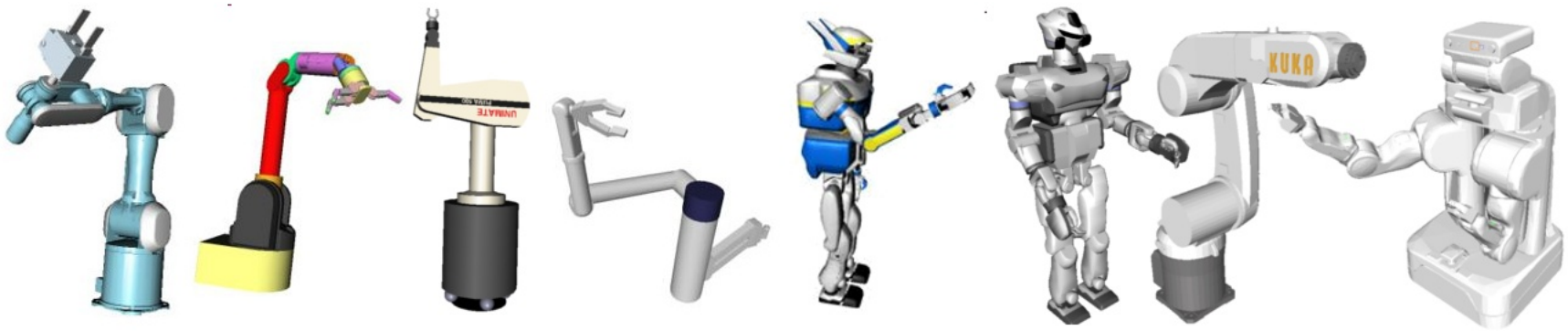
Inverse kinematics, Jacobians and Motion Control



Inverse Kinematics

- Given desired position/rotation of end tool, find angles/translations of each joint
- Given H = $\begin{pmatrix} R & d \\ 0 & 1 \end{pmatrix}$ and $T^0_n(q) = H$
- .. then ”solve” equations to find q
- 12 (nonlinear) equations with 4 unknowns (for crustcrawler with 4 joints..)

ROS – MoveIt! doc

Kinematics

The Kinematics Plugin

MoveIt! uses a plugin infrastructure, especially targeted towards allowing users to write their own inverse kinematics algorithms. Forward kinematics and finding jacobians is integrated within the RobotState class itself. The default inverse kinematics plugin for MoveIt! