Robotics II - Planning

Lecture 3/3

M.Sc. Oscar Lima

November, 23th 2019

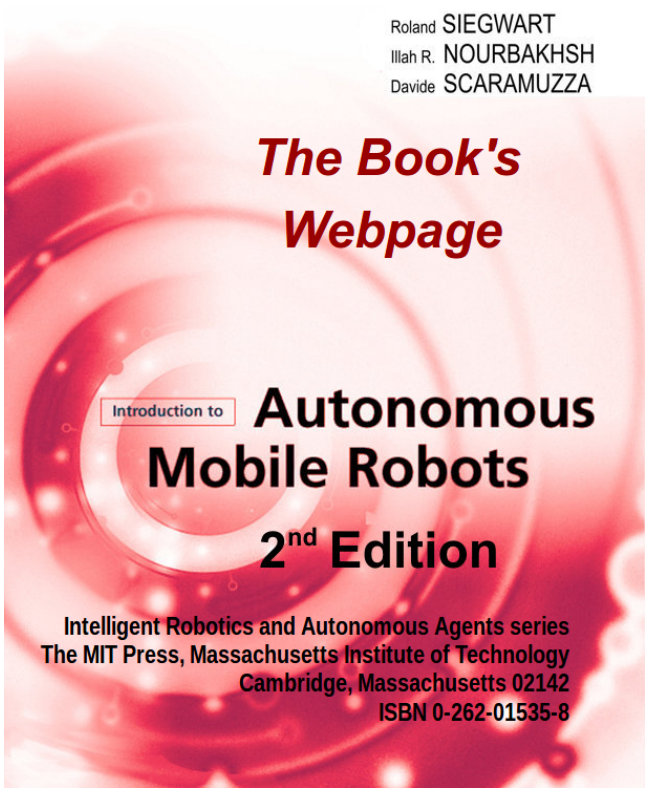
Contents of this lecture

- Introduction
  - Resources
  - Problem description
- Background
  - Path
  - Trajectory
  - Forward Kinematics (FK)
  - DH Parameters and URDF model
  - Inverse Kinematics (IK)
  - Grasp planning problem
  - Octomap
  - Collision checking
  - Configuration space
- Motion Planning Approaches
  - Graph Representation
  - Point robot simplification
  - Visibility graph
  - Voronoi diagram
  - Probabilistic Roadmap (PRM)
  - Cell decomposition
  - Potential Fields
  - Rapidly Exploring Random Trees (RRT)
  - Sampled based motion planning
- Plan Execution
  - ROS Navigation stack
  - OMPL
  - ROS MoveIt
  - Grasping cubes application scenario

Introduction

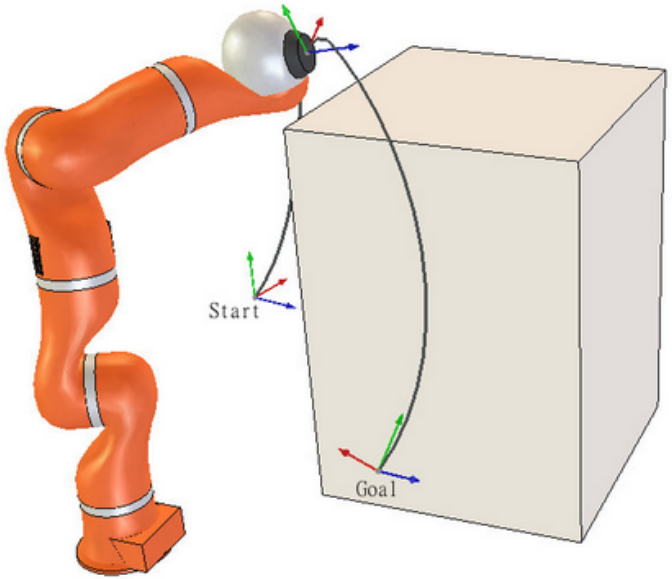
Resources (Literature + 1 link)

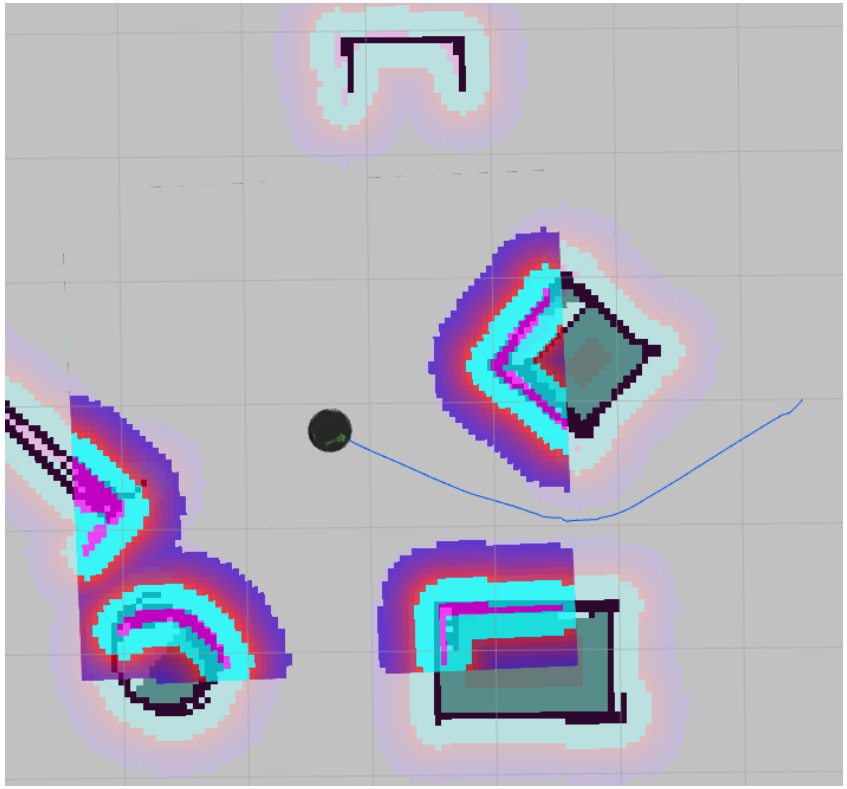
- Siegwart R., Nourbakhsh I., Scaramuzza D.,
  Introduction to Autonomous Mobile Robots
  2011, MIT press
- Latombe J.,
  Robot motion planning
  2012, Springer
- Robotics Penn Engineering online lecture (w/ multiple motion planning lectures)



Motion Planning problem

- Input: Robot, obstacles, initial state, goal state, constraints
- Output: Trajectory or Path

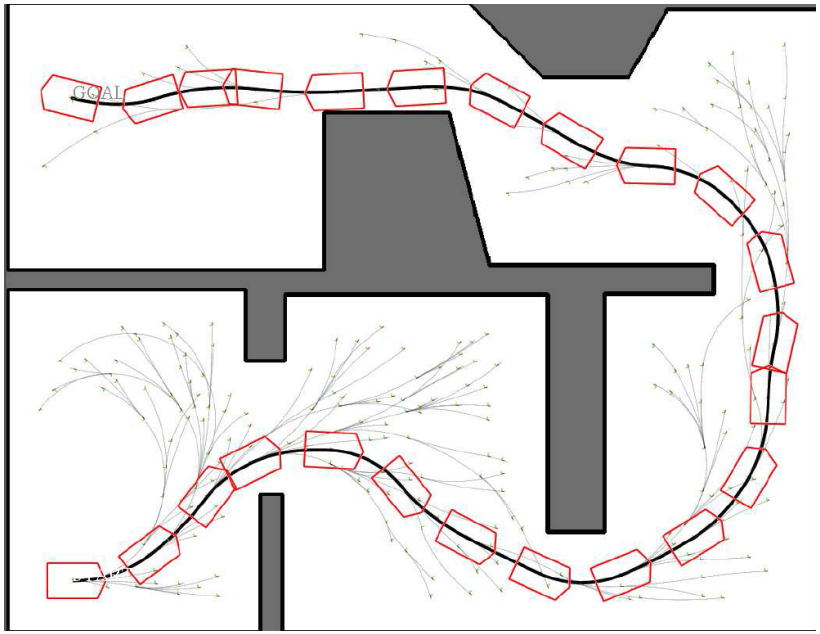


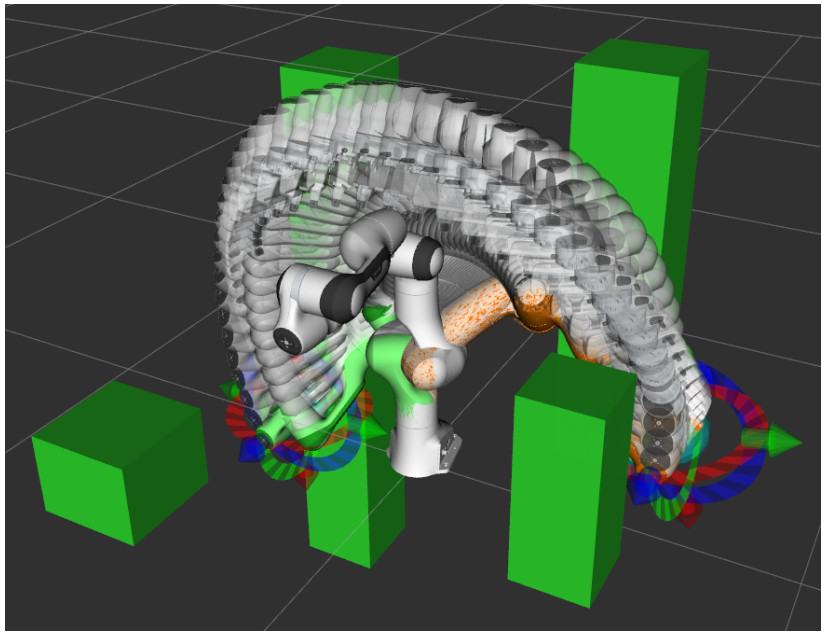


Background

Path:

- Sequence of states (initial state + intermediate states + goal state)
- In the context of 2D Navigation: List of poses
  - (nav) $plan = (P_0, P_1, P_2, ..., P_G)$, $\forall P \in plan, P = (x, y, \theta)$
- In the context of Manipulation: List of arm configurations
  - (manip) $plan = (C_0, C_1, C_2, ..., C_G)$, $\forall C \in plan, C = (\theta_1, \theta_2, ..., \theta_n)$
  - for an n degree of freedom manipulator





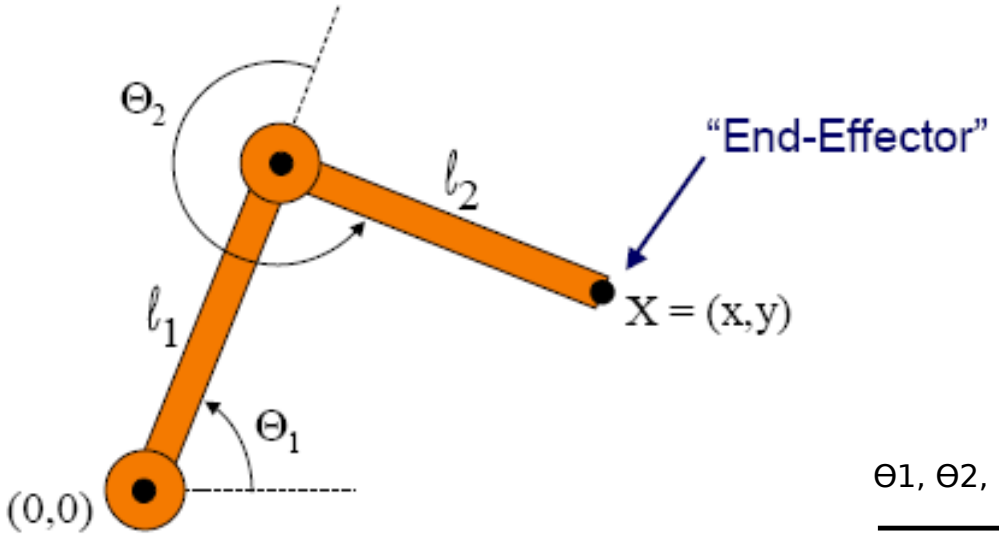
Trajectory extends the path concept

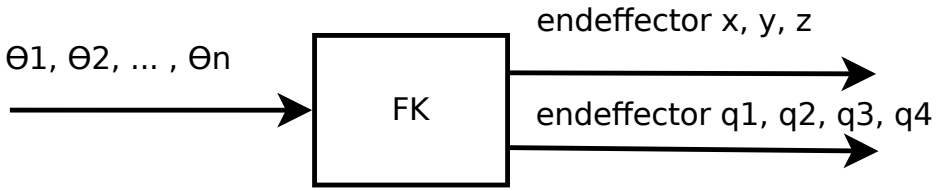
- Path $\subset$ Trajectory
- Trajectory $=$ Path $+$ Velocities $+$ Accelerations

