上海科技大学
ShanghaiTech University

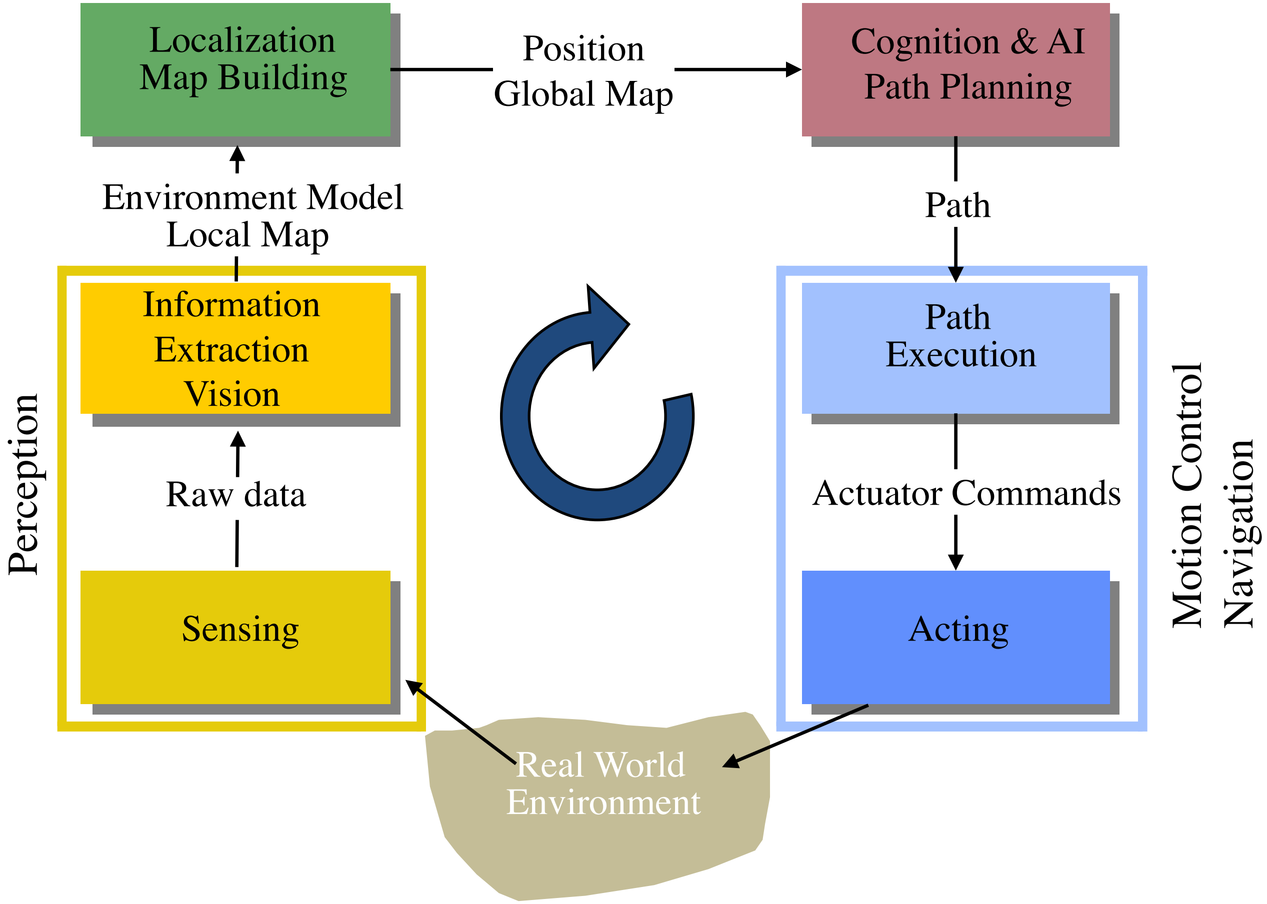
CS283: **Robotics Fall 2017: Robot Arms**

Sören Schwertfeger / 师泽仁

ShanghaiTech University

REVIEW

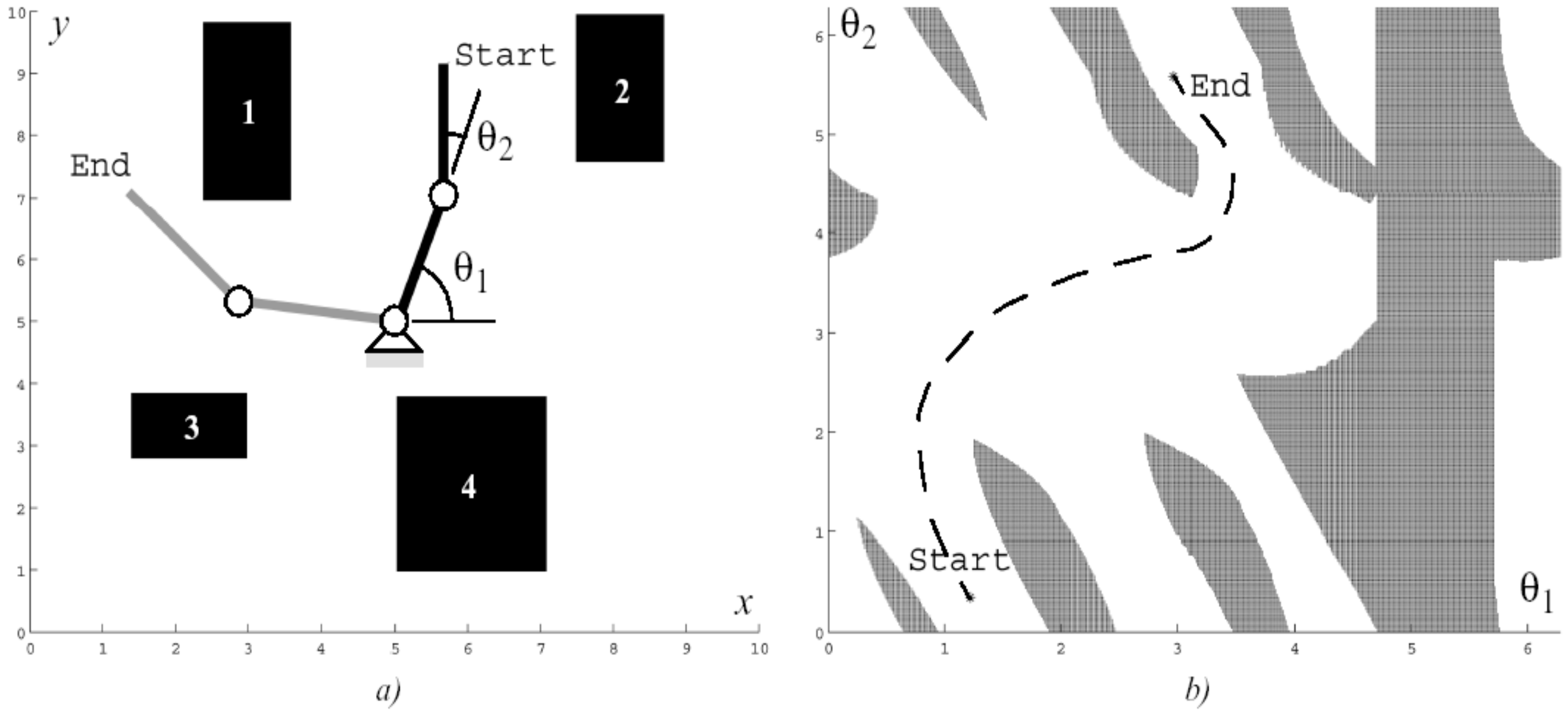
General Control Scheme for Mobile Robot Systems



With material from Roland Siegwart and Davide Scaramuzza, ETH Zurich

Work Space (Map) → Configuration Space

- State or configuration q can be described with k values q_i



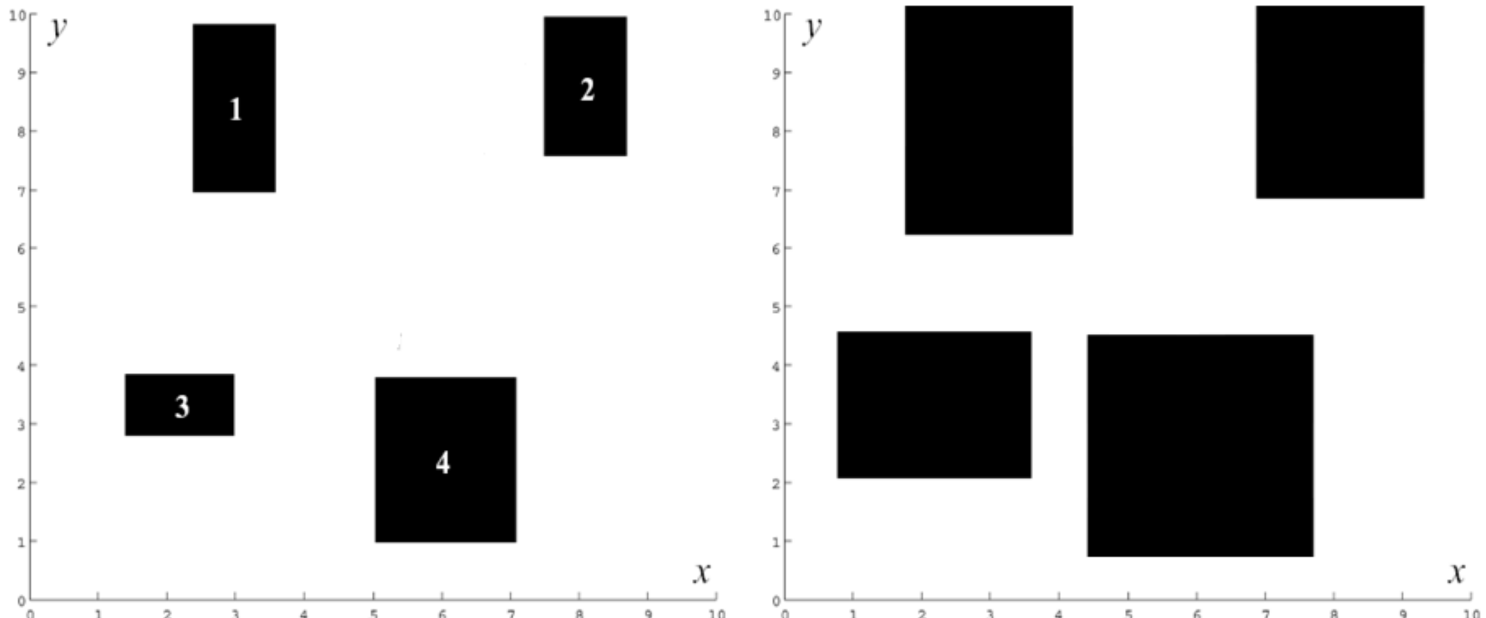
Work Space

Configuration Space:
the dimension of this
space is equal to the Degrees of Freedom (DoF)
of the robot

- What is the configuration space of a mobile robot?

Configuration Space for a Mobile Robot

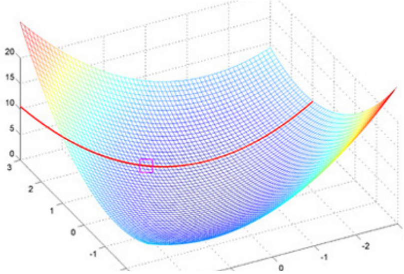
- Mobile robots operating on a flat ground (2D) have 3 DoF: (x, y, θ)
- Differential Drive: only two motors => only 2 degrees of freedom directly controlled (forward/ backward + turn) => non-holonomic
- Simplification: assume robot is holonomic and it is a point => configuration space is reduced to 2D (x,y)
- => inflate obstacle by size of the robot radius to avoid crashes => obstacle growing



Path Planning: Overview of Algorithms

1. Optimal Control

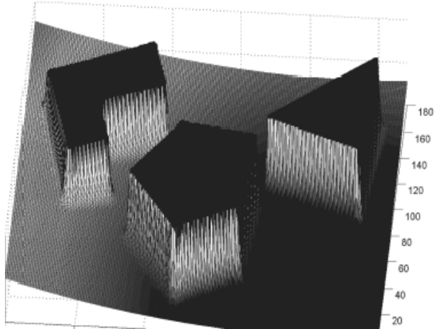
- Solves truly optimal solution
- Becomes intractable for even moderately complex as well as nonconvex problems



Source: http://mitocw.udsm.ac.tz

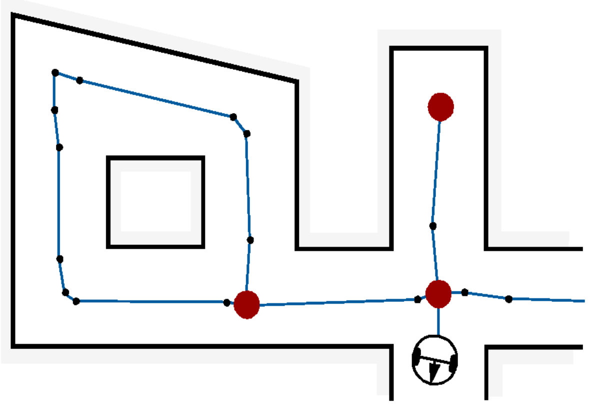
2. Potential Field

- Imposes a mathematical function over the state/configuration space
- Many physical metaphors exist
- Often employed due to its simplicity and similarity to optimal control solutions

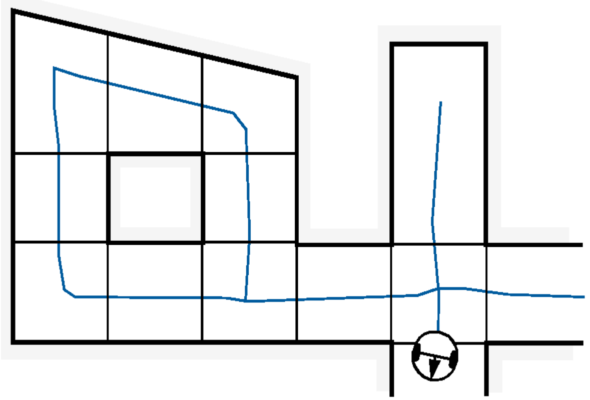


3. Graph Search

- Identify a set edges between nodes within the free space



- Where to put the nodes?



