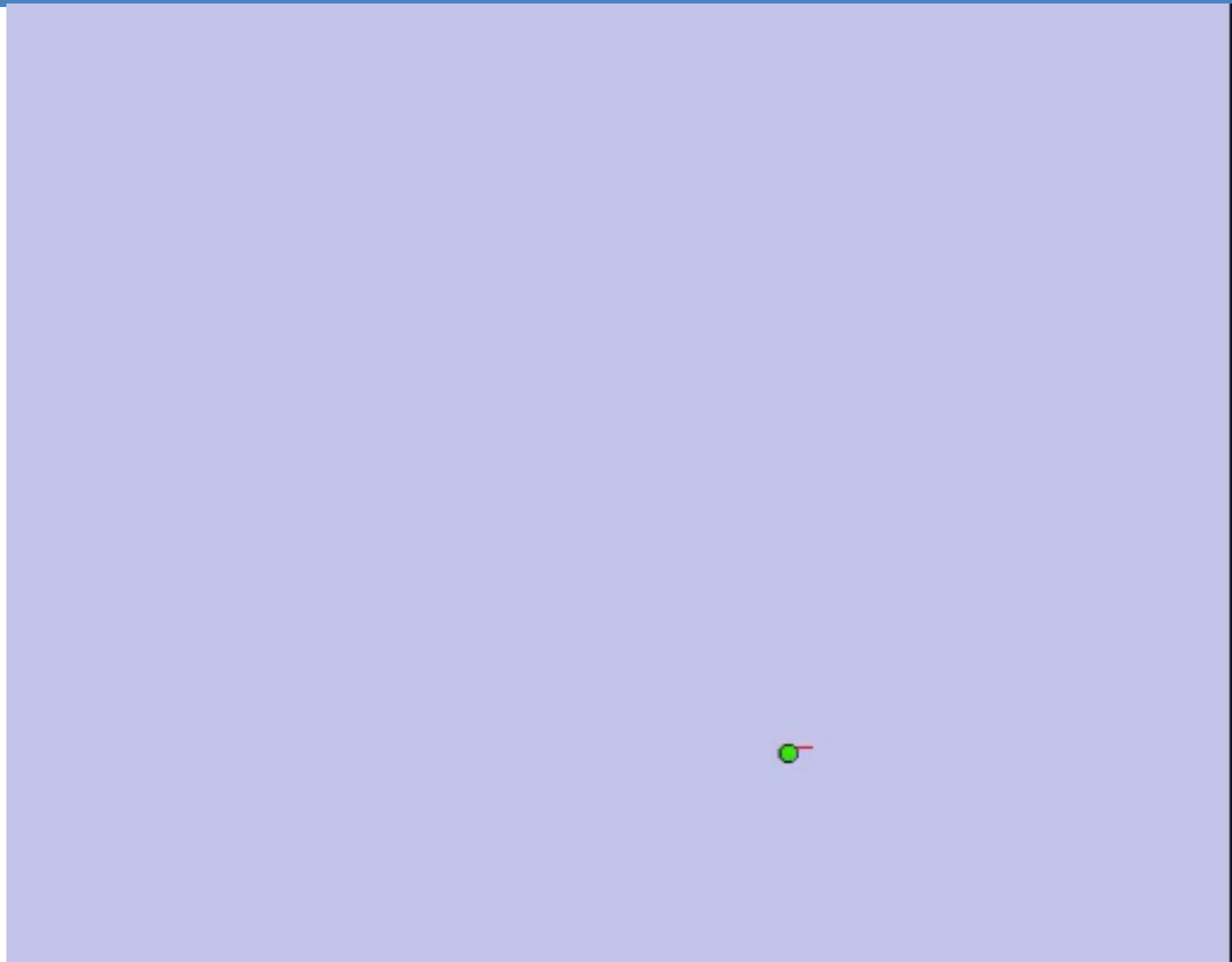
Raw Odometry

- Famous Intel Research Lab dataset (Seattle) by Dirk Hähnel

Courtesy of S. Thrun

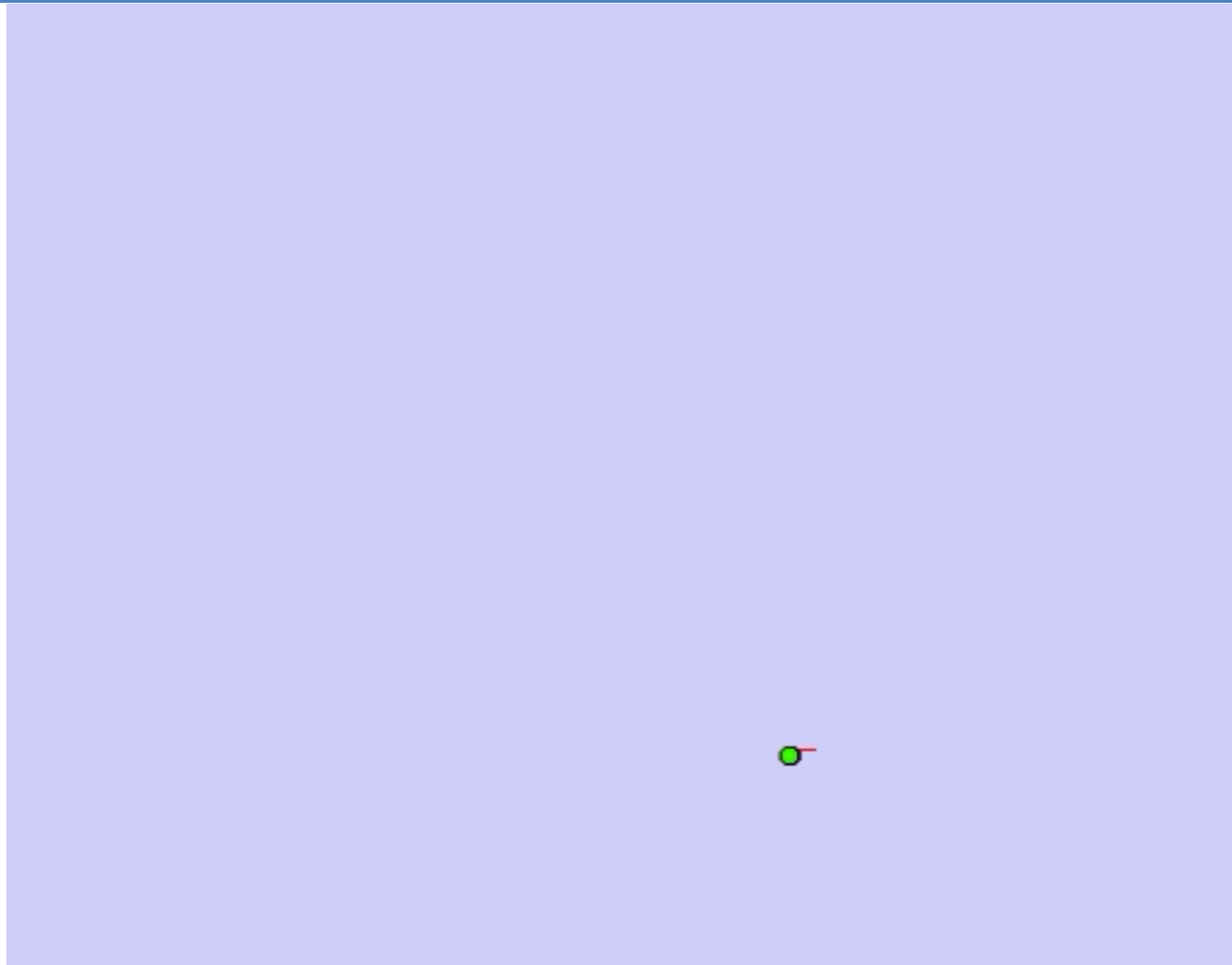
http://robots.stanford.edu/videos.html

Scan Matching: compare to sensor data from previous scan



Courtesy of S. Thrun

FastSLAM: Particle-Filter SLAM

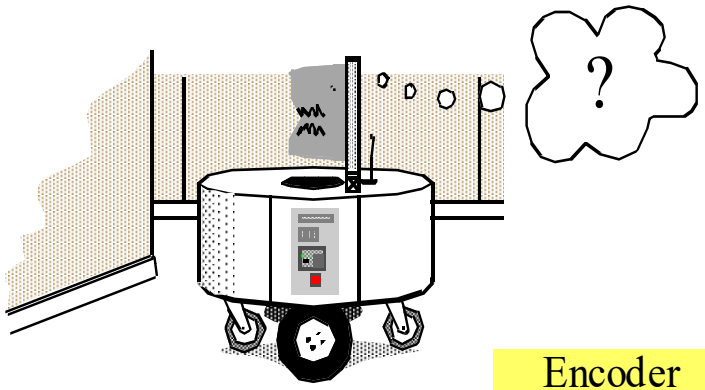


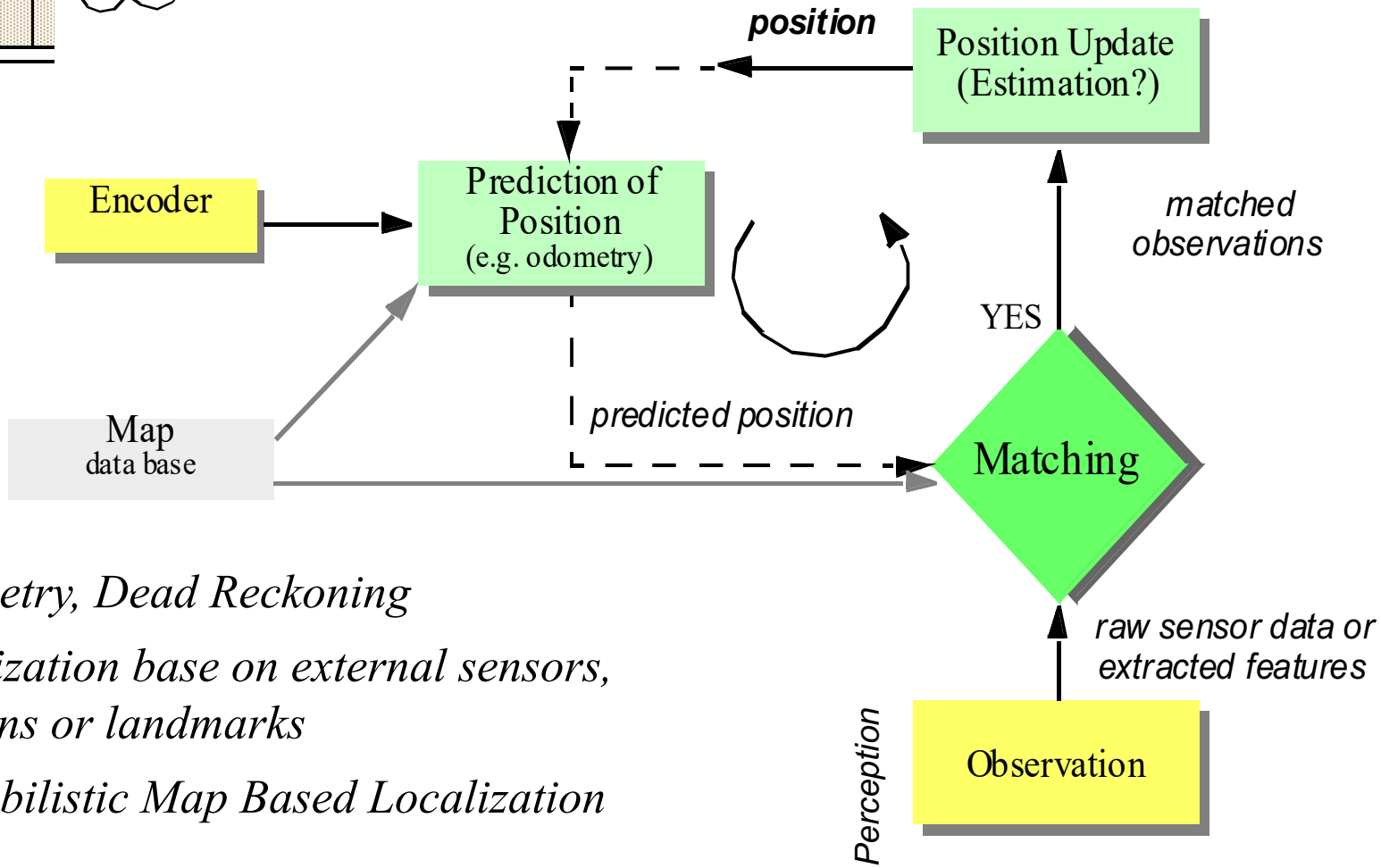
Courtesy of S. Thrun

Scan Matching/ Registration

- Take one sensor scan
- Match against:
  - Another sensor scan
  - Against the map
- Output:
  - The Transform (2D: 3DoF; 3D: 6DoF; each maybe with scale)
  - Uncertainty about the result (e.g. covariance matrix)
- Used for Localization:
  - Typical algorithms for point clouds: ICP!

Map based localization





- *Odometry, Dead Reckoning*
- *Localization base on external sensors, beacons or landmarks*
- *Probabilistic Map Based Localization*

Localization

- Based on control commands => Open Loop!
- Wheel odometry
  - Compass, Accelerometer, Gyro => IMU
- Scan Matching of Range Sensors == Registration (rigid => no scaling or shearing)
  - ICP: scan to scan or scan to map
    - Needs good initial guess
  - NDT registration
  - Feature-based registration
  - Direct/ optimization based registration
- Grid-based Localization
- Kalman Filter Based Localization
- Monte-Carlo Localization (MCL) == Particle Filter
  - Adaptative MCL => AMCL
- Visual Odometry (VO)
  - With IMU: Visual Inertial Odometry (VIO)
- SLAM techniques
- 3D Reconstruction
  - Structure from Motion/ Bundle Adjustment
  - Localization is by-product
- Absolute Localization:
  - GPS
  - Markers (e.g. QR code)
  - Landmarks (e.g. ShanghaiTech Tower)