This branches our options in motion planning

- Geometric planning - Path
- Kynodinamic planning - Trajectory (or open loop control)

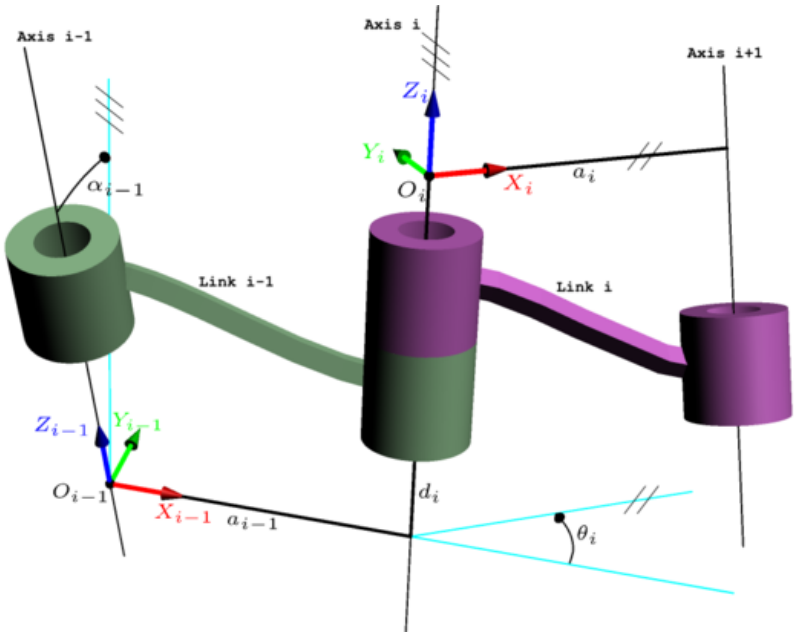
- input: manipulator joint angles
- output: endeffector **pose** in cartesian space[1]
- e.g. from robot manipulator encoder we get the following readings:
- robot state $= (0.3[rad], 1.2[rad]) \rightarrow$ FK solver:
- endeffector pose $= (x, y, z, q_1, q_2, q_3, q_4)$
- where q_x is the endeffector orientation expressed in quaternion





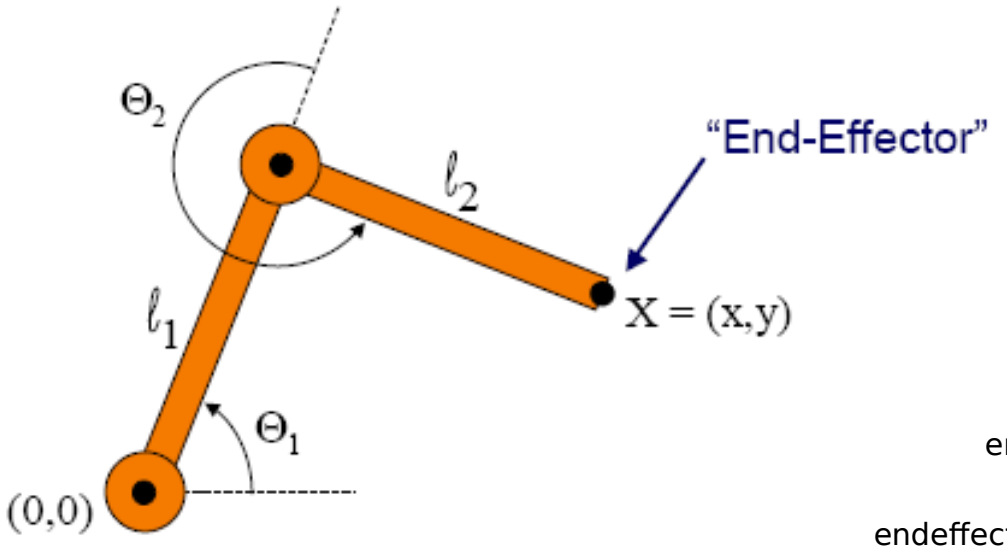
[1] In general gives us the pose of all links in the robot

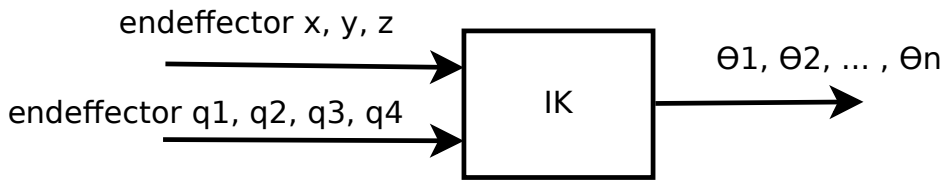
- Standard method to describe kinematic chains in robotics
- DH parameters facilitate the FK computation with lineal algebra
- DH parameters can easily be extracted from URDF model
- URDF: Universal Robot Description Format - Describes robots as XML trees
- URDF ROSCON 2012 introduction video[2]



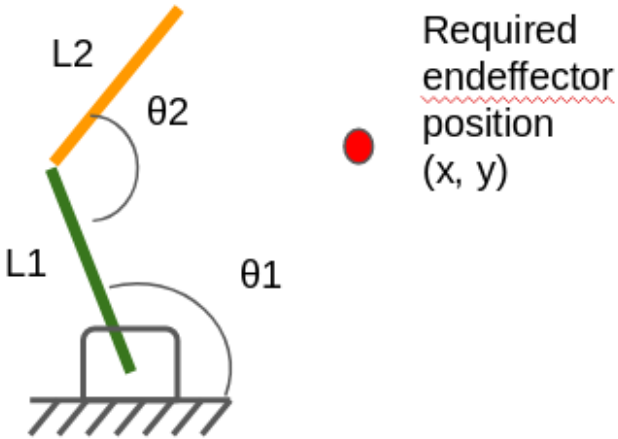
[2] https://youtu.be/g9WHxOpAUnst

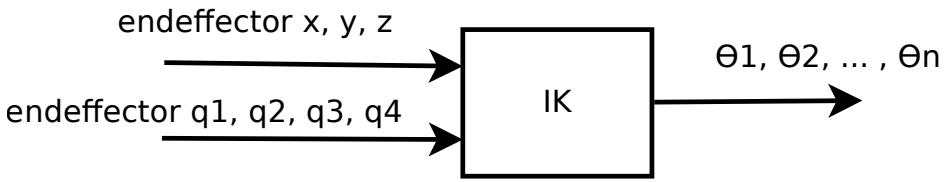
- input: desired endeffector pose in cartesian space
- output: joint angles needed to achieve the required pose
- Solutions can be obtained:
  - Closed form solution (analytically)
  - Numeric methods
  - Neural Networks





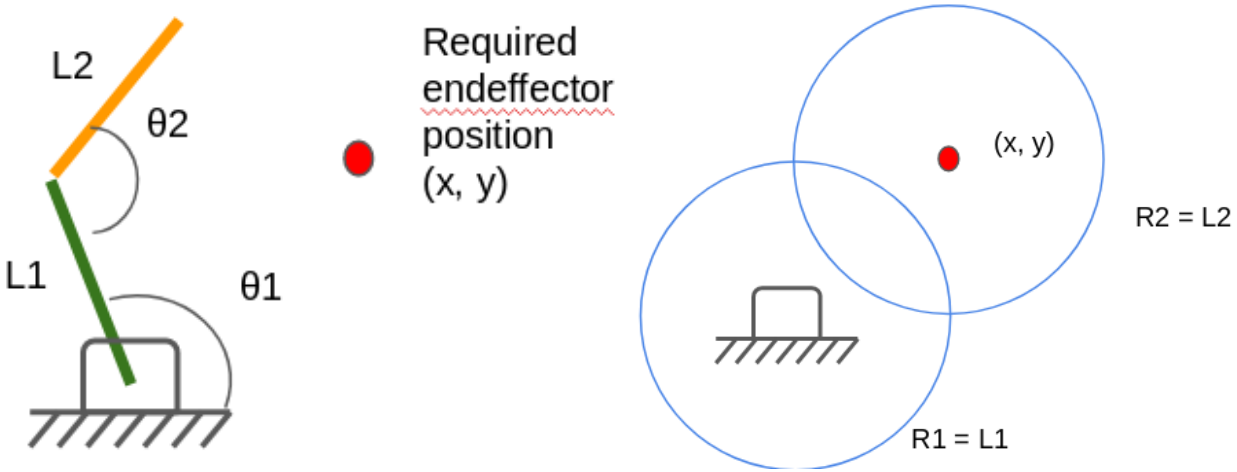
- Example: find close form solution for a 2 dof arm

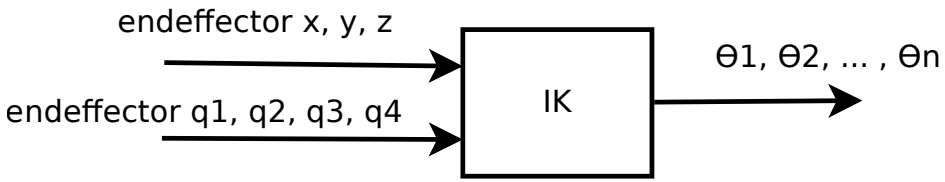




Inverse kinematics (2)

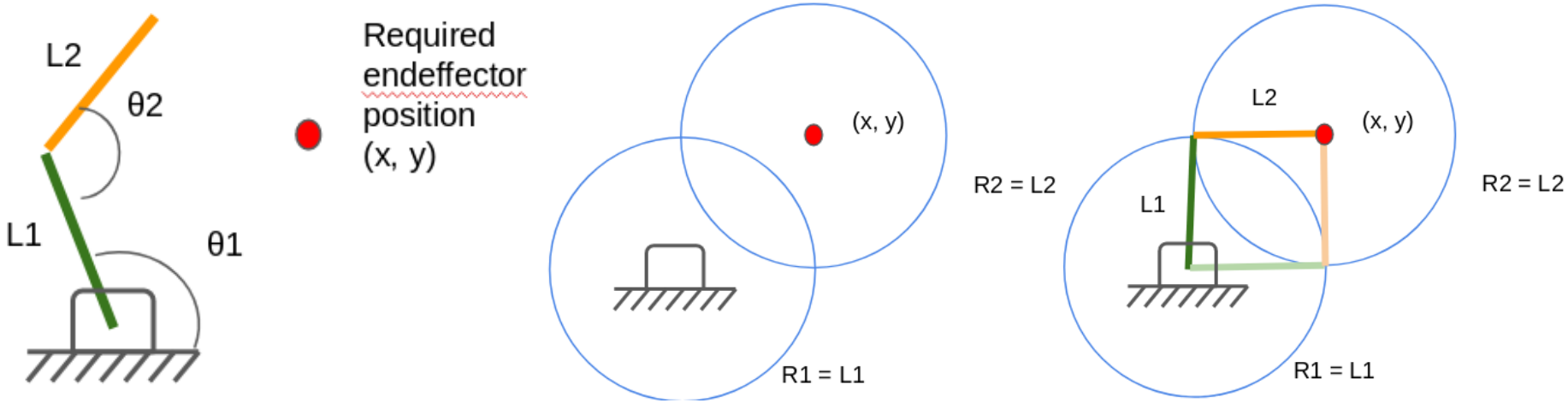
- Example: find close form solution for a 2 dof arm