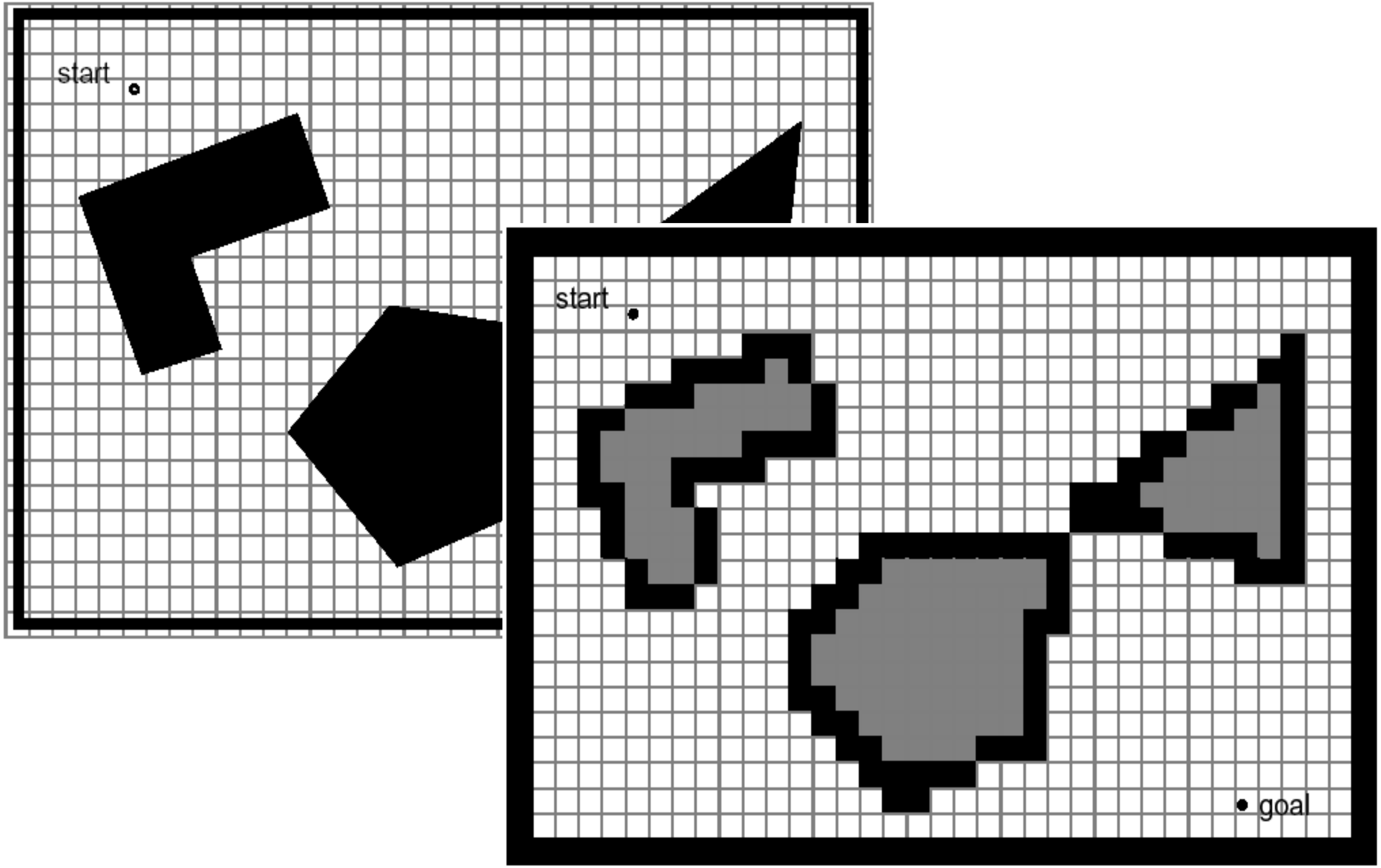
Graph Search

- Overview
  - Solves a least cost problem between two states on a (directed) graph
  - Graph structure is a discrete representation

- Limitations
  - State space is discretized → completeness is at stake
  - Feasibility of paths is often not inherently encoded

- Algorithms
  - (Preprocessing steps)
  - Breath first
  - Depth first
  - Dijkstra
  - A* and variants
  - D* and variants

Graph Construction: Approximate Cell Decomposition (3/4)



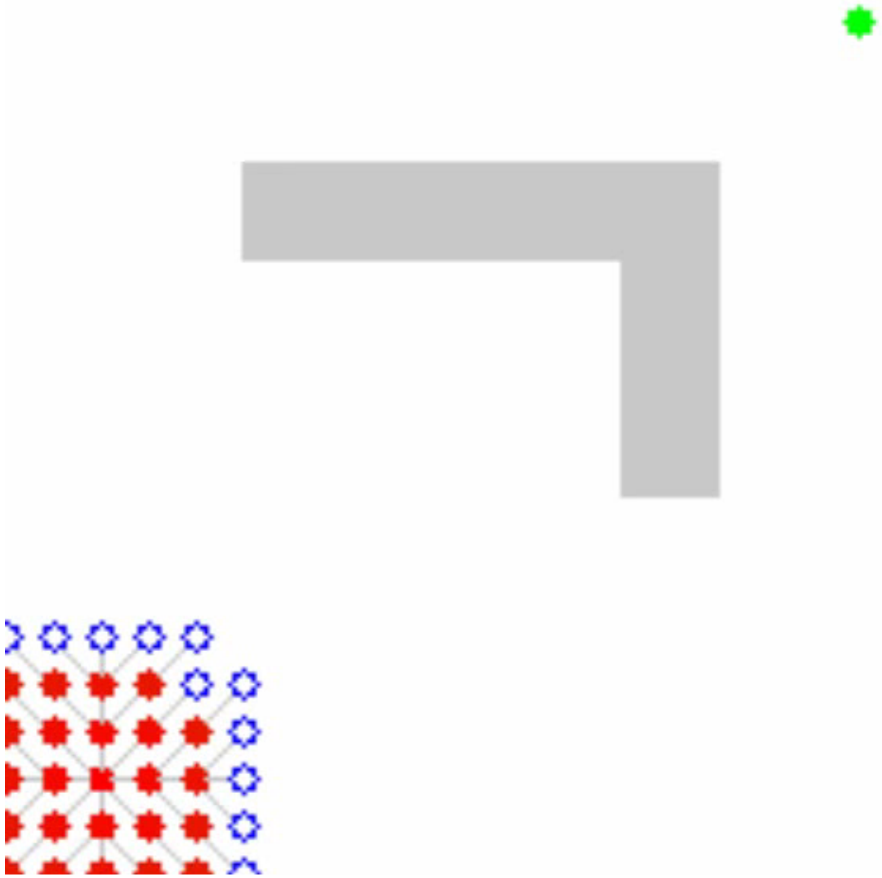
Diklstra's Algorithm

- Assign all vertices infinite distance to goal
- Assign 0 to distance from start
- Add all vertices to the queue

- While the queue is not empty:
  - Select vertex with smallest distance and remove it from the queue
  - Visit all neighbor vertices of that vertex,
  - calculate their distance and
  - update their (the neighbors) distance if the new distance is smaller

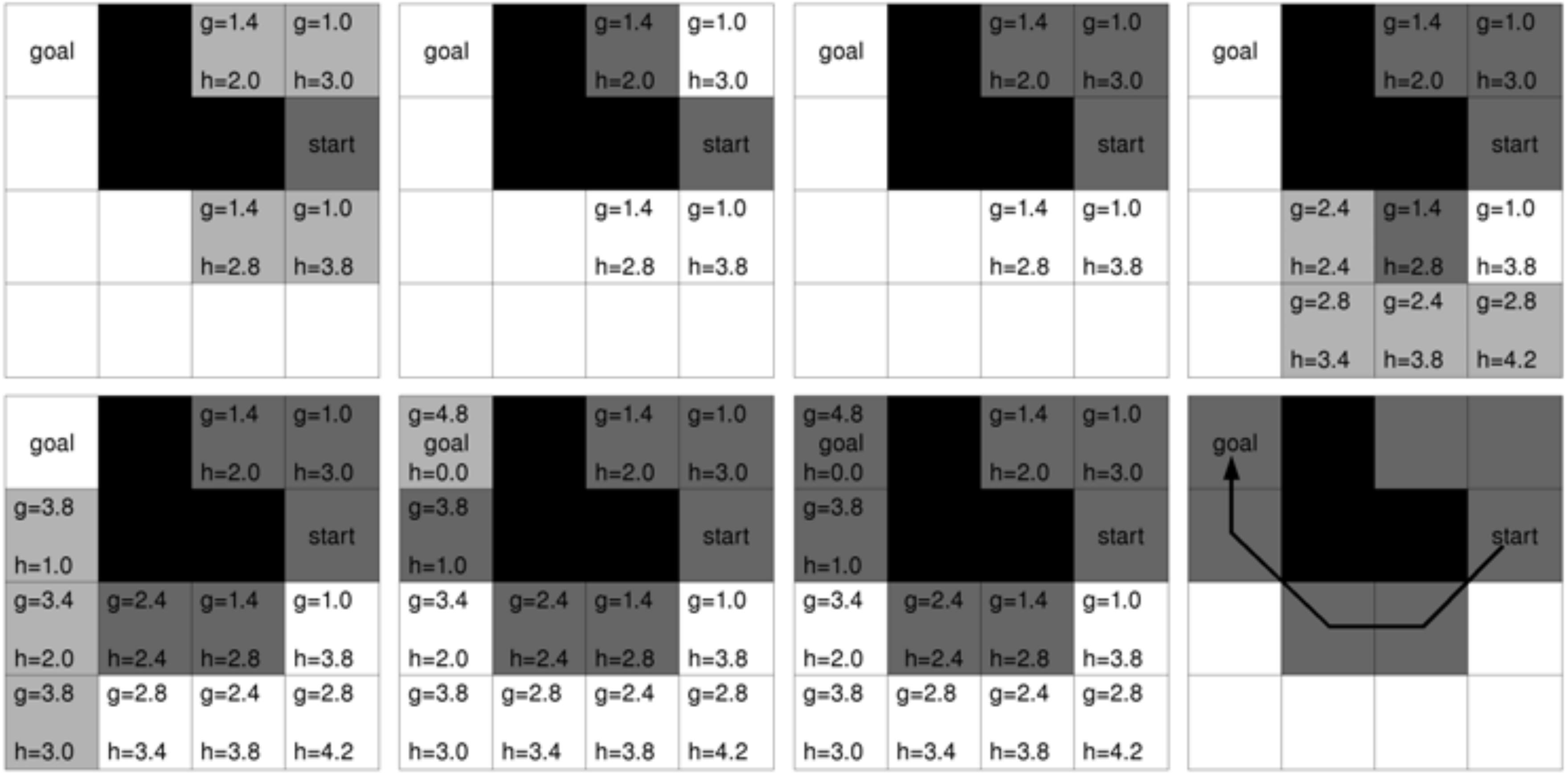
Dijkstra's Algorithm for Path Planning: Grid Maps

- Graph:
  - Neighboring free cells are connected:
    - 4-neighborhood: up/ down/ left right
    - **8-neighborhood**: also diagonals
  - All edges have weight 1
- Stop once goal vertex is reached
- Per vertex: save edge over which the shortest distance from start was reached => Path



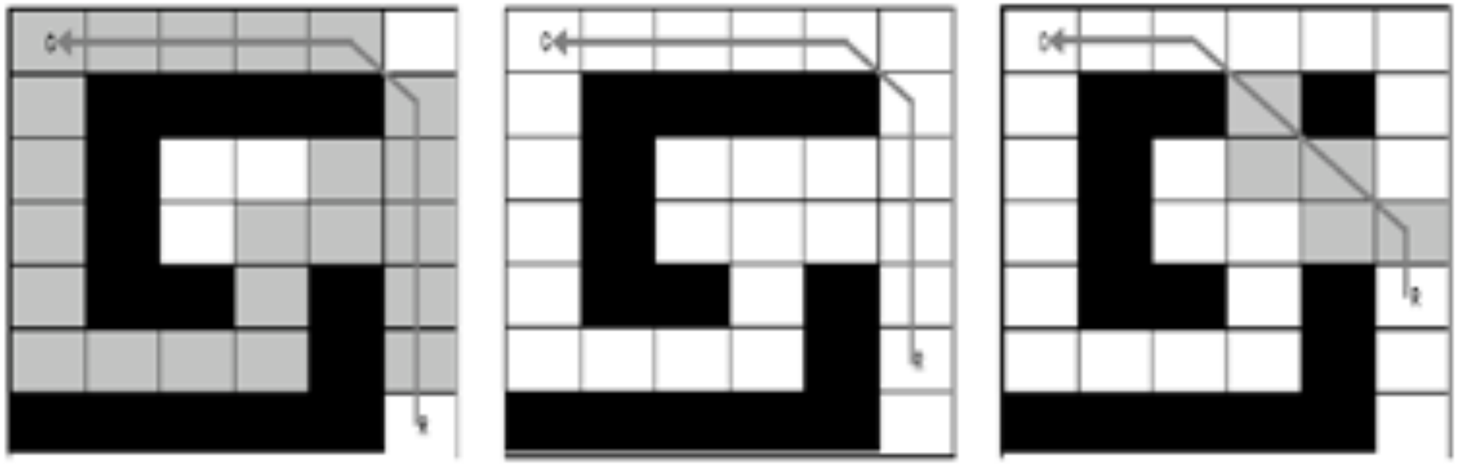
Graph Search Strategies: A* Search

- Similar to Dijkstra's algorithm, except that it uses a heuristic function h(n)
- $f(n) = g(n) + \varepsilon\, h(n)$



Graph Search Strategies: D* Search

- Similar to A* search, except that the search starts from the goal outward
- f(n) = g(n) + ε h(n)
- First pass is identical to A*
- Subsequent passes reuse information from previous searches

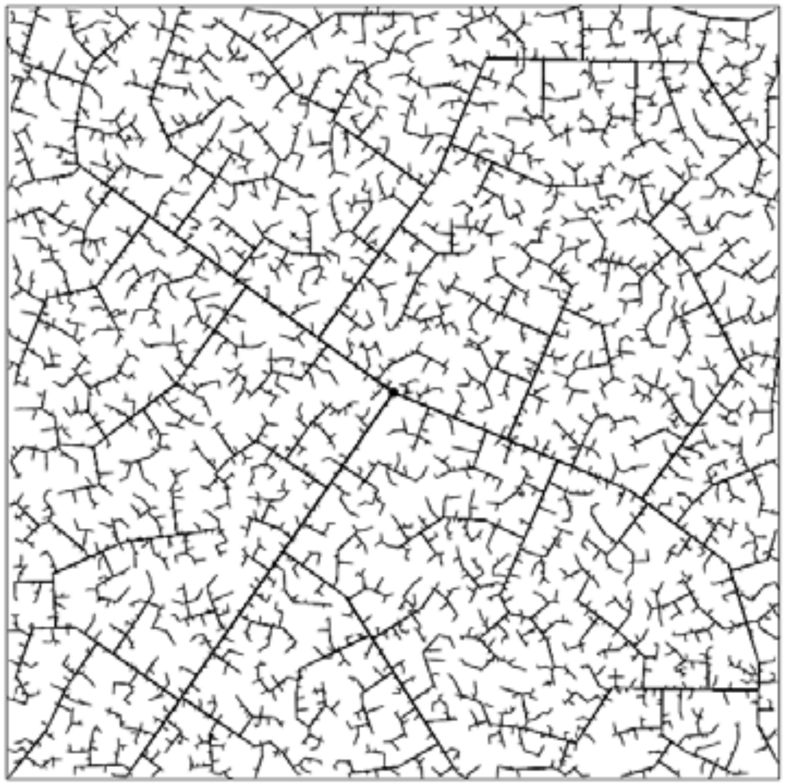


$\epsilon = 1.0$ $\epsilon = 1.0$ $\epsilon = 1.0$

Graph Search Strategies: Randomized Search

- Most popular version is the rapidly exploring random tree (RRT)
  - Well suited for high-dimensional search spaces
  - Often produces highly suboptimal solutions

45 iterations



2345 iterations

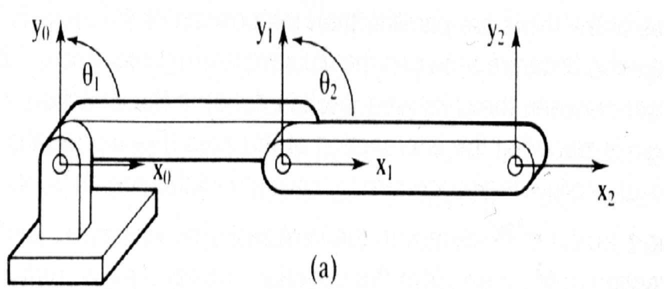
ROBOT ARMS

Robot Arm

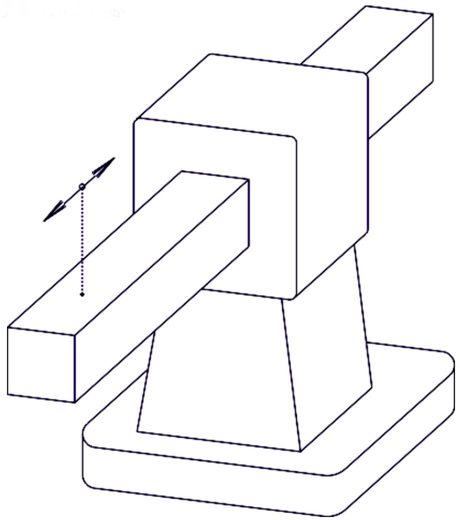
- Consists of Joints and Links …
- and a Base and a Tool (or End-Effector or Tip)

Joints

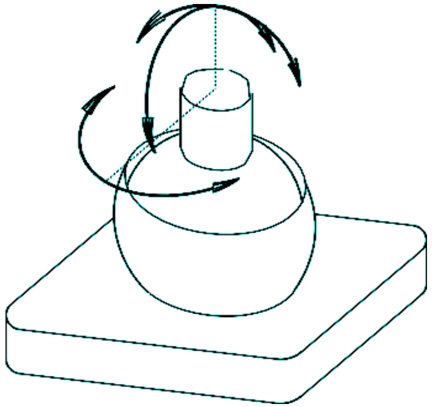
- Revolute Joint: 1DOF



- Prismatic Joint/ Linear Joint: 1DOF



- Spherical Joint: 3DOF



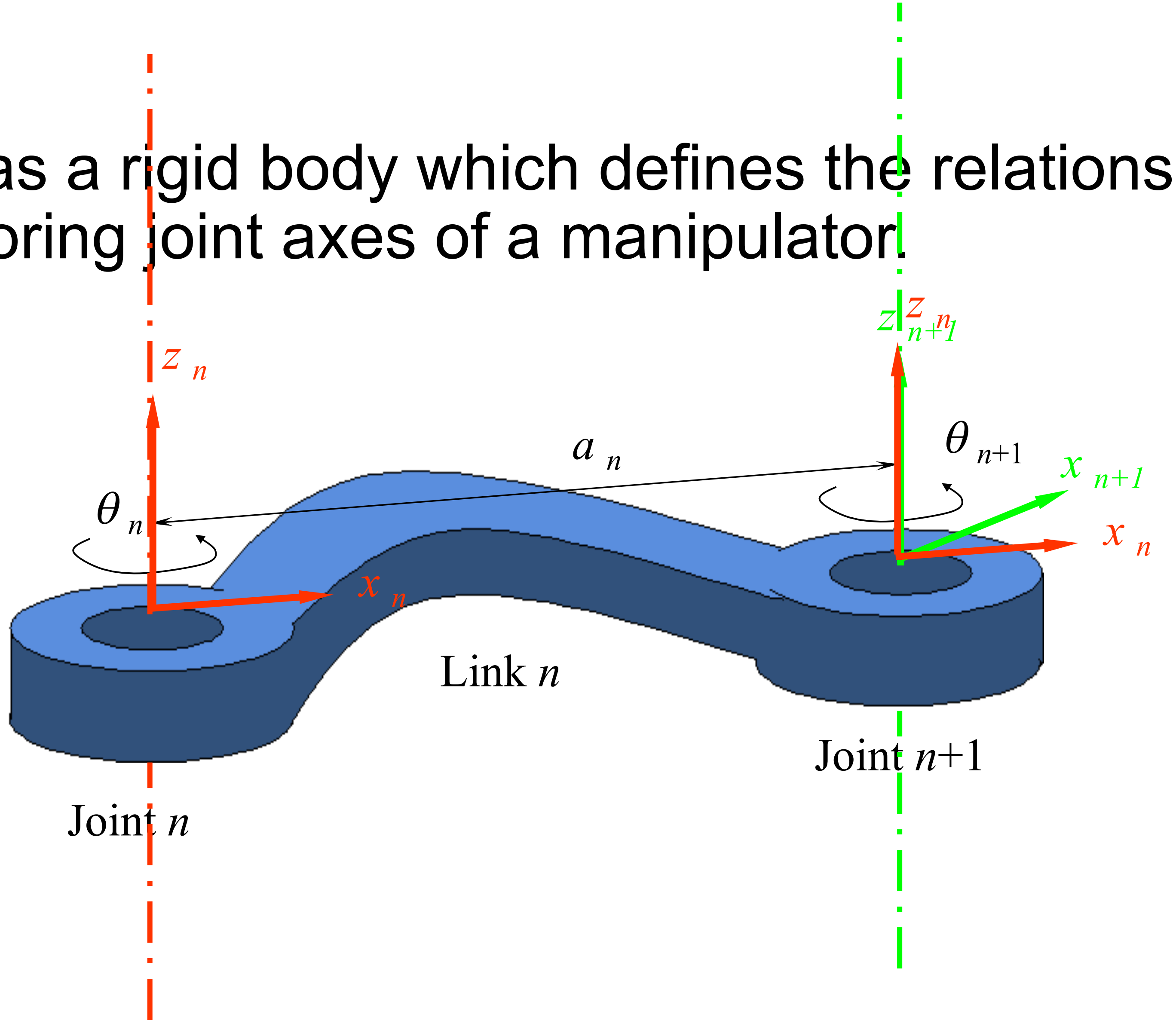
Note on Joints

- Without loss of generality, we will consider only manipulators which have joints with a single degree of freedom.

- A joint having n degrees of freedom can be modeled as n joints of one degree of freedom connected with n-1 links of zero length.

Link

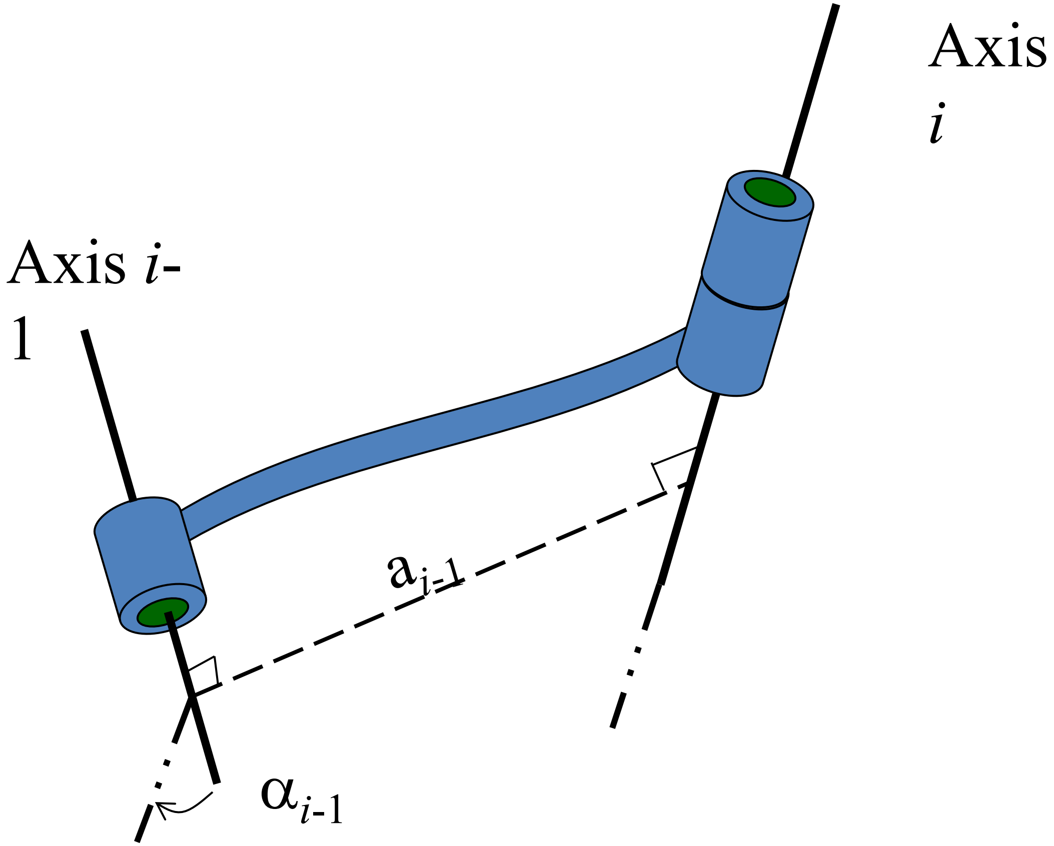
- A link is considered as a rigid body which defines the relationship between two neighboring joint axes of a manipulator.



The Kinematics Function of a Link

- The kinematics function of a link is to maintain a fixed relationship between the two joint axes it supports.

- This relationship can be described with two parameters: the link length a, the link twist a



Link Length

- Is measured along a line which is mutually perpendicular to both axes.

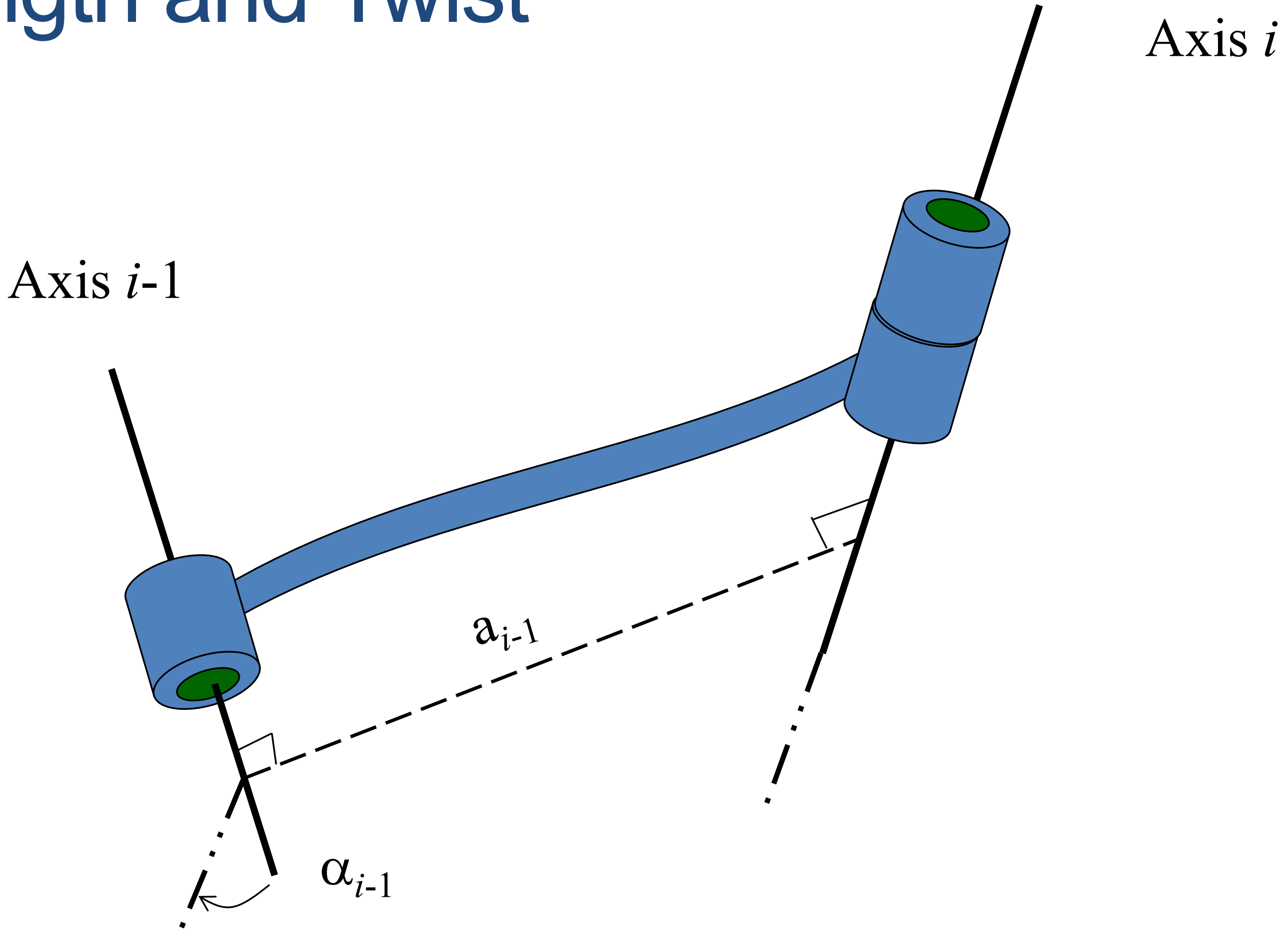
- The mutually perpendicular always exists and is unique except when both axes are parallel.

Link Twist

- Project both axes i-1 and i onto the plane whose normal is the mutually perpendicular line, and measure the angle between them

- Right-hand coordinate system

Link Length and Twist

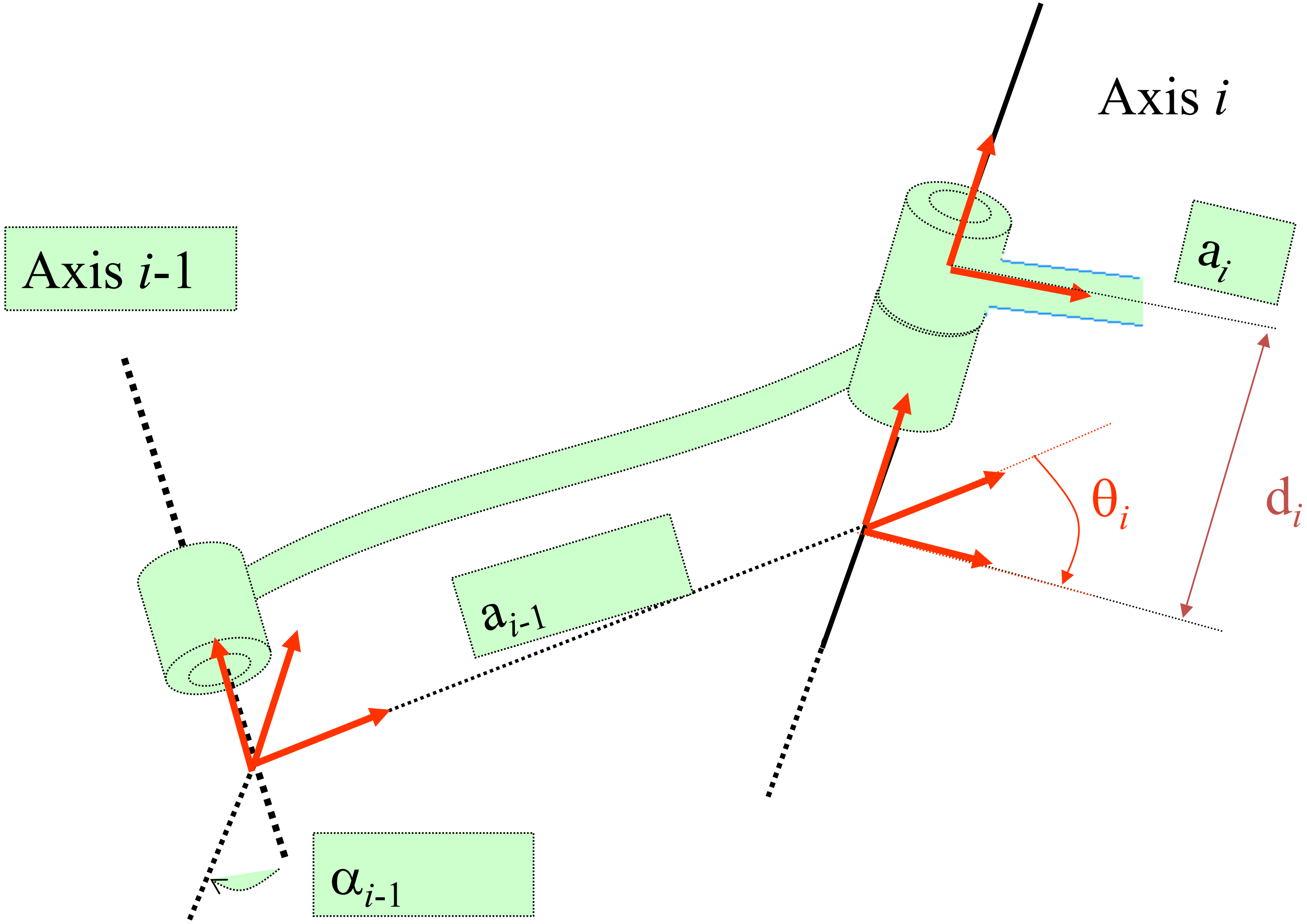


Joint Parameters

A joint axis is established at the connection of two links. This joint will have two normals connected to it one for each of the links.

- The relative position of two links is called *link offset* d_n whish is the distance between the links (the displacement, along the joint axes between the links).
- The *joint angle* θ_n between the normals is measured in a plane normal to the joint axis.

Link and Joint Parameters



Link and Joint Parameters

4 parameters are associated with each link. You can align the two axis using these parameters.

- Link parameters:

a_n the length of the link.

α_n the twist angle between the joint axes.

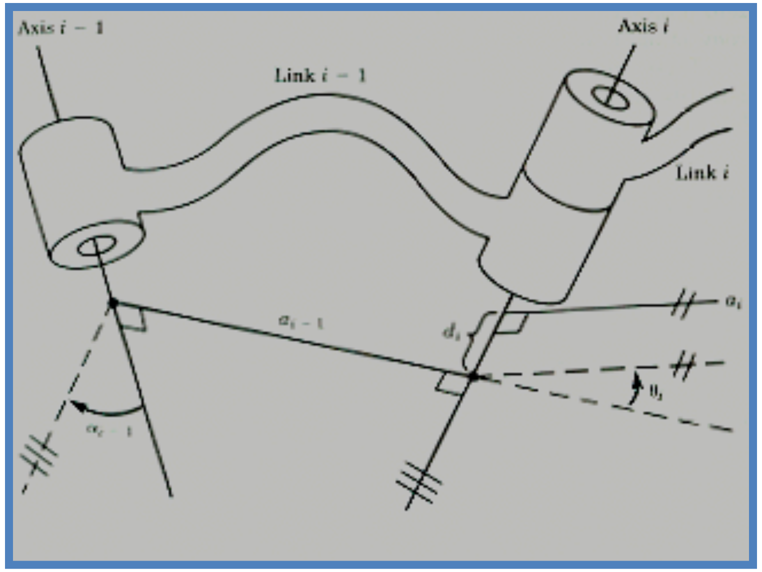
- Joint parameters:

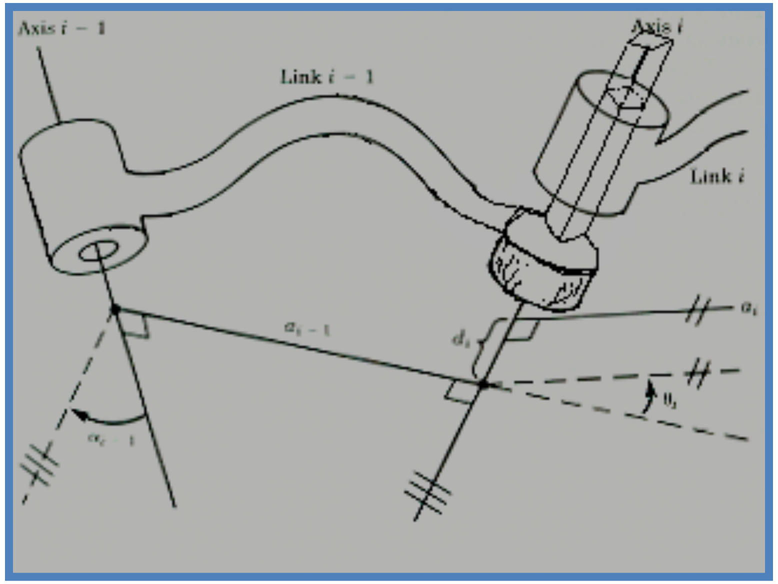
θ_n the angle between the links.

d_n the distance between the links

Link Connection Description:

For Revolute Joints: a, α, and d. are all fixed, then "θ_i" is the. Joint Variable.



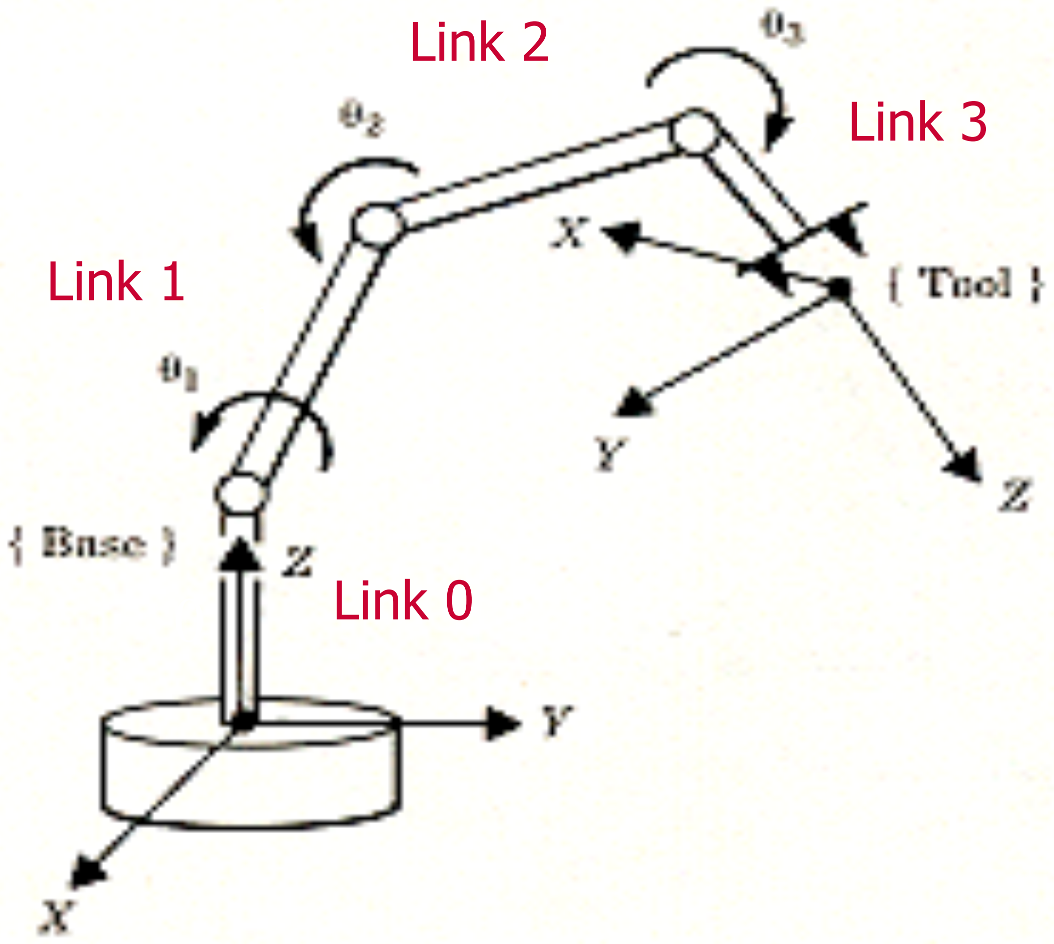


For Prismatic Joints: a, α, and θ. are all fixed, then "d_i" is the. Joint Variable.

These four parameters: (Link-Length a_{i-1}), (Link-Twist α_{i-1}), (Link-Offset d_i), (Joint-Angle θ_i) are known as the *Denavit-Hartenberg Link Parameters*.

Links Numbering Convention

Base of the arm:	**Link-0**
1st moving link:	**Link-1**
.	**.**
.	**.**
.	**.**
Last moving link:	**Link-n**



A 3-DOF Manipulator Arm

First and Last Links in the Chain

- $a_{0=}\ \alpha_{n=0.0}$

- $\alpha_{0=}\ \alpha_{n=0.0}$

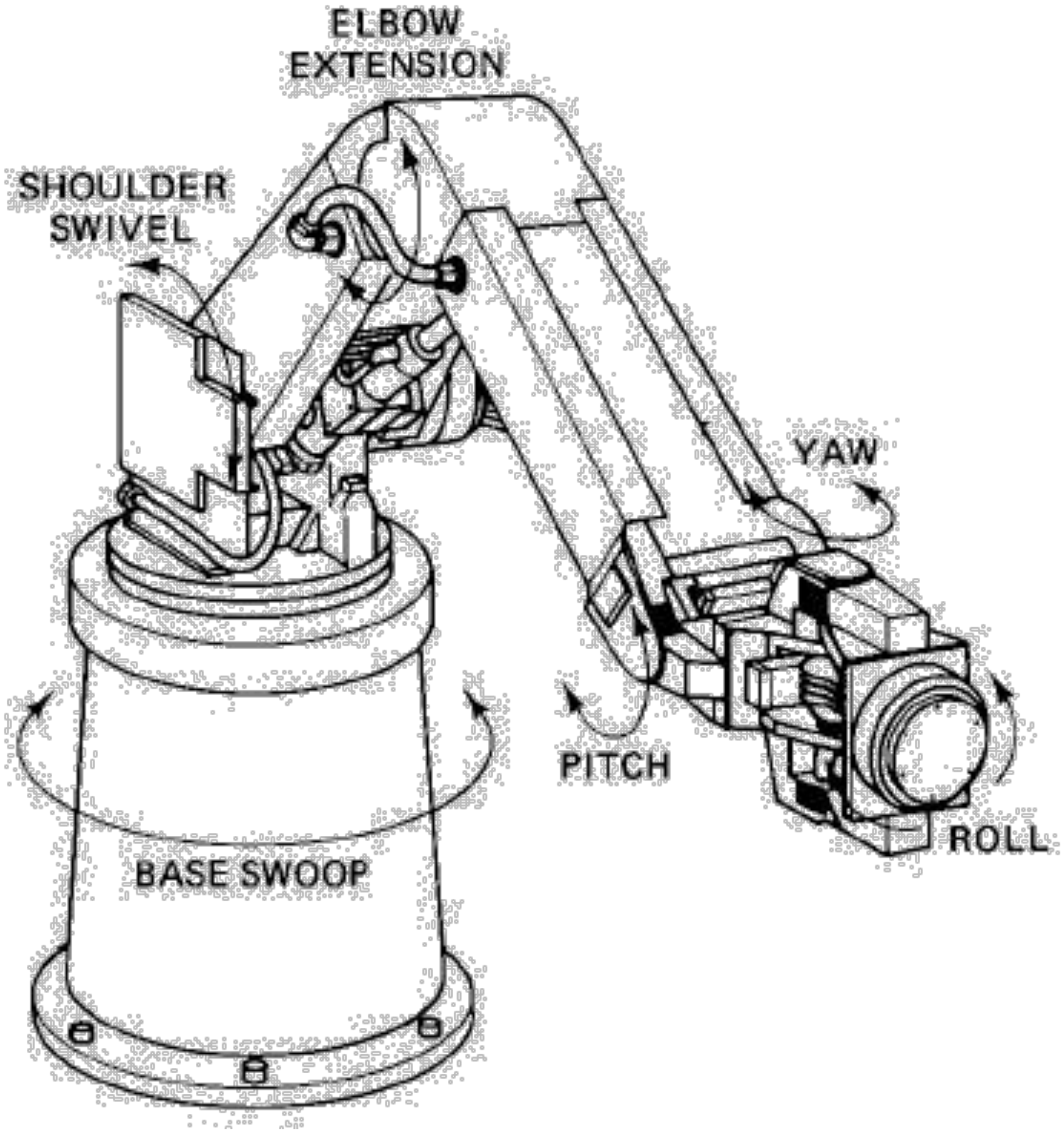
- *If joint 1 is revolute:* $d_{0=}\ 0$ *and* θ_1 *is arbitrary*

- *If joint 1 is prismatic:* $d_{0=}$ *arbitrary and* $\theta_1 = 0$

Robot Specifications

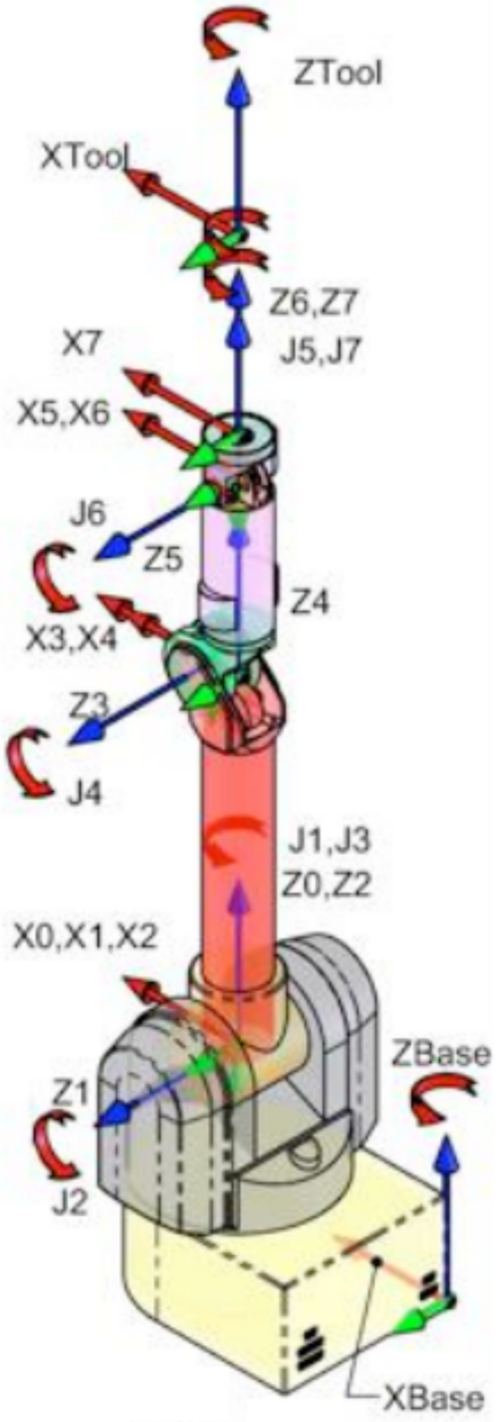
Number of axes

- Major axes, (1-3) => position the wrist
- Minor axes, (4-6) => orient the tool
- Redundant, (7-n) => reaching around obstacles, avoiding undesirable configuration



Frames

- Choose the base and tool coordinate frame
  - Make your life easy!
- Several conventions
  - Denavit Hartenberg (DH), modified DH, Hayati, etc.