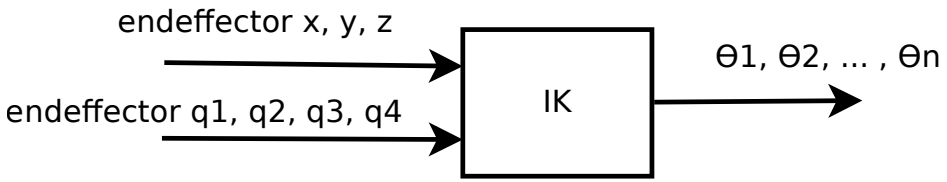




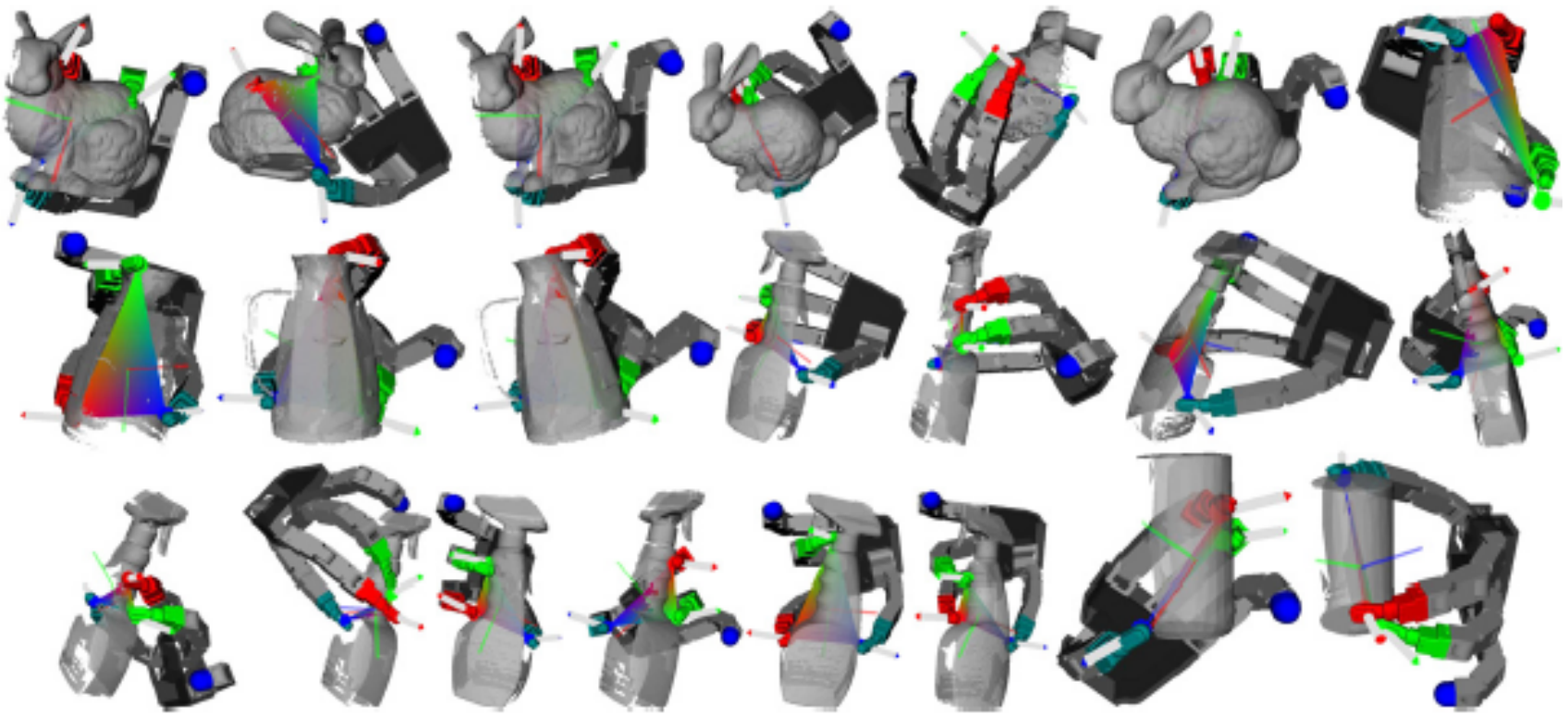
Inverse kinematics (2)

- Example: find close form solution for a 2 dof arm

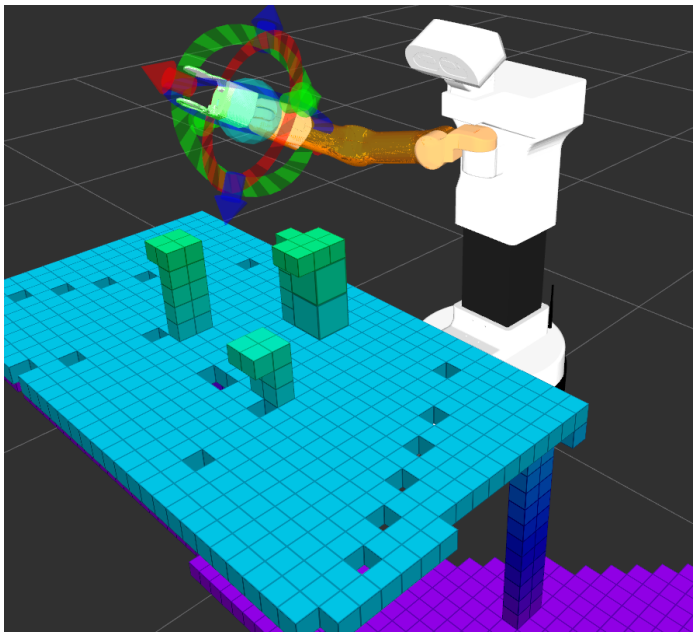




- input: object + gripper (endeffector)
- output: endeffector pose

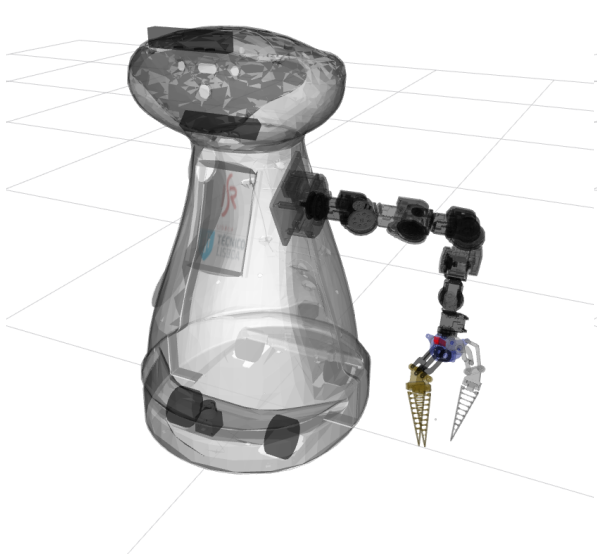


- 3D Occupancy grid map build e.g. from sensor data (pointcloud data)
- Implemented with octree data structure[3]
- MoveIt[4] uses a binary version of it
- Used to store world state (planning scene = obstacle representation)

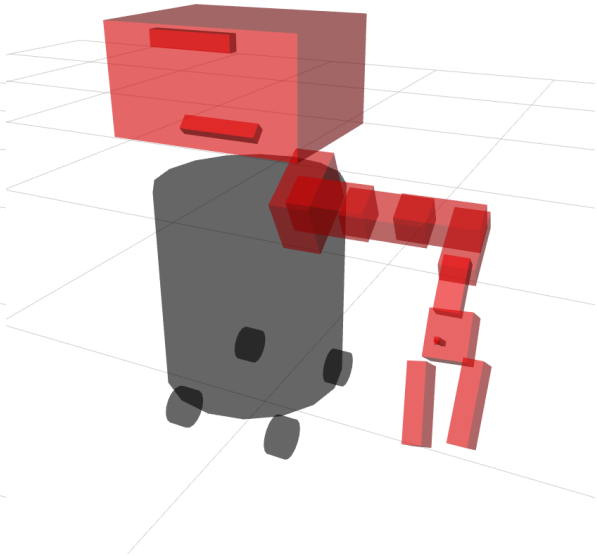


[3] Where each internal node has exactly eight children
[4] Manipulation framework in robotics

- Typically libraries are used for this purpose (e.g. FCL[5])
- Detect whether the two models overlap
- Input: simplified robot model + Obstacles, output: *overlap*, $\neg$*overlap*
- Collision queries are slow unless dedicated hardware is used [Murray et al. 2016][6]



visual robot model



simplified collision model

[5] https://github.com/flexible-collision-library/fcl

[6] https://youtu.be/VvhuVphsq4I

Configuration space

- Navigation configuration space: $q = (x, y, \Theta)$
- Manipulation configuration space: $q = (q1, q2)$
- $q_{free} \subset q_{space}$, where $q_{free} = \forall q \in q_{space}$ which are collision free

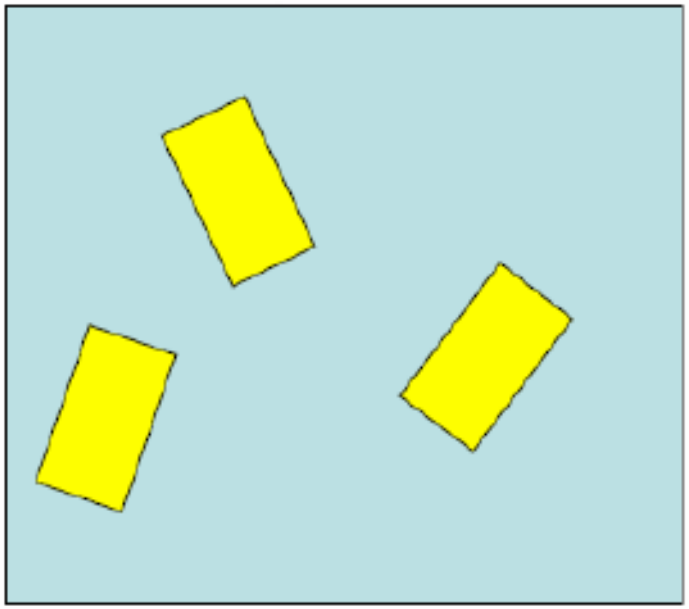


Fig a. Navigation

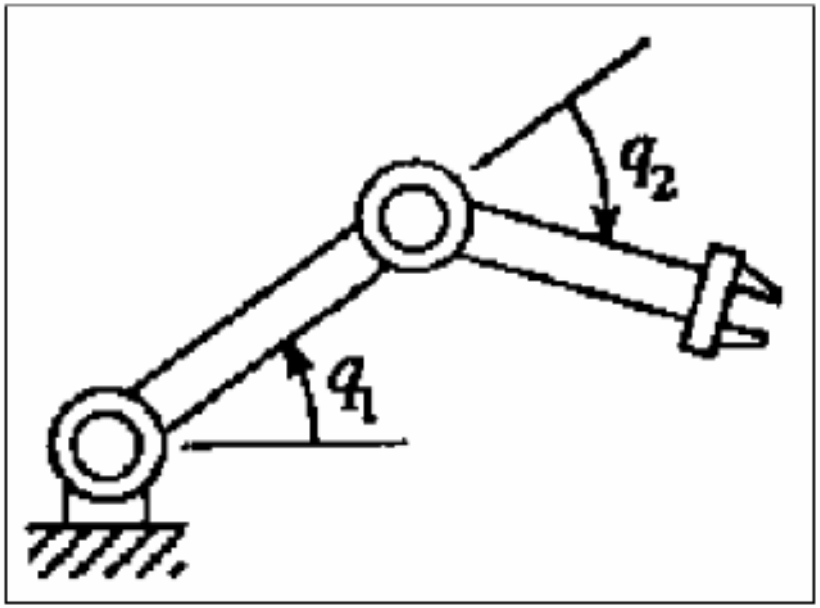


Fig b. Manipulation

Motion Planning Approaches

Basic approaches

- Roadmaps
  - Visibility Graphs
  - Voronoi diagrams
- Cell decomposition
- Potential fields
- Non linear optimization (not covered in this lecture)