KINEMATICS

Kinematics

Forward Kinematics (angles to position) (it is straight-forward -> easy)

What you are given: The length of each link
The angle of each joint

What you can find: The position of any point (i.e. it's (x, y, z) coordinates)

Inverse Kinematics (position to angles) (more difficult)

What you are given: The length of each link
The position of some point on the robot

What you can find: The angles of each joint needed to obtain that position

Kinematics

Cartesian Space

Tool Frame (E)
(aka End-Effector)
Base Frame (B)

$${}^{B}_{E}T = \begin{Bmatrix} {}^{B}_{E}t \\ {}^{B}_{E}R \end{Bmatrix}$$

Forward Kinematics

$${}^{B}_{E}T = f(q)$$

$$q = f^{-1}({}^{B}_{E}T)$$

Inverse Kinematics

Joint Space

Joint 1 = q_1
Joint 2 = q_2
Joint 3 = q_3
...
Joint n = q_n

Rigid body transformation
Between coordinate frames

Linear algebra

Kinematics: Velocities

Jacobian

$${}^{B}_{E}V = J(q)\dot{q}$$

Cartesian Space
Tool Frame (E)
(aka End-Effector)
Base Frame (B)

$${}^{B}_{E}V = \begin{Bmatrix} {}^{B}_{E}v \\ {}^{B}_{E}w \end{Bmatrix}$$

v: linear velocity
w: angular velocity

Joint Space

Joint 1 = $\dot{q}_1$
Joint 2 = $\dot{q}_2$
Joint 3 = $\dot{q}_3$
...
Joint n = $\dot{q}_n$

$$\dot{q} = J^{-1}(q)\ {}^{B}_{E}V$$

Inverse Jacobian

Rigid body transformation
Between coordinate frames

Linear algebra

INVERSE KINEMATICS (IK)

Inverse Kinematics (IK)

- Given end effector position, compute required joint angles
- In simple case, analytic solution exists
  - Use trig, geometry, and algebra to solve

- Generally (more DOF) difficult
  - Use Newton’s method
  - Often more than one solution exist!

- Analytic solution of 2-link inverse kinematics

$$x^2 + y^2 = a_1^2 + a_2^2 - 2a_1a_2\cos(\pi - \theta_2)$$

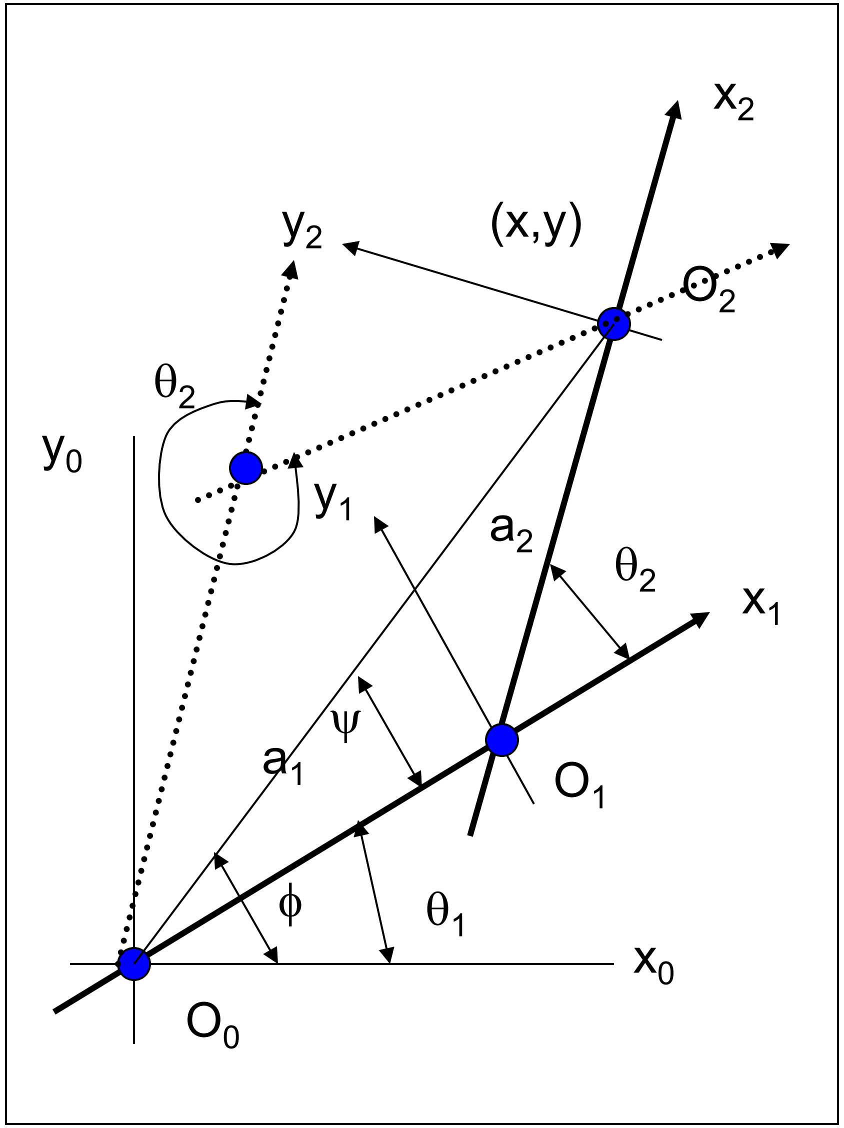
$$\cos\theta_2 = \frac{x^2 + y^2 - a_1^2 - a_2^2}{2a_1a_2}$$

for greater accuracy

$$\tan^2\frac{\theta_2}{2} = \frac{1-\cos\theta}{1+\cos\theta} = \frac{2a_1a_2 - x^2 - y^2 + a_1^2 + a_2^2}{2a_1a_2 + x^2 + y^2 - a_1^2 - a_2^2}$$

$$= \frac{\left(a_1^2 + a_2^2\right)^2 - \left(x^2 + y^2\right)}{\left(x^2 + y^2\right) - \left(a_1^2 - a_2^2\right)^2}$$

$$\theta_2 = \pm 2\tan^{-1}\sqrt{\frac{\left(a_1^2 + a_2^2\right)^2 - \left(x^2 + y^2\right)}{\left(x^2 + y^2\right) - \left(a_1^2 - a_2^2\right)^2}}$$



- Two solutions: elbow up & elbow down

Iterative IK Solutions

- Frequently analytic solution is infeasible
- Use Jacobian
- Derivative of function output relative to each of its inputs
- If y is function of three inputs and one output

$$y = f(x_1, x_2, x_3)$$

$$\delta y = \frac{\delta f}{\partial x_1} \cdot \delta x_1 + \frac{\delta f}{\partial x_2} \cdot \delta x_2 + \frac{\delta f}{\partial x_3} \cdot \delta x_3$$

- Represent Jacobian J(X) as a 1x3 matrix of partial derivatives

Jacobian

- In another situation,
  end effector has 6 DOFs and
  robotic arm has 6 DOFs

- $f(x_1, \ldots, x_6) = (x, y, z, r, p, y)$

- Therefore J(X) = 6x6 matrix

$$\begin{bmatrix} \frac{\partial f_x}{\partial x_1} & \frac{\partial f_y}{\partial x_1} & \frac{\partial f_z}{\partial x_1} & \frac{\partial f_r}{\partial x_1} & \frac{\partial f_p}{\partial x_1} & \frac{\partial f_y}{\partial x_1} \\ \frac{\partial f_x}{\partial x_2} & & & & & \\ \frac{\partial f_x}{\partial x_3} & & & & & \\ \frac{\partial f_x}{\partial x_4} & & & & & \\ \frac{\partial f_x}{\partial x_5} & & & & & \\ \frac{\partial f_x}{\partial x_6} & & & & & \end{bmatrix}$$

Jacobian

- Relates velocities in parameter space to velocities of outputs

$$\dot{Y} = J(X) \cdot \dot{X}$$

- If we know $Y_{current}$ and $Y_{desired}$, then we subtract to compute Y_{dot}
- Invert Jacobian and solve for X_{dot}

PLANNING

Kinematic Problems for Manipulation

- Reliably position the tip - go from one position to another position
- Don’t hit anything, avoid obstacles
- Make smooth motions
  - at reasonable speeds and
  - at reasonable accelerations
- Adjust to changing conditions -
  - i.e. when something is picked up *respond to the change in weight*

Planning Problem

- (Arm) Pose: Set of join values
- (Arm) Trajectory:
  - Given a start pose and an end pose
  - A list of intermediate poses
  - That should be reached one after the other
  - Often with associated (desired) velocities and accelerations
- Constrains:
  - Don’t collide with yourself
  - Don’t collide with anything else
  - Additional possible constrains:
    - Maximum joint velocities or accelerations
    - Keep global orientation of a joint (often end-effector) within certain boundaries

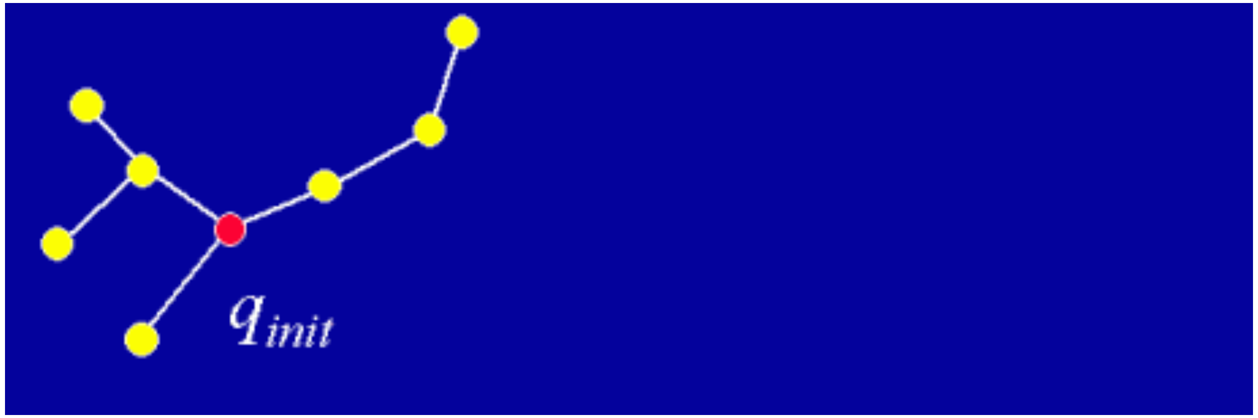
Planning Problem cont.

- Often the goal specified in Cartesian space (not joint space)
- => use IK to get joint space
- => often multiple (even infinitely many) solutions
    - Which one select for planning?
    - Plan for several solutions and select best!?

RRT

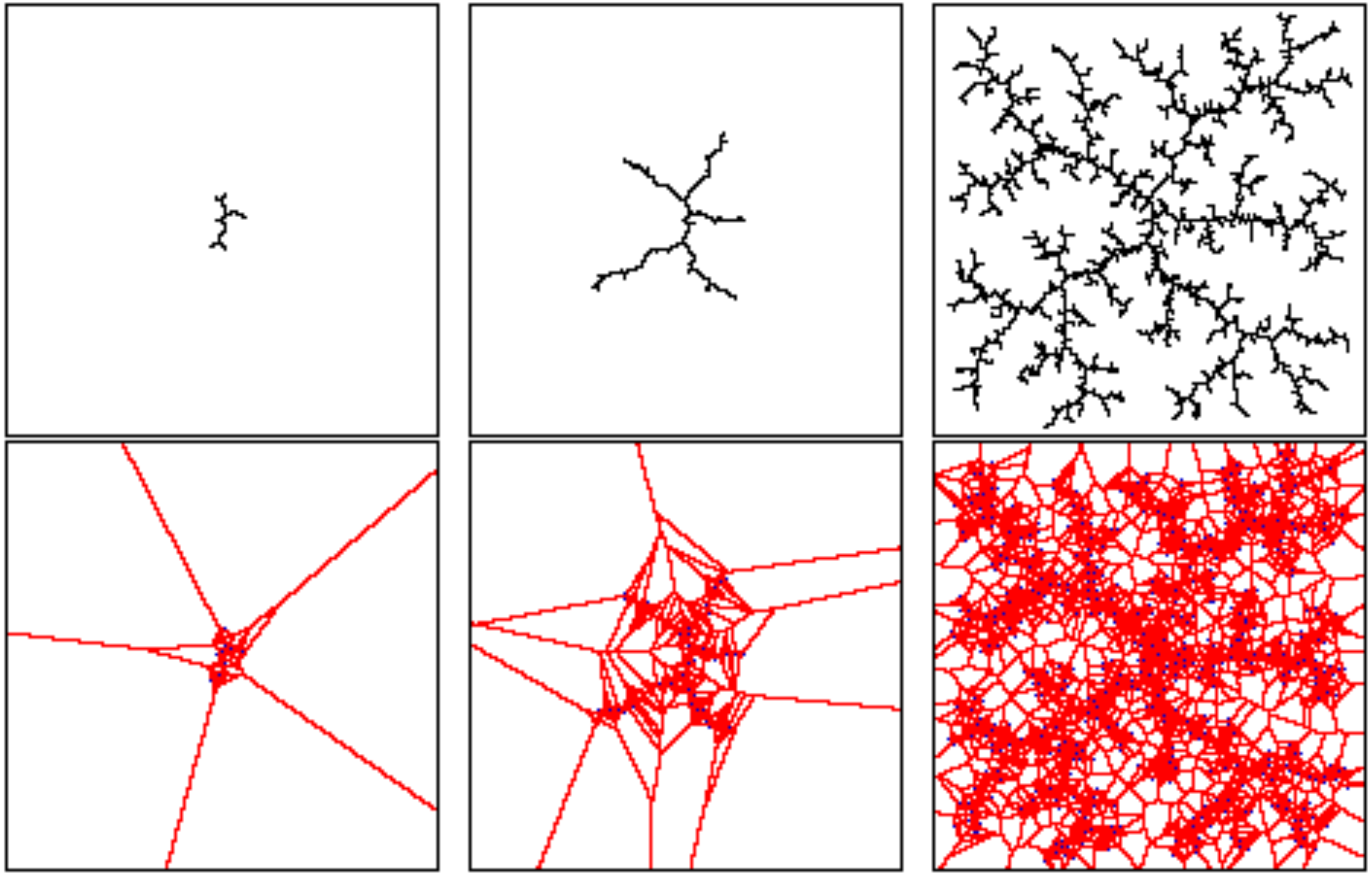
```
BUILD_RRT(q_init)
    T.init(q_init);
    for k = 1 to K do
        q_rand ← RANDOM_CONFIG();
        EXTEND(T, q_rand);
    Return T
```

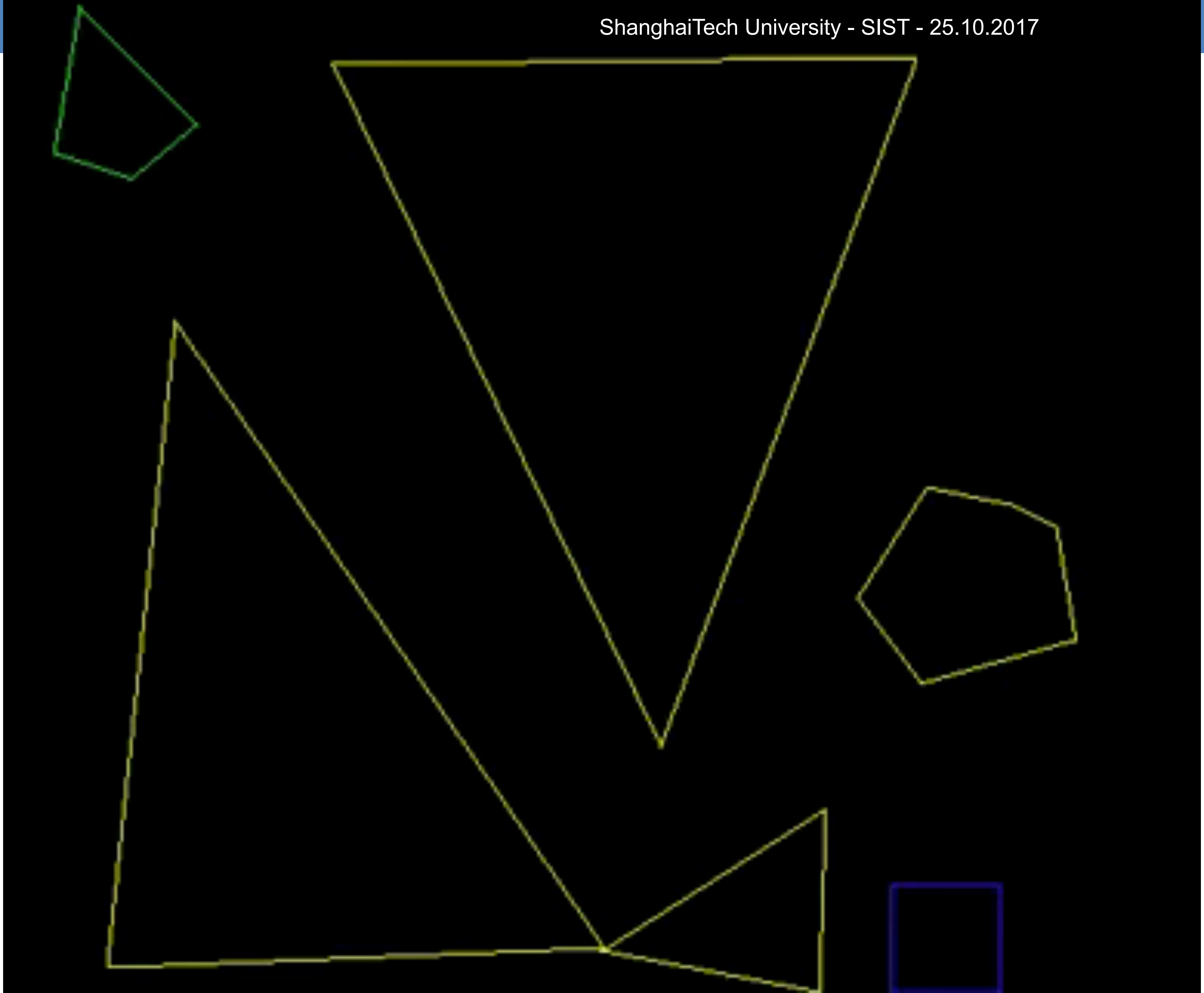
```
EXTEND(T, q)
    q_near ← NEAREST_NEIGHBOR(q, T);
    if NEW_CONFIG(q, q_near, q_new) then
        T.add_vertex(q_new);
        T.add_edge(q_near, q_new);
        if q_new = q then
            Return Reached;
        else
            Return Advanced;
    Return Trapped;
```

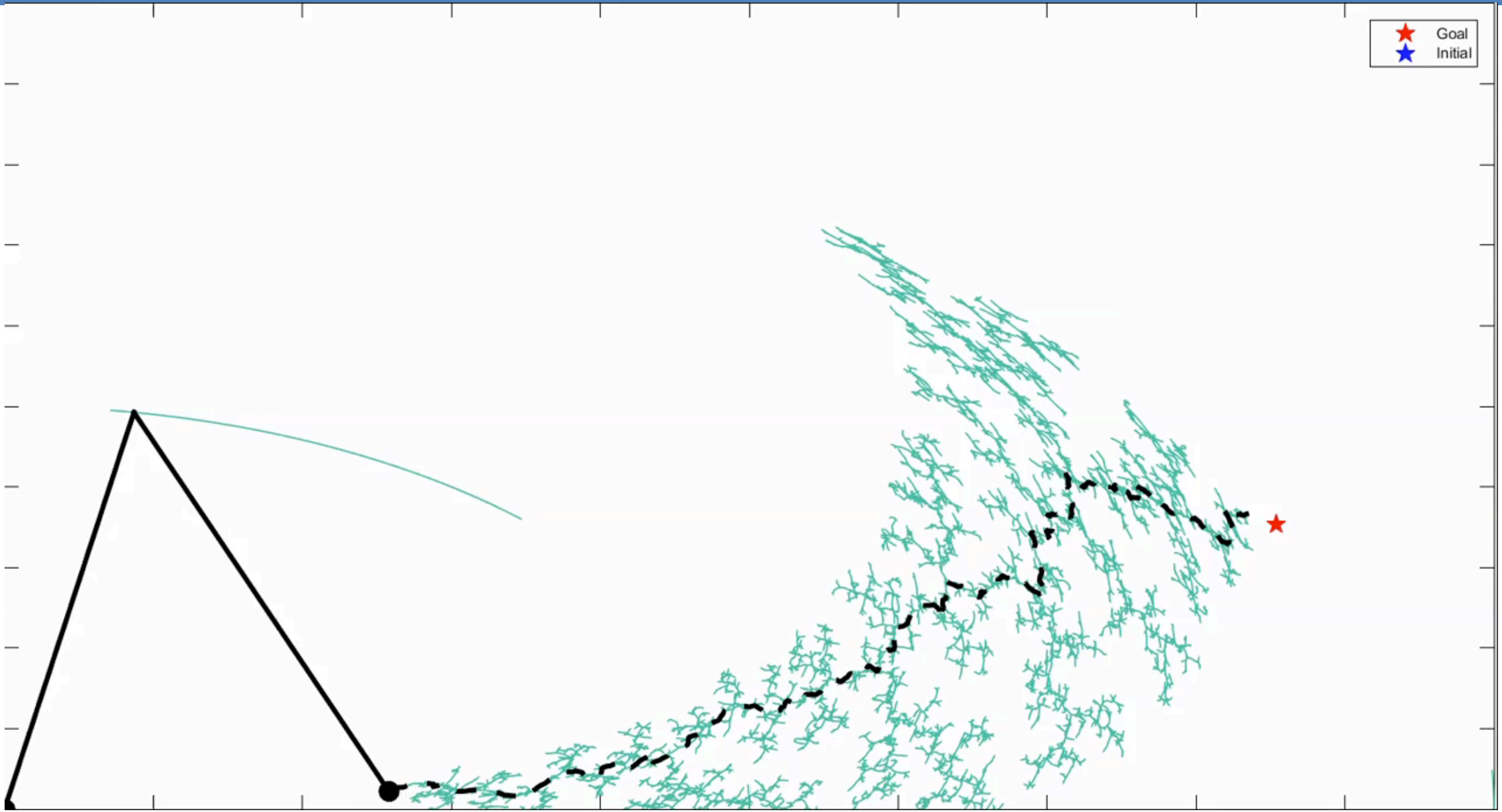


Why are RRT's rapidly exploring?

The probability of a node to be selected for expansion is proportional to the area of its Voronoi region