Extensions

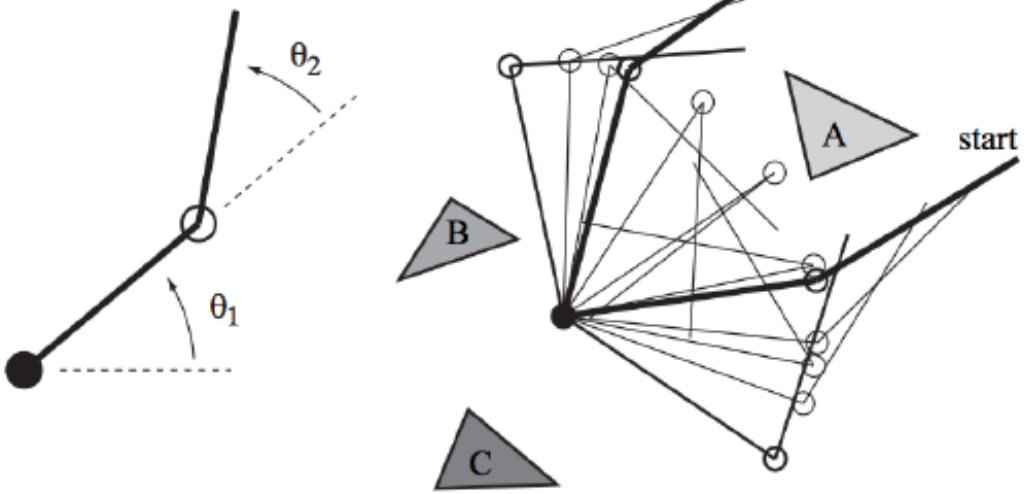
- Sampling techniques

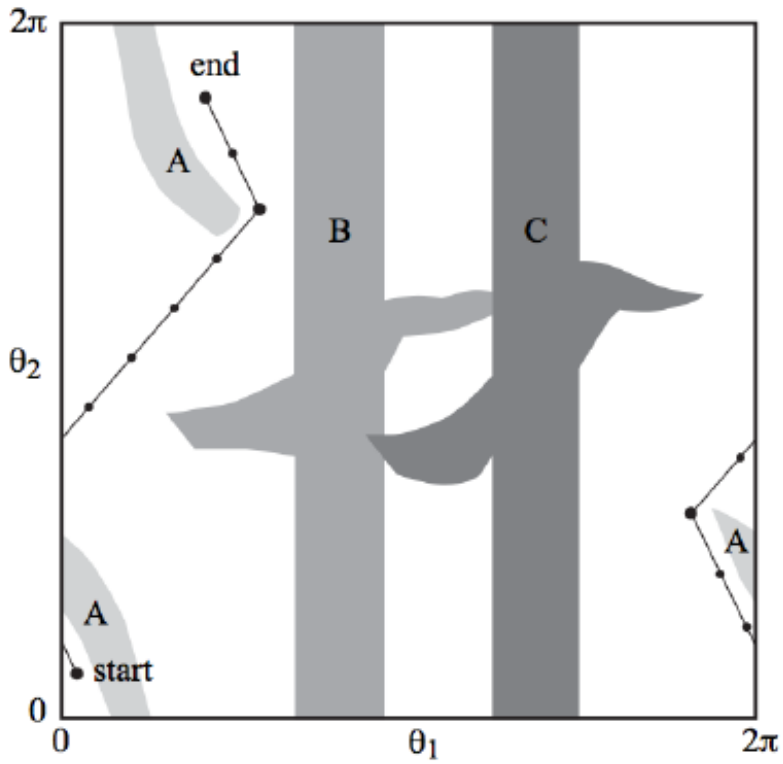
General idea

For most planners:

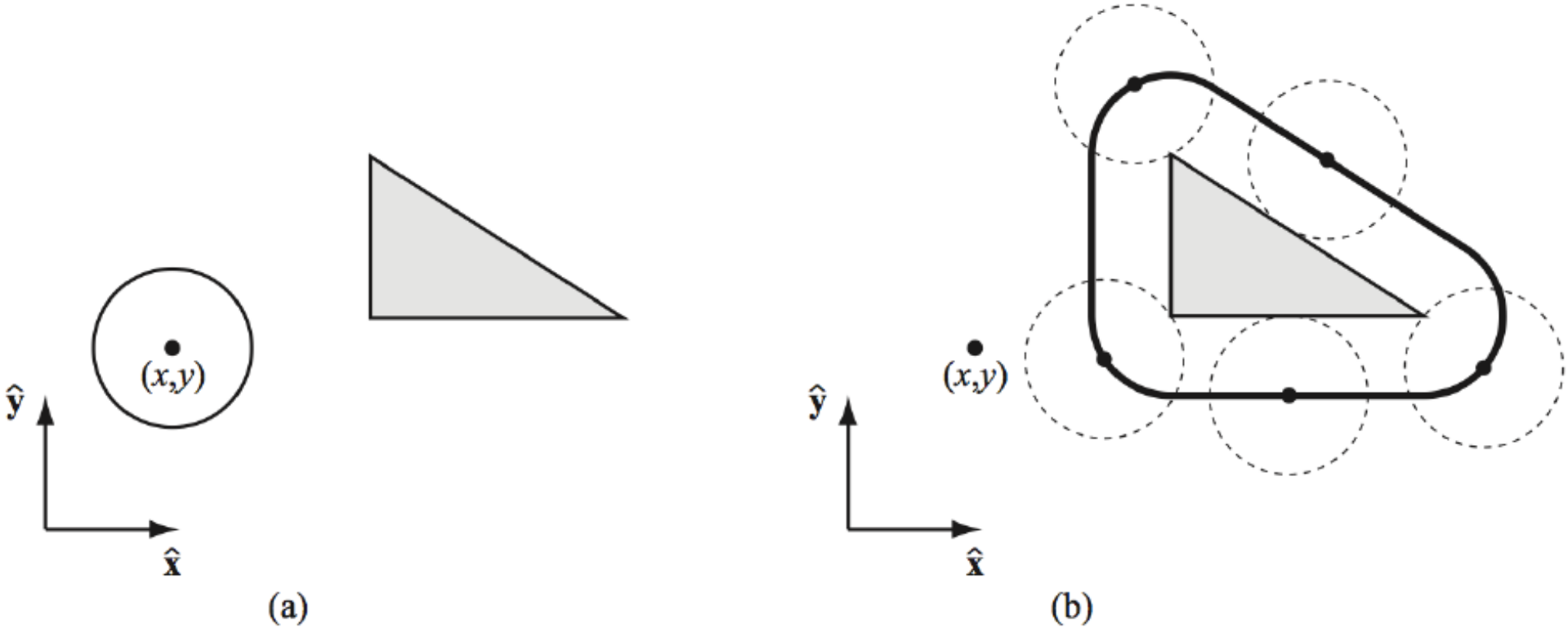
- Input: robot, obstacles, S_0, S_G
- Build a discretized graph representation of C_{free}
- Search the graph for a free collision path

- Nodes: Robot configuration (a point in C-space)
- Edges: free collision steps



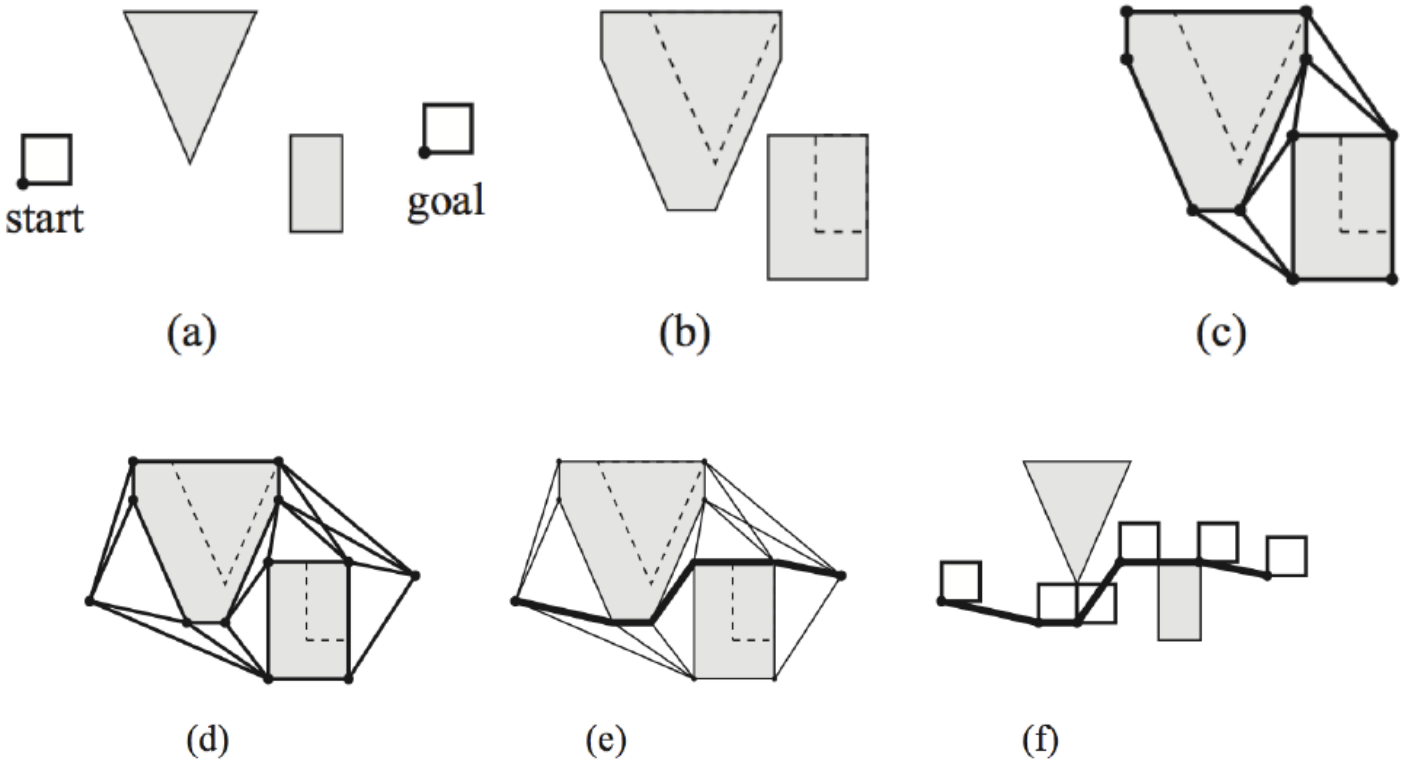


- Consider a 2D navigation scenario
- Robot can be reduced to a point if we grow the obstacles by r
- Any robot pose outside of the boundary is collision free

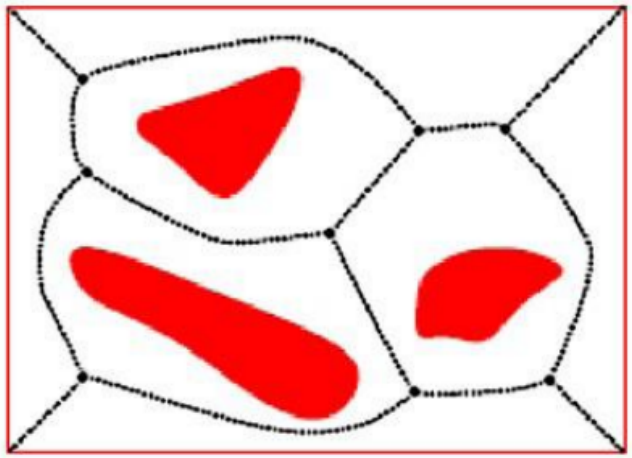


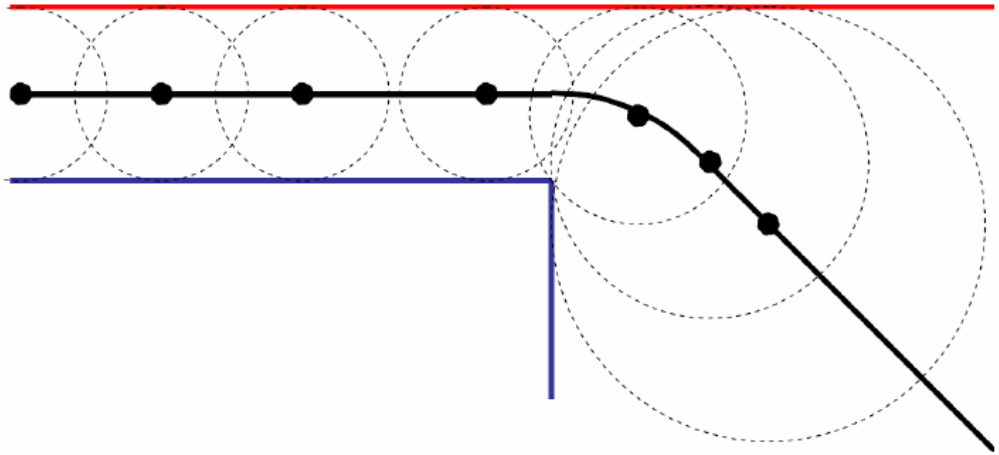
Visibility graph

- Input start and goal position (a)
- Grow obstacles proportionally to the shape of the robot (b)
- Connect all vertices visible to each other (c)
- Search for shortest path (e)