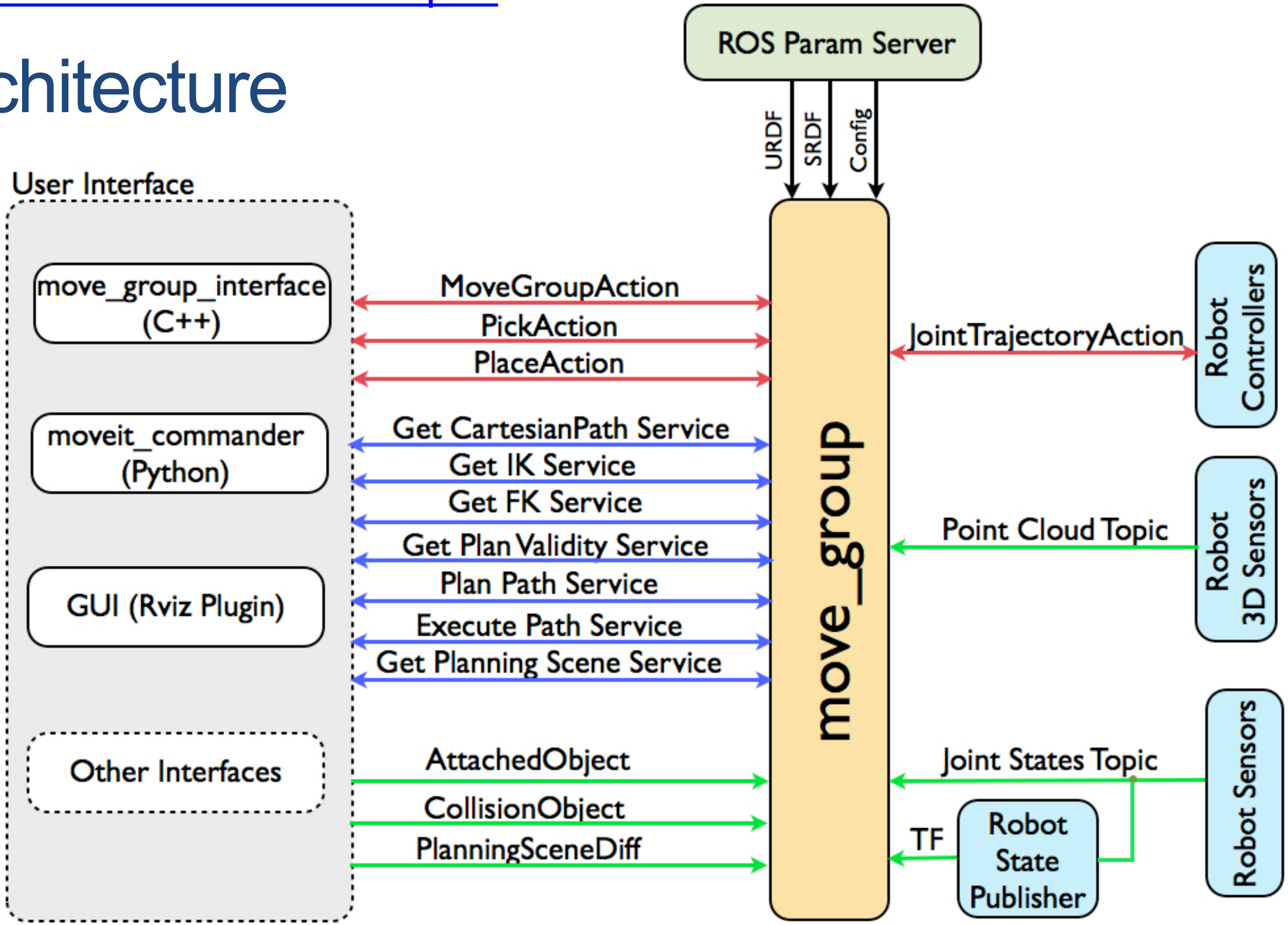



Goal
Initial

MOVEIT

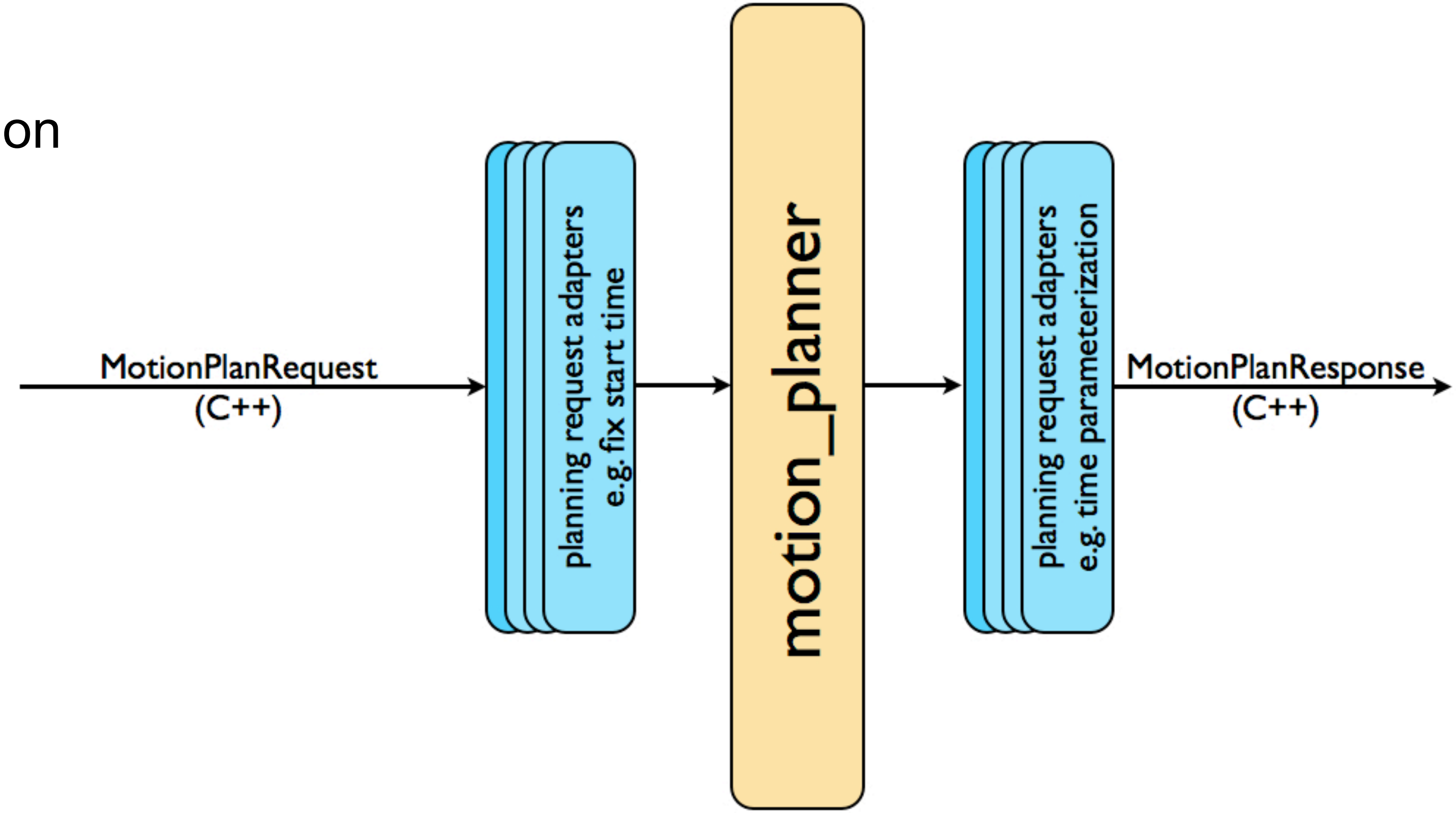
http://moveit.ros.org/documentation/concepts/

System Architecture

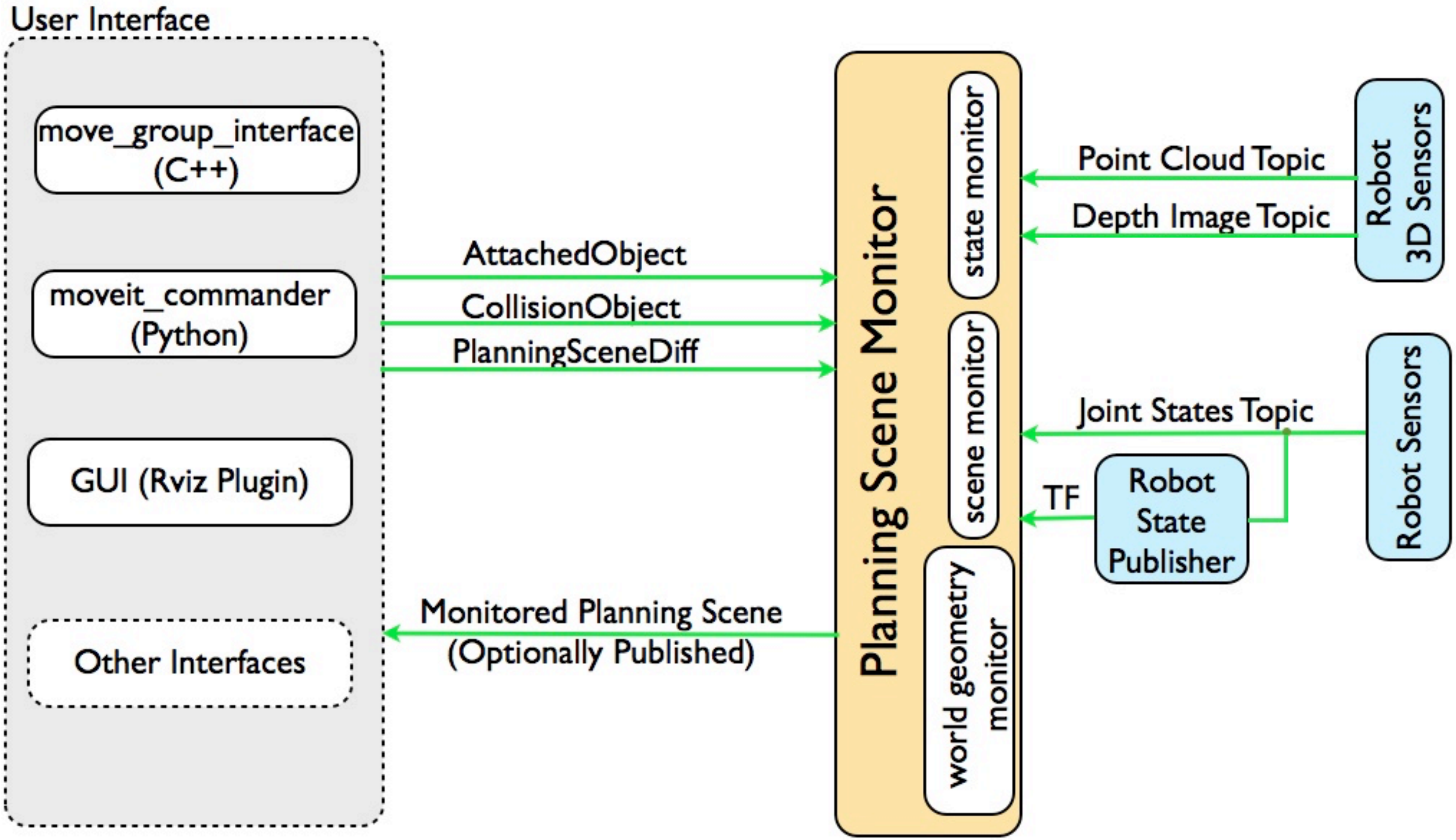


Motion Planning

- Mainly:
- OMPL (Open Motion Planning Library)

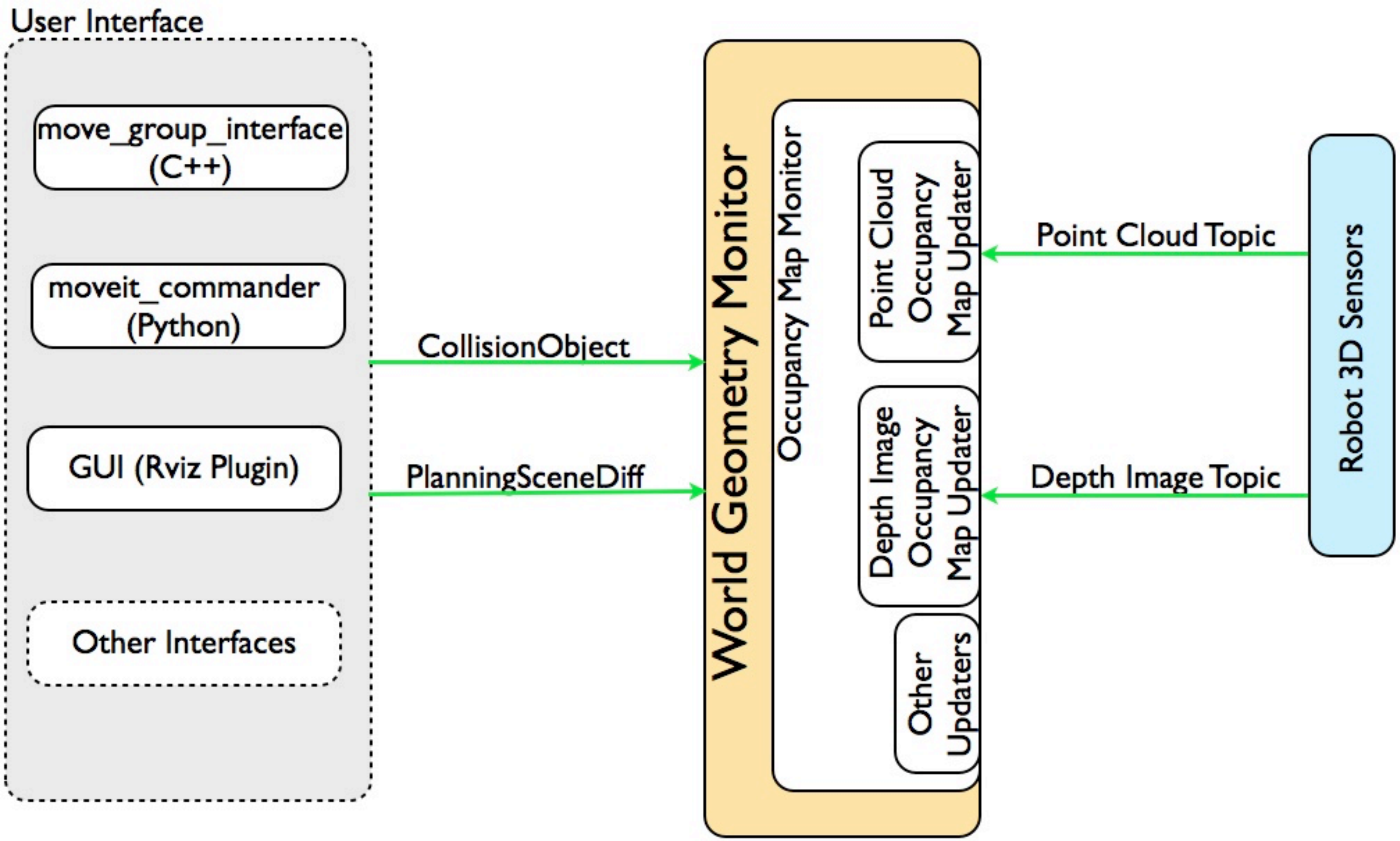


Planning Scene



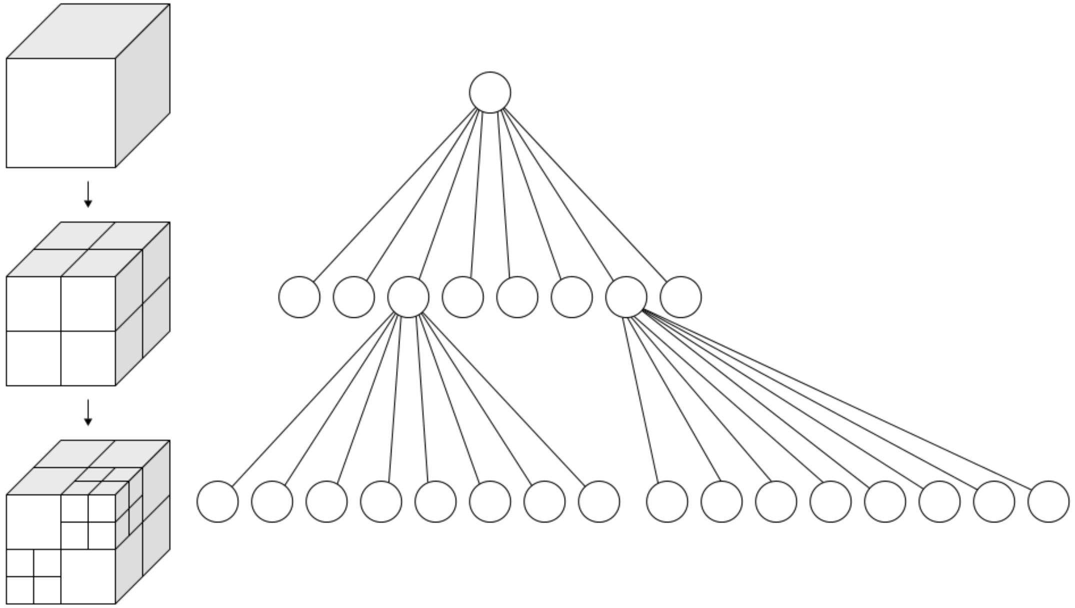
3D Perception

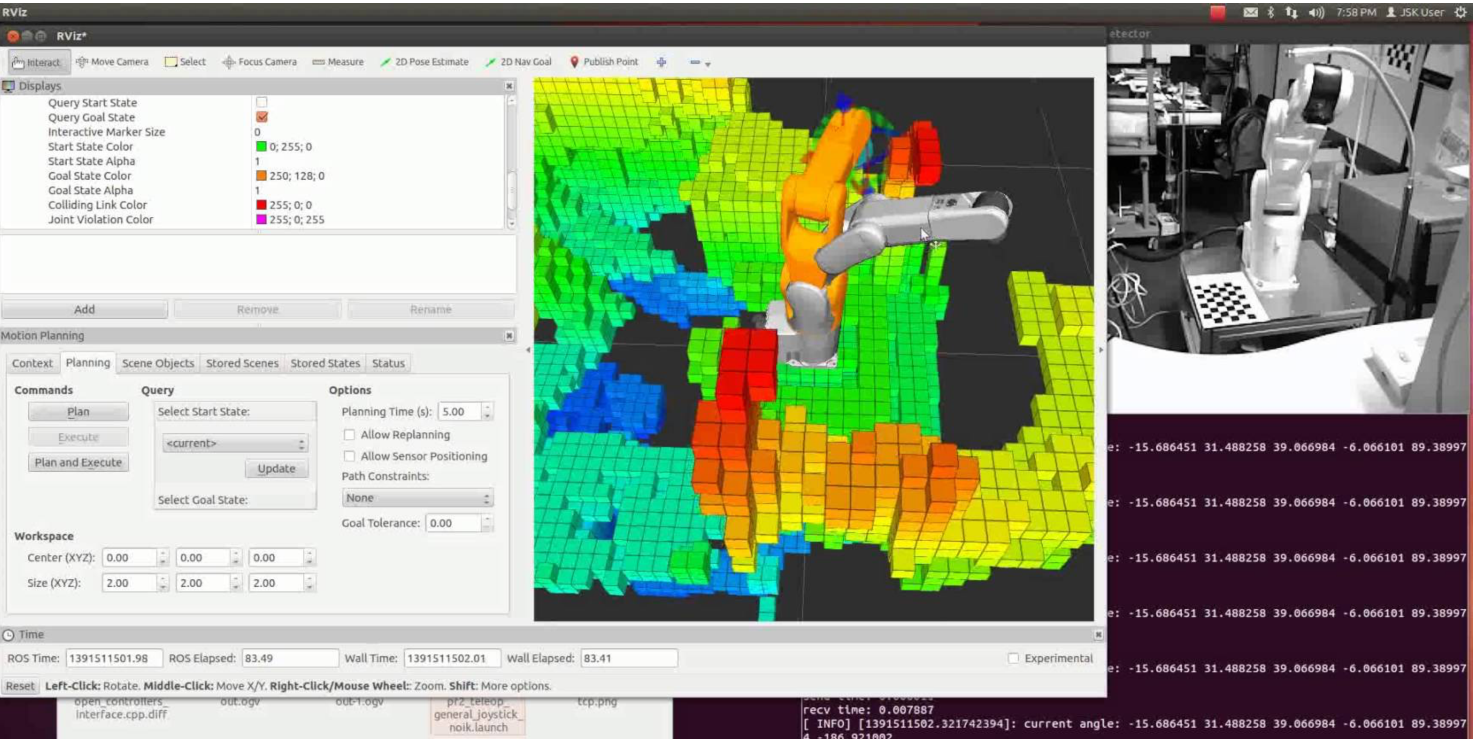
- Octomap



Octomap / Octree

- Depth sensor (usually Kinect)
- http://wiki.ros.org/octomap

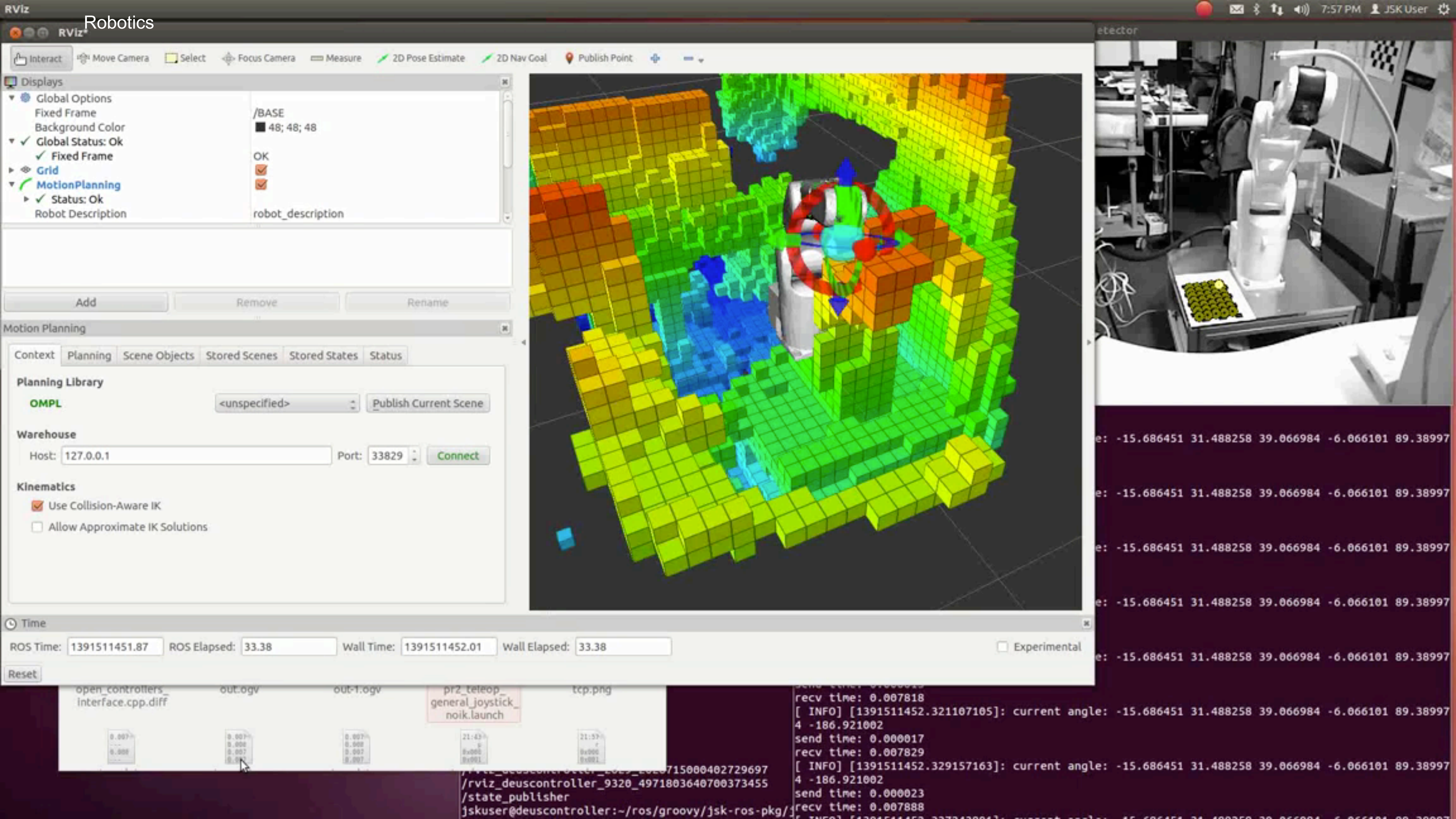




Grasp an Object: Steps

1. Startup robot and sensors
2. Detect object & its pose
3. Select grasping points on the object
4. Scan the scene and environment (for collision checking later)
5. Use IK to check if grasping point can be reached – checks for collisions – may try thousands of possibilities (before concluding that there is always a collision)
6. Use motion planning to plan from current pose to goal pose: Lots of collision checks! Might realize that it is impossible after a long time
7. Execute that trajectory: Check if we reached the intermediate pose (within the time constraint) and command the next
8. Controller: take dynamics into account to move to the next intermediate pose
9. Once goal is reached close fingers.
10. Check if object is in fingers
11. Add the object to the collision description of the robot
12. Plan the path to the goal pose…

Robotics


RViz
RViz*
Interact
Move Camera
Select
Focus Camera
Measure
2D Pose Estimate
2D Nav Goal
Publish Point
Displays
Global Options
Fixed Frame
/BASE
Background Color
48; 48; 48
Global Status: Ok
Fixed Frame
OK
Grid
MotionPlanning
Status: Ok
Robot Description
robot_description
Add
Remove
Rename
Motion Planning
Context
Planning
Scene Objects
Stored Scenes
Stored States
Status
Planning Library
OMPL
<unspecified>
Publish Current Scene
Warehouse
Host: 127.0.0.1
Port: 33829
Connect
Kinematics
Use Collision-Aware IK
Allow Approximate IK Solutions
Time
ROS Time: 1391511451.87
ROS Elapsed: 33.38
Wall Time: 1391511452.01
Wall Elapsed: 33.38
Experimental
Reset
7:57 PM
JSK User
recv time: 0.007818
[INFO] [1391511452.321107105]: current angle: -15.686451 31.488258 39.066984 -6.066101 89.38997
4 -186.921002
send time: 0.000017
recv time: 0.007829
[INFO] [1391511452.329157163]: current angle: -15.686451 31.488258 39.066984 -6.066101 89.38997
4 -186.921002
send time: 0.000023
recv time: 0.007888
/rviz_deuscontroller_9320_4971803640700373455
/state_publisher
jskuser@deuscontroller:~/ros/groovy/jsk-ros-pkg/
out.ogv
out-1.ogv
tcp.png

CONTROL

Finding the Dynamic Model of a Robotic System

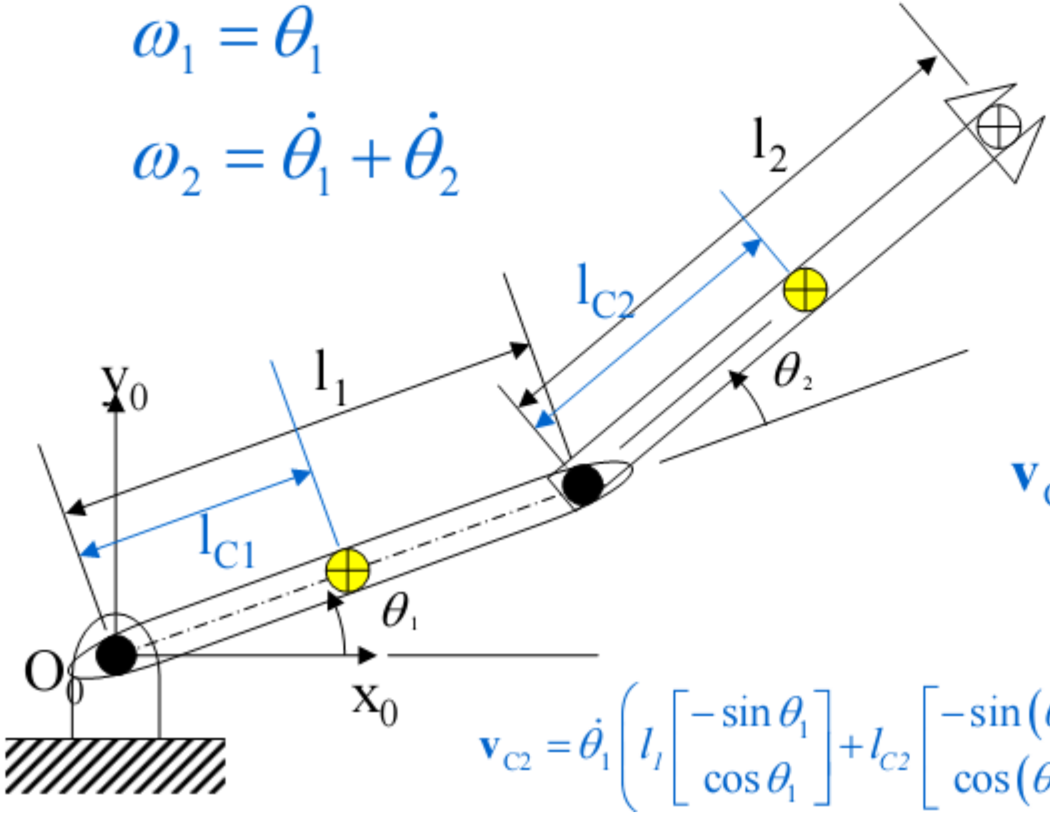
- Dynamics
- Lagrange Method
- Equations of Motion

Step 1: Identify Model Mechanics

Example: 2-link robotic arm

$$\omega_1 = \dot{\theta}_1$$

$$\omega_2 = \dot{\theta}_1 + \dot{\theta}_2$$



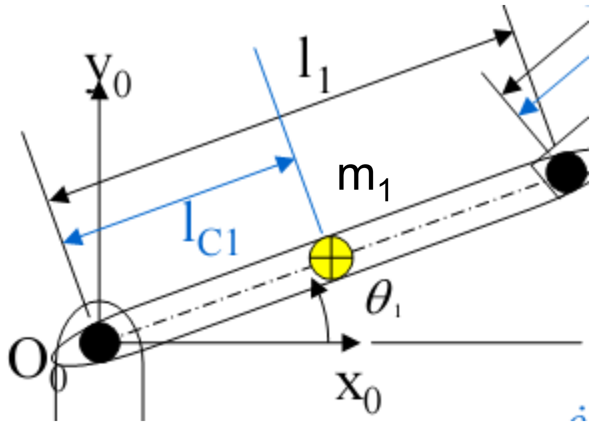
$$\mathbf{v}_{C1} = l_{C1}\dot{\theta}_1 \begin{bmatrix} -\sin\theta_1 \\ \cos\theta_1 \end{bmatrix}$$

$$\mathbf{v}_{C2} = \dot{\theta}_1 \left(l_1 \begin{bmatrix} -\sin\theta_1 \\ \cos\theta_1 \end{bmatrix} + l_{C2} \begin{bmatrix} -\sin(\theta_1+\theta_2) \\ \cos(\theta_1+\theta_2) \end{bmatrix} \right) + l_{C2}\dot{\theta}_2 \begin{bmatrix} -\sin(\theta_1+\theta_2) \\ \cos(\theta_1+\theta_2) \end{bmatrix}$$

Source: Peter R. Kraus, 2-link arm dynamics

Step 2: Identify Parameters

- For each link, find or calculate
  - Mass, m_i
  - Length, l_i
  - Center of gravity, l_{Ci}
  - Moment of Inertia, i_i



$i_1 = m_1 l_1^2 / 3$

Step 3: Formulate Lagrangian

- Lagrangian L defined as difference between kinetic and potential energy:

$$L(\mathbf{q}, \dot{\mathbf{q}}) = T - V$$

- L is a scalar function of q and dq/dt
- L requires only first derivatives in time

Kinetic and Potential Energies

- Kinetic energy of individual links in an *n*-link arm

$$T_i = \frac{1}{2} m_i \mathbf{v}_{\mathrm{Ci}}^{\mathrm{T}} \mathbf{v}_{\mathrm{Ci}} + \frac{1}{2} \boldsymbol{\omega}_{\mathrm{i}}^{\mathrm{T}} \mathbf{I}_{\mathrm{i}} \boldsymbol{\omega}_{\mathrm{i}} \qquad T = \sum_{i=1}^{n} T_i$$

- Potential energy of individual links

$$V_i = m_i l_{Ci} g \sin(\theta_i) h_{0i}$$

Height of link end

Energy Sums (2-Link Arm)

- T = sum of kinetic energies:

$$T = \frac{1}{2} m_1 \left| \mathbf{v}_{C1} \right|^2 + \frac{1}{2} m_2 \left| \mathbf{v}_{C2} \right|^2 + \frac{1}{2} I_1 \omega_1^2 + \frac{1}{2} I_2 \omega_2^2$$

$$= \frac{1}{2} m_1 l_{C1}^2 \dot{\theta}_1^2 + \frac{1}{2} m_2 (l_1^2 \dot{\theta}_1^2 + 2 l_1 l_{C2} \dot{\theta}_1 (\dot{\theta}_1 + \dot{\theta}_2) \cos\theta_2 + l_{C2}^2 (\dot{\theta}_1 + \dot{\theta}_2)^2)$$

$$+ \frac{1}{2} I_1 \dot{\theta}_1^2 + \frac{1}{2} I_2 \left(\dot{\theta}_1 + \dot{\theta}_2 \right)^2$$

- V = sum of potential energies:

$$V = m_1 g l_{C1} \sin\theta_1 + m_2 g \left(l_1 \sin\theta_1 + l_{C2} \sin(\theta_1 + \theta_2) \right)$$

Step 4: Equations of Motion

- Calculate partial derivatives of L wrt q_i, dq_i/dt and plug into general equation:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = \boxed{Q_i^{'}} \qquad (i = 1, 2, \ldots, n)$$

Non-conservative Forces (damping, inputs)

Inertia (d^2q_i/dt^2)

Conservative Forces

Equations of Motion – Structure

$$\mathbf{M}\ddot{\mathbf{q}} + \mathbf{c} + \mathbf{f}_g = \boldsymbol{\tau}$$

- M – Inertia Matrix
  - Positive Definite
  - Configuration dependent
  - Non-linear terms: sin(θ), cos(θ)
- C – Coriolis forces
  - Non-linear terms: sin(θ), cos(θ), $(d\theta/dt)^2$, $(d\theta/dt)*\theta$
- F_g – Gravitational forces
  - Non-linear terms: sin(θ), cos(θ)

Source: Peter R. Kraus, 2-link arm dynamics