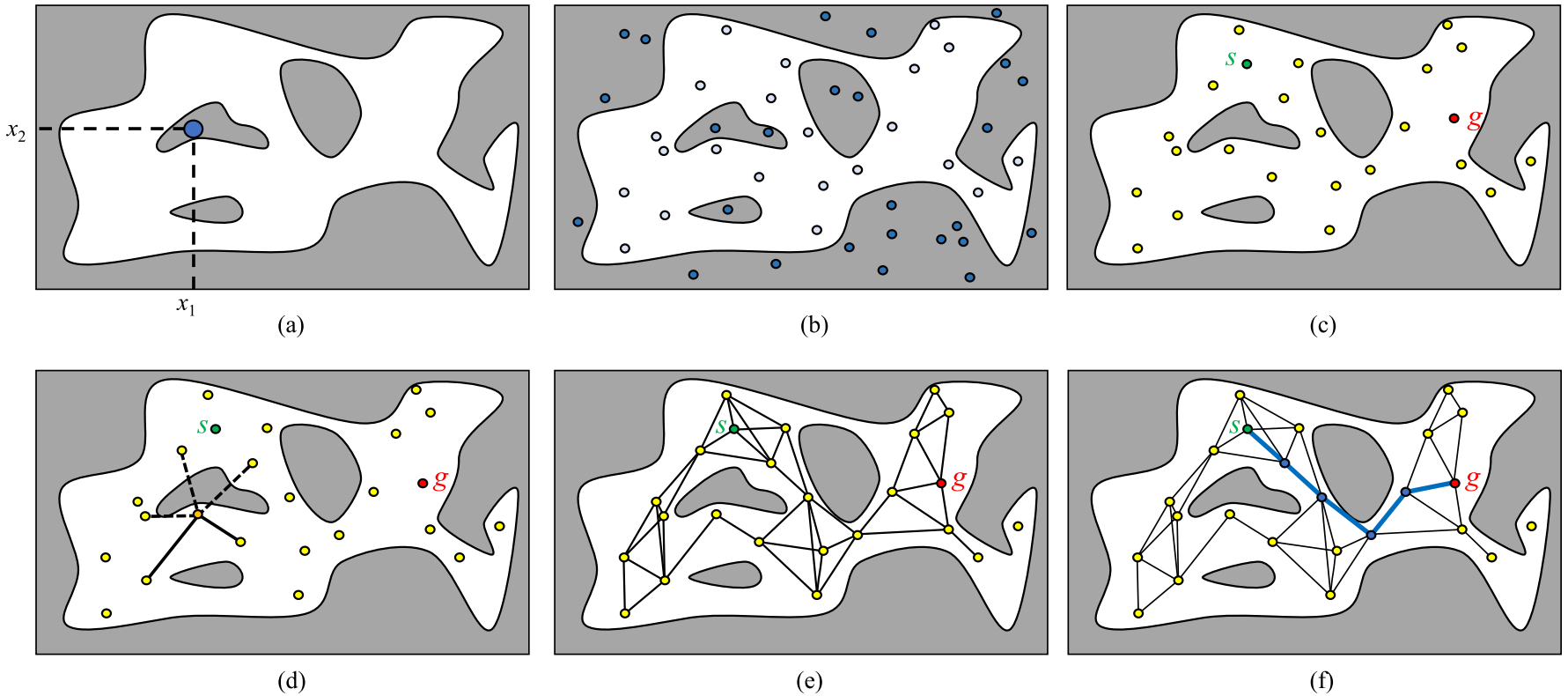
- Set of equidistant points from closest obstacles
- Maximizes clearance between points and obstacles
- Idea: Construct a path between q_{start} and q_{goal} by following edges on the Voronoi diagram



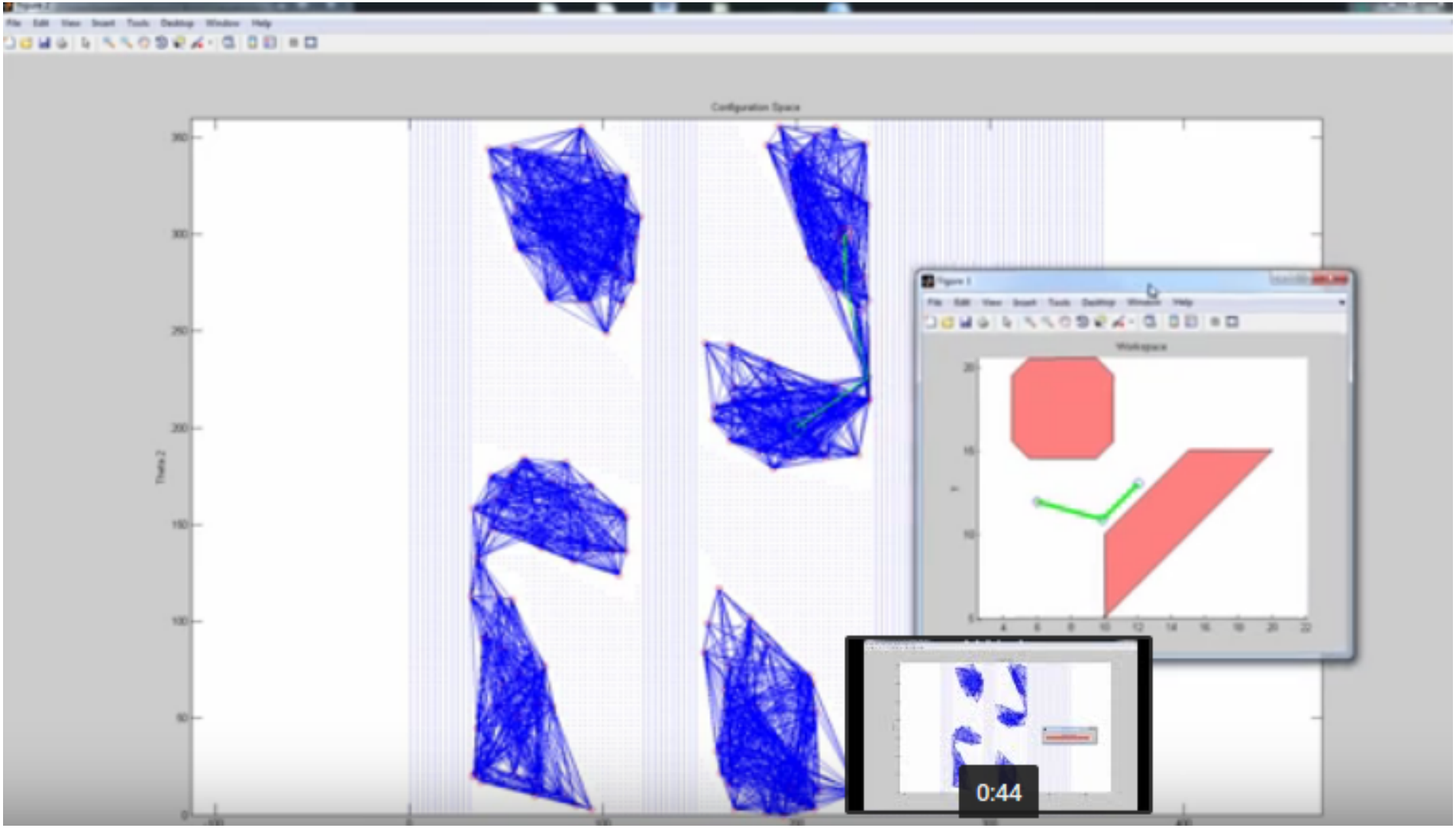


Probabilistic Roadmap

- Avoid searching the entire space
- Staying on the roads is guaranteed to avoid obstacles
- Find path between q_start and q_{goal} by using roadmap



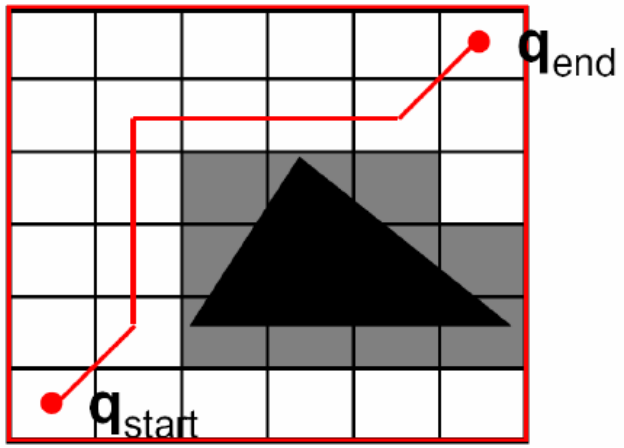
PRM demo video[7]



[7] https://www.youtube.com/watch?v=CIHMdKHedUw

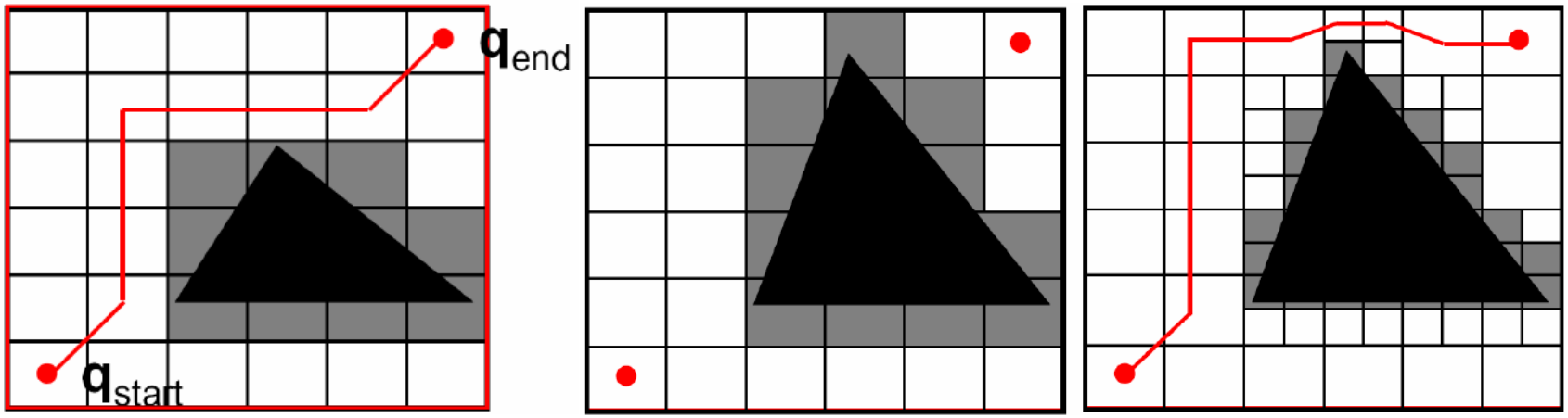
Cell decomposition

- Define a discrete grid in configuration space
- Mark any cell that intersects the obstacle as occupied
- Find path through cells via search (BFS, A*, Dijkstra)
- Algorithm completeness depends on grid resolution

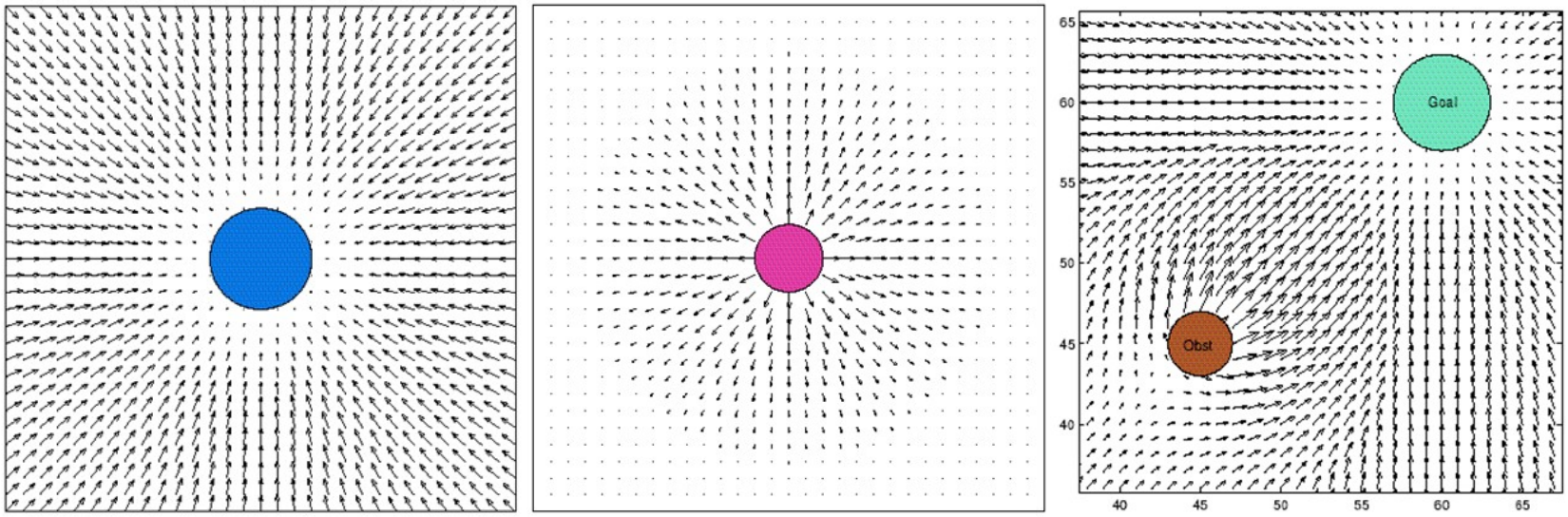


Cell decomposition

- Define a discrete grid in configuration space
- Mark any cell that intersects the obstacle as occupied
- Find path through cells via search (BFS, A*, Dijkstra)
- Algorithm completeness depends on grid resolution

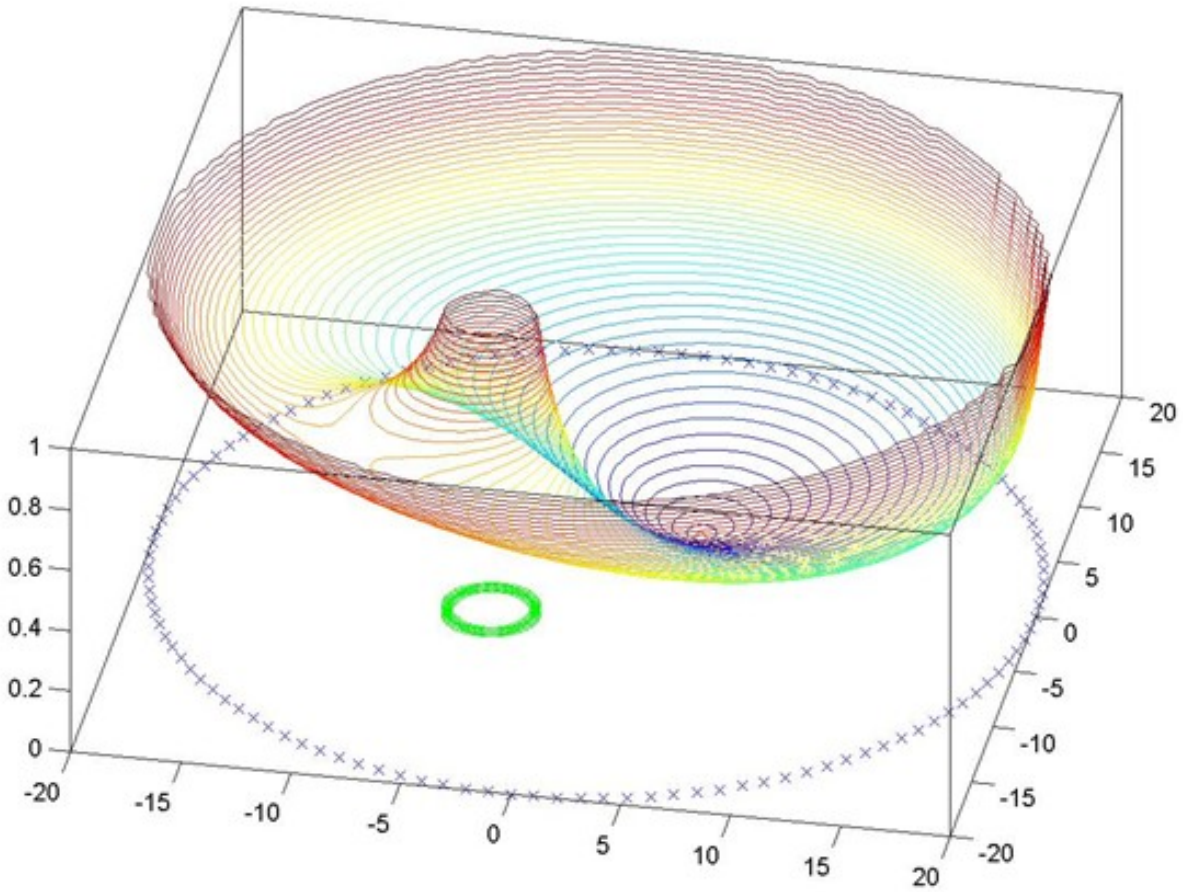


- Goal location generates attraction force, pulling the robot towards the goal
- Obstacles generate a repulsive potential pushing the robot far away from obstacles
- Negative gradient of the resultant potential is treated as an artificial force applied to the robot
  - $U(q) = U_{goal}(q) + \Sigma U_{obstacles}(q)$
  - $F(q) = -\Delta U(q)$



Potential fields (2)

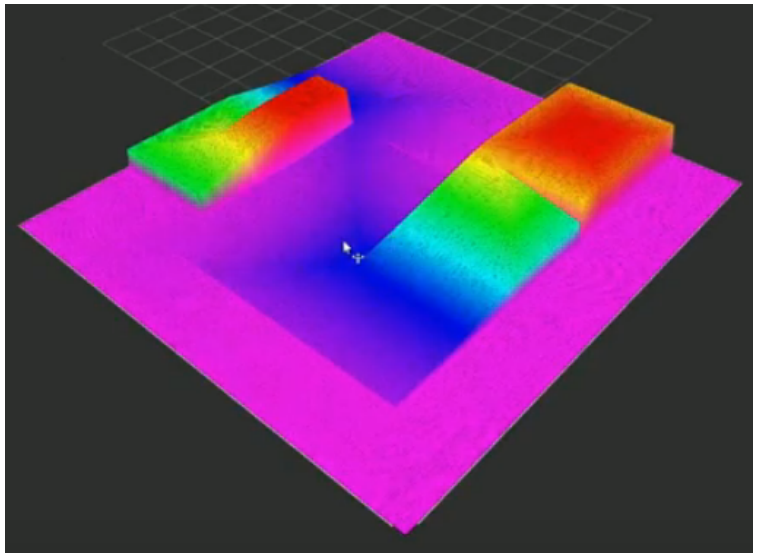
- Gradient visualisation[8]

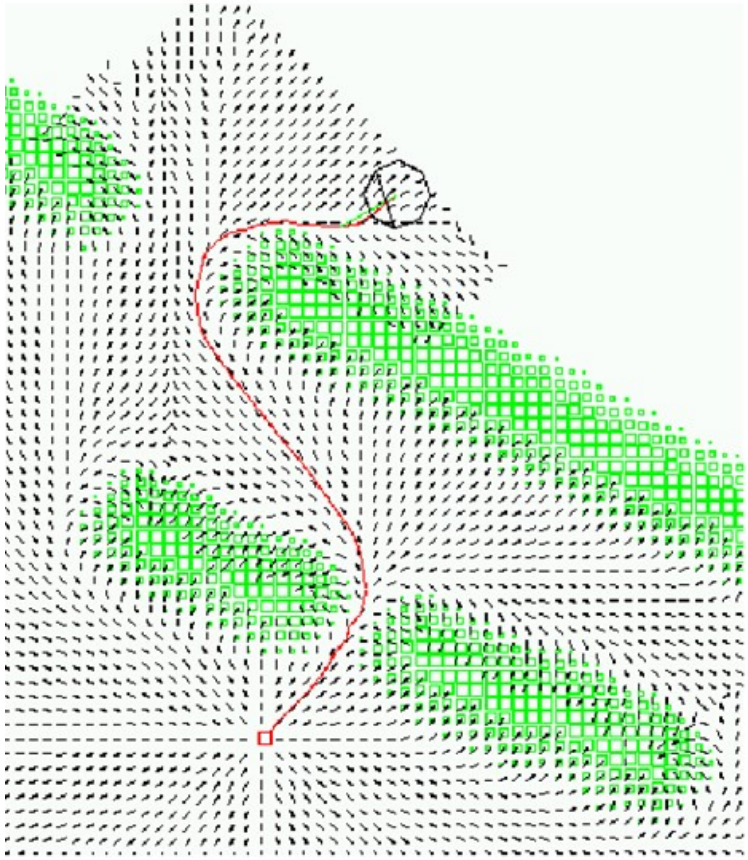


[8] image taken from: https://slideplayer.com/slide/1716961/

Potential fields (3)

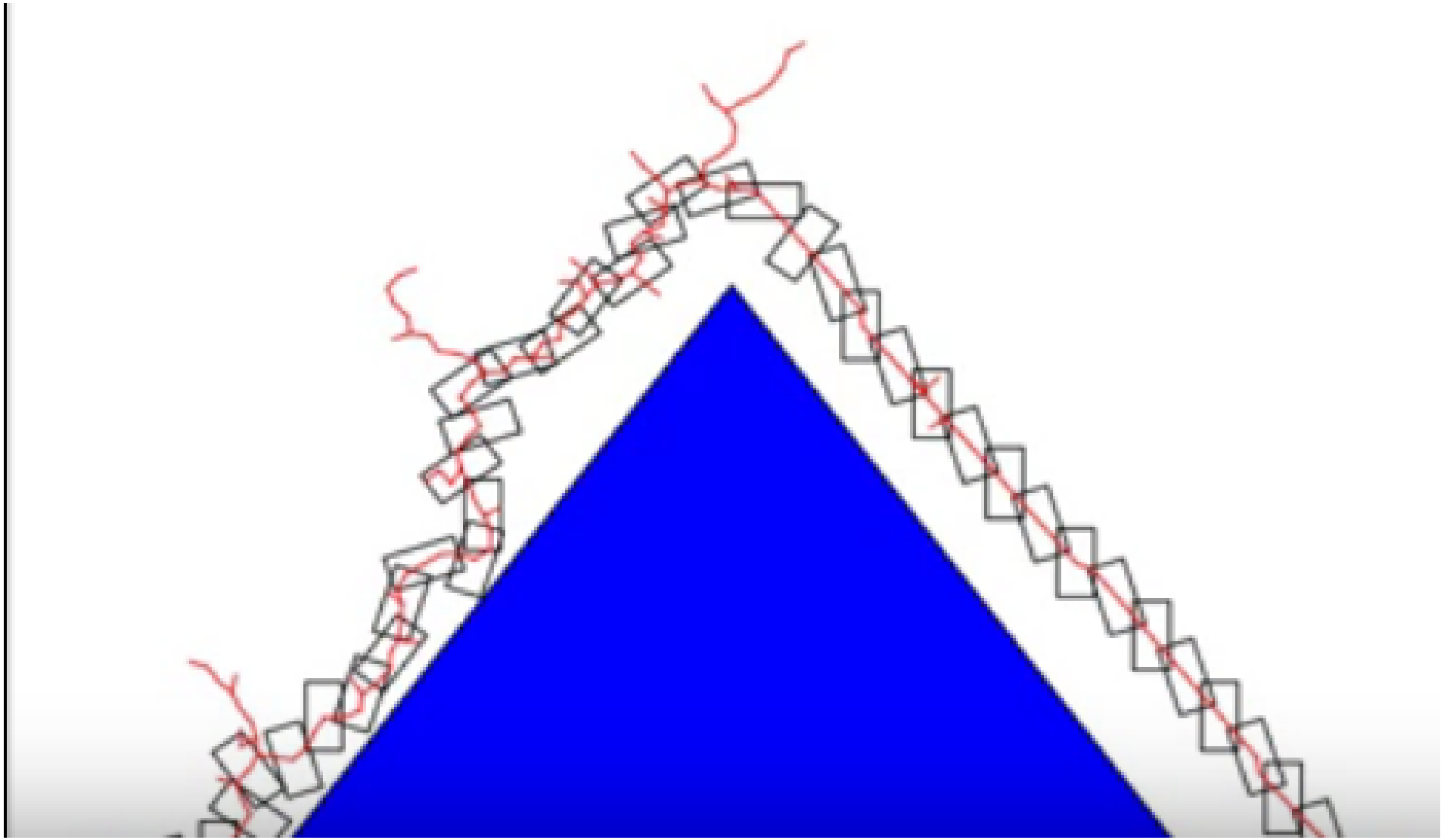
- Pros
  - Fast: can be generated in real time
  - Planning and controlling are merged into one function
- Cons
  - Can be trapped in local minima
  - Solved when used as local planners
  - Multiple solutions to local minima problem available in literature (e.g. Wang and Chirikjian 2000)





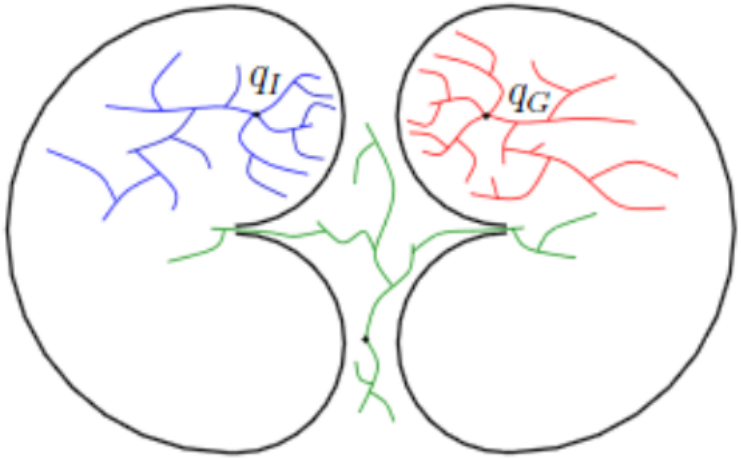
- Generic algorithm (non specific to robotics)
- Non convex, high dimensional spaces
- Tree is constructed incrementally with random valid samples
- Used widely for obstacle avoidance

RRT demo video[9]



[9] https://www.youtube.com/watch?v=rPgZyq15Z-Q

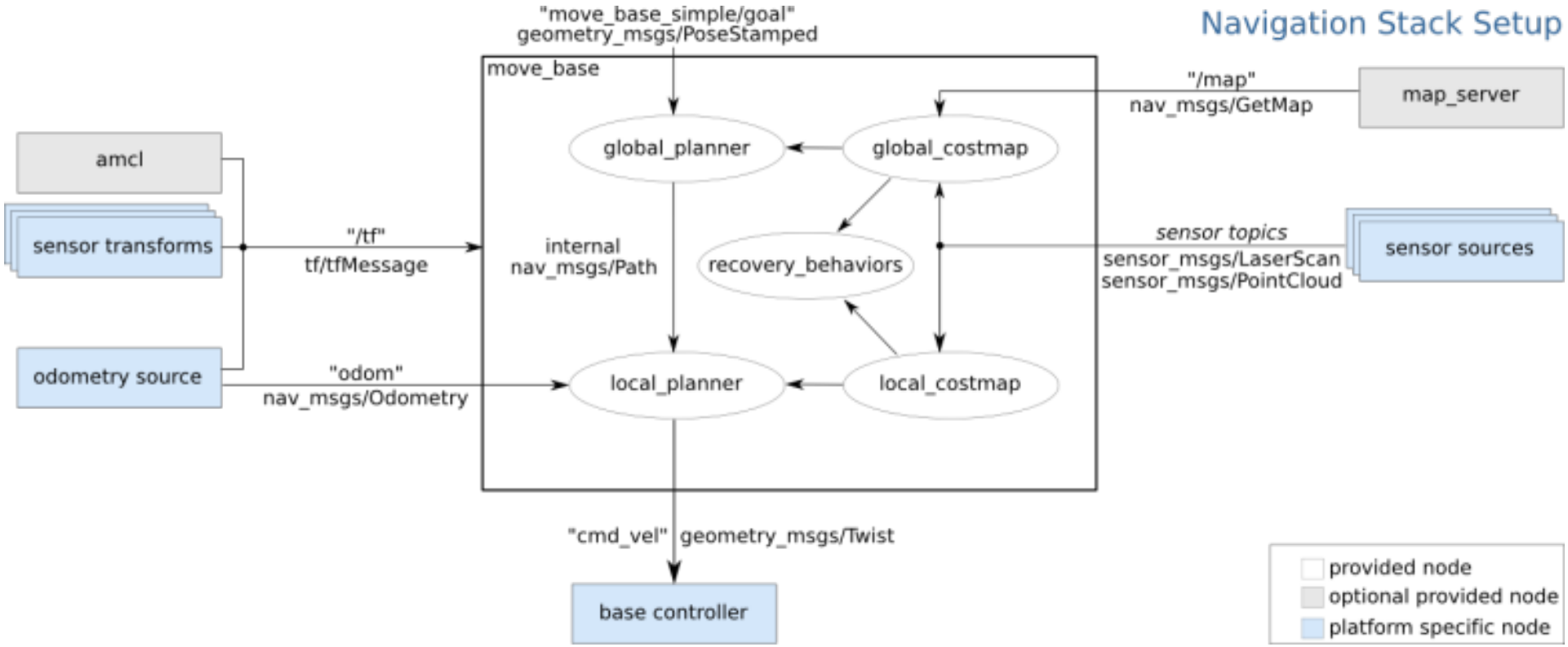
- Collision checking function required (usually boolean, is expensive)
- Generate random samples, discard if collision
- Keep distance between samples small to avoid collision between states, step $= \Delta q$
- Stop when both start and end nodes belong to graph
- Search resultant graph



bidirectional (blue + red) or multidirectional search

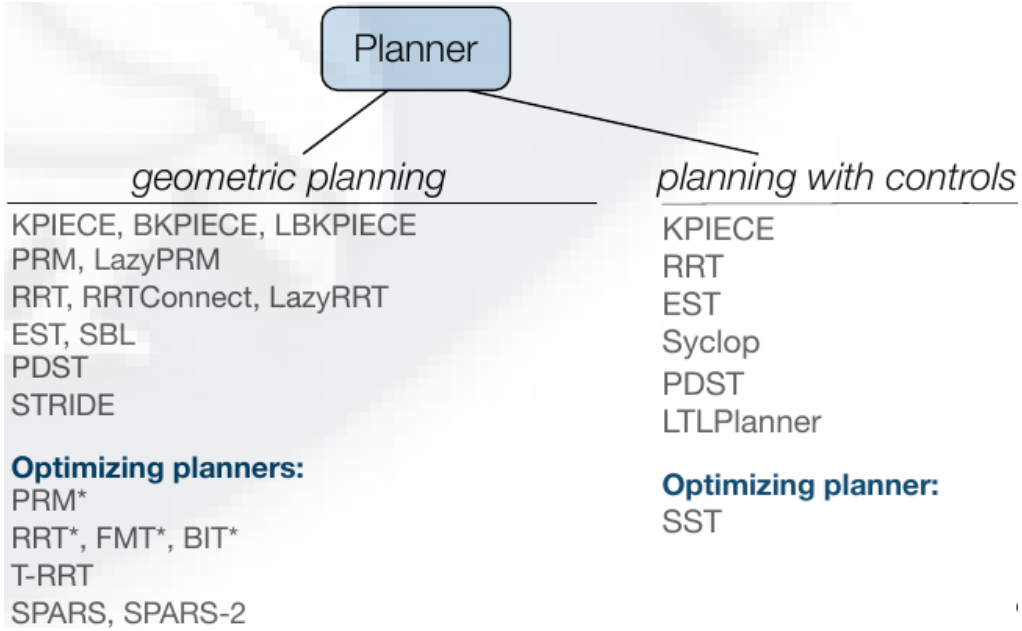
Motion Plan Execution

- ROS Navigation stack[10]



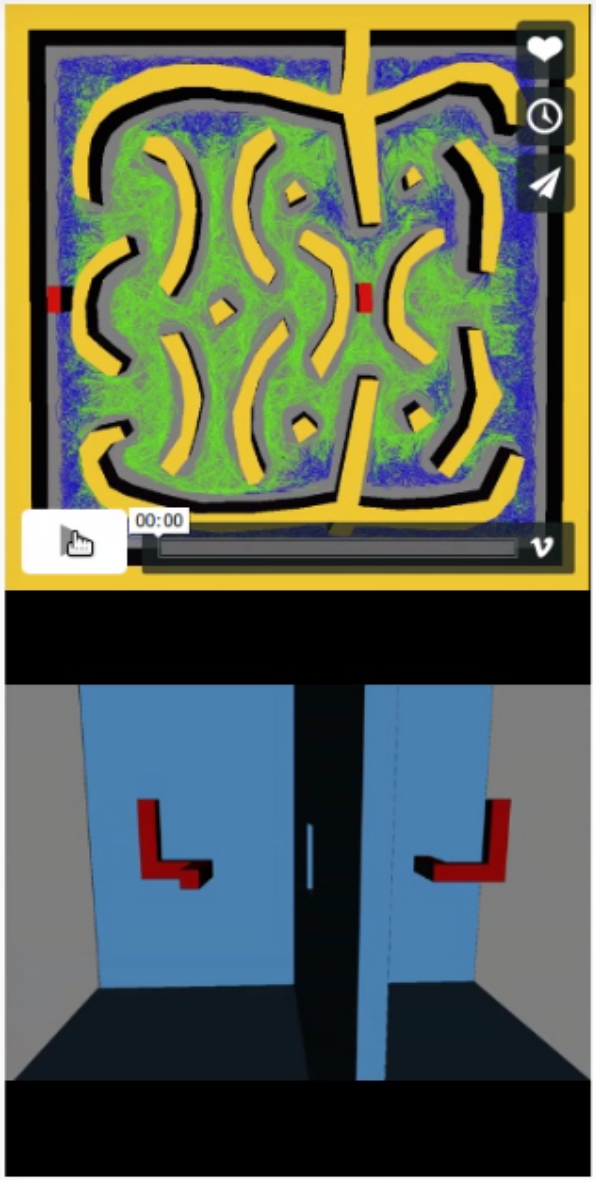
[10] https://youtu.be/OZ_-jAtGyZQ

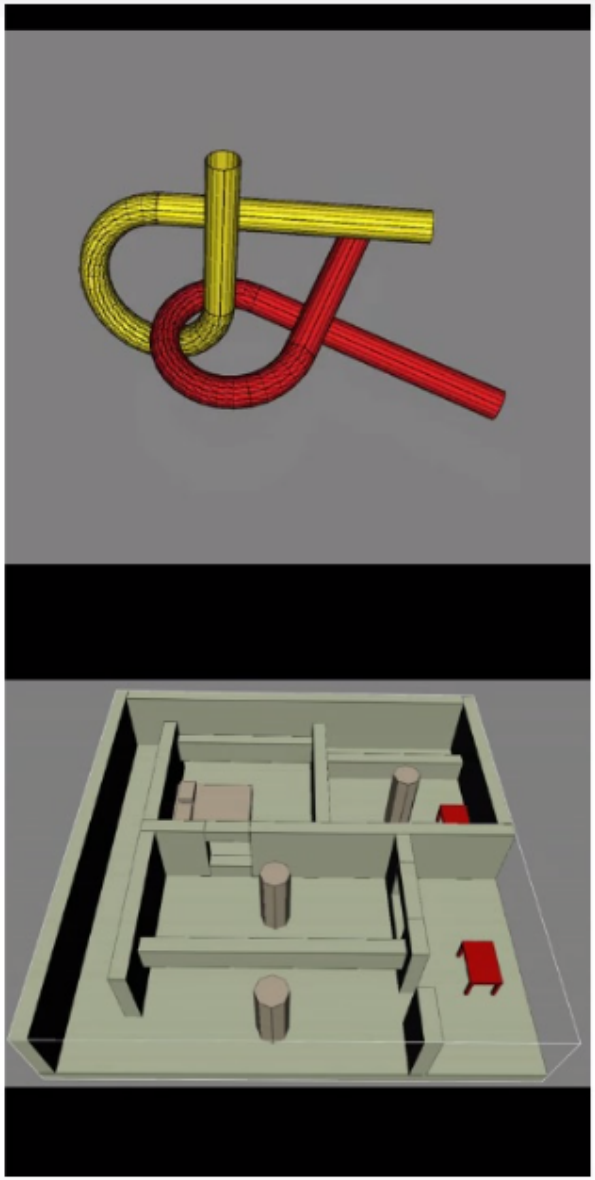
- Collection of multiple state-of-the-art sampling-based motion planning algorithms
- Offers a common ground for plan comparison (benchmark[11])
- Gui web version playground available in[12]
- Available planners:

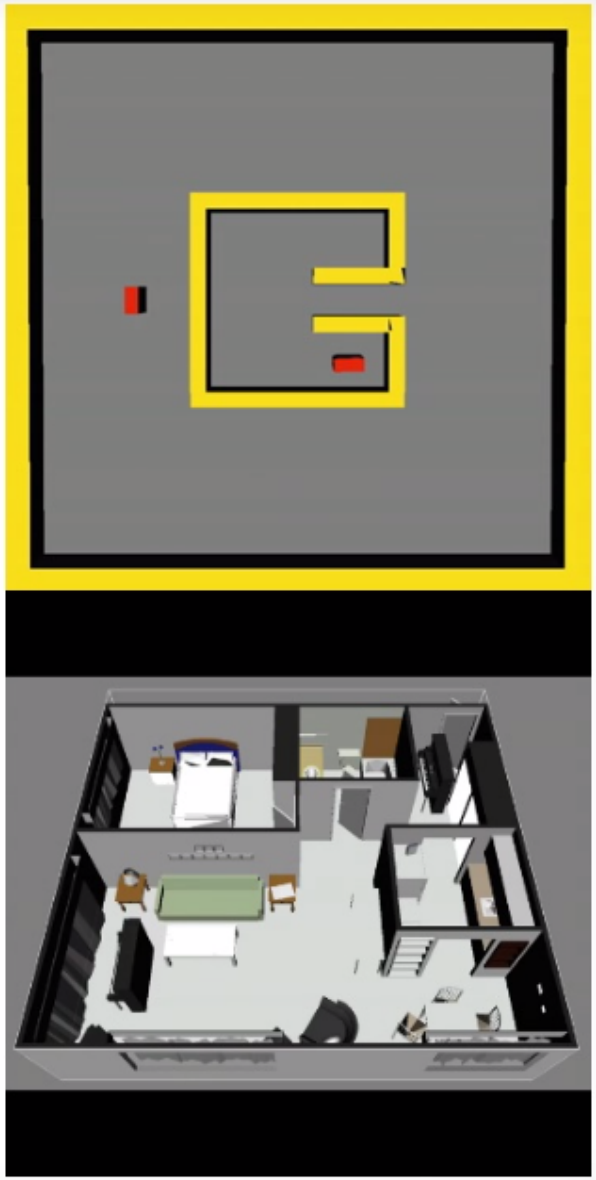


[11] http://plannerarena.org/
[12] http://omplapp.kavrakilab.org/
[13] http://ompl.kavrakilab.org/

OMPL Gui app demo



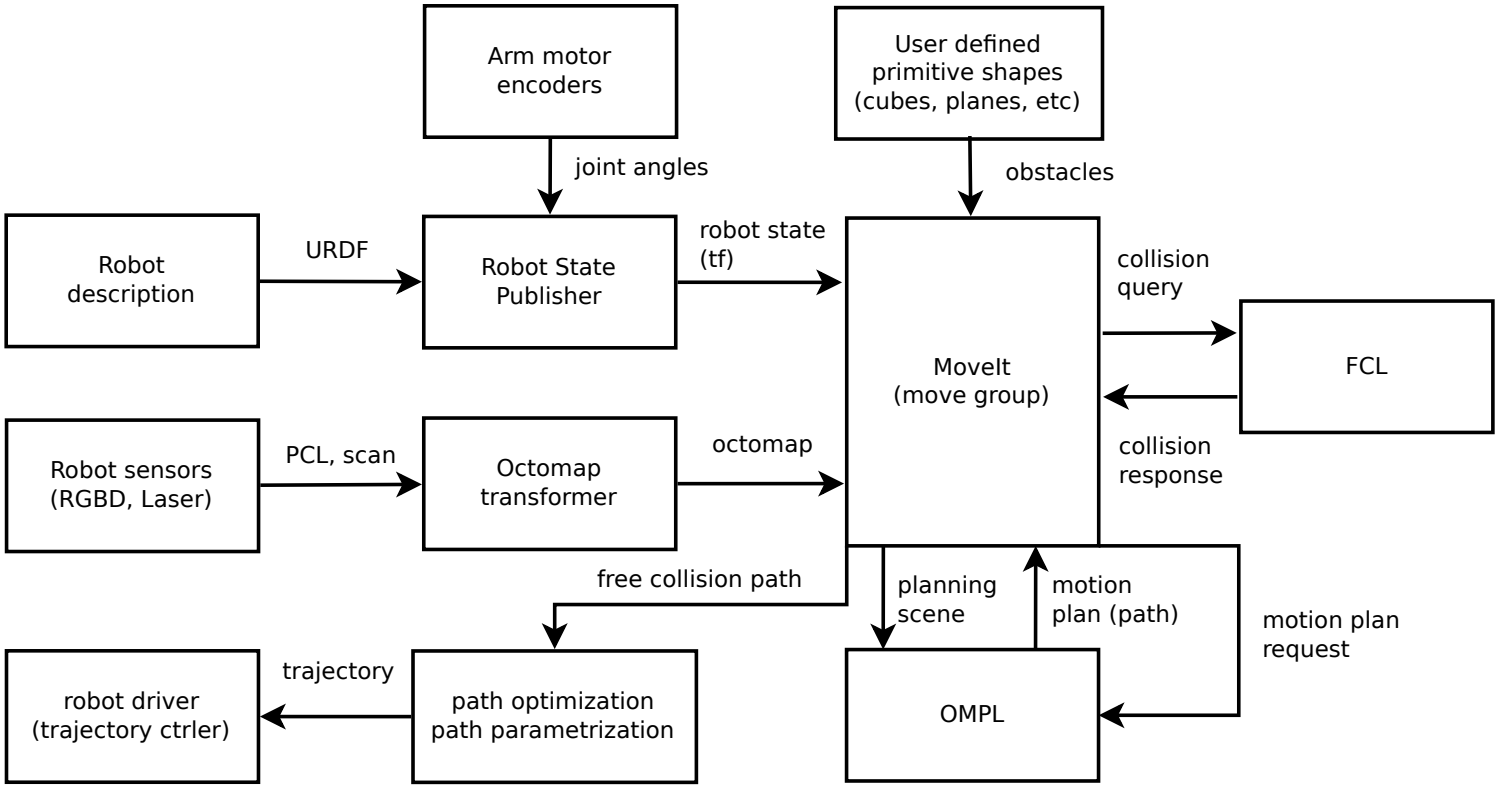




OMPL Planning with dynamic constraints demo[14]

[14] http://ompl.kavrakilab.org/gallery.html

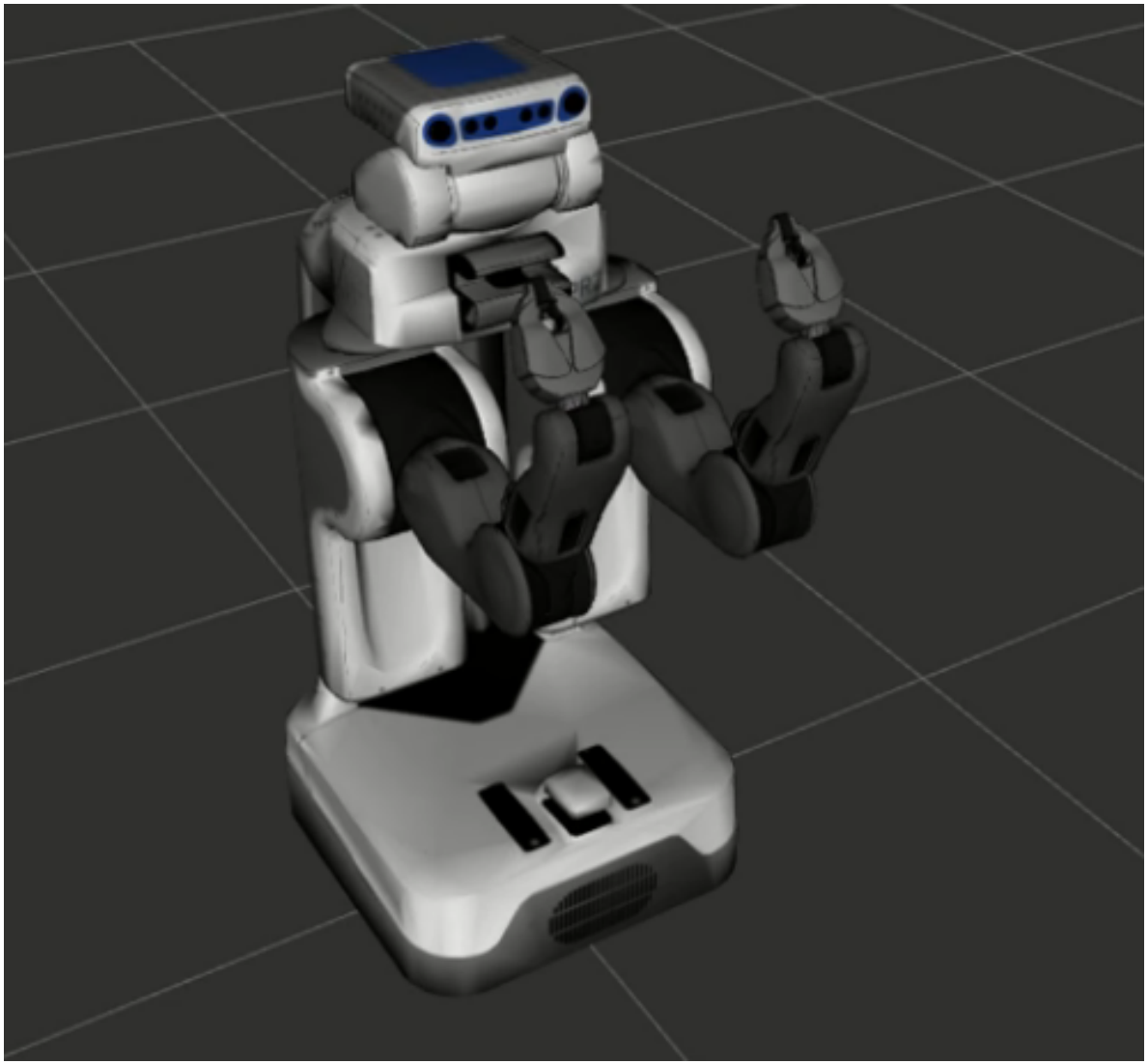
- Manipulation framework for robotics
- Integrates all required components for motion planning execution on real robots
- ROSCON 2012 MoveIt Introduction video[15], architecture diagram:



[15] https://youtu.be/r1zbuLc8RhI

[16] https://moveit.ros.org/

MoveIt! Demo, Plan execution (video)[17]



[17] https://youtu.be/4FSmZRQh37Q

- Picking challenge!
- See[18]

[18] https://opensource.fetchrobotics.com/icra-challenge/2019/01/28/tutorial.html

Next lecture contents

- Planning tutorial
  - A*
  - STRIPS (hands on)
  - and more...

Thank You for Your Attention.
Do You Have Questions?

References I

[Mur+16] Sean Murray et al. "Robot Motion Planning on a Chip." In: *Robotics: Science and Systems*. 2016.

[WC00] Yunfeng Wang and Gregory S Chirikjian. "A new potential field method for robot path planning". In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*. Vol. 2. IEEE. 2000, pp. 977–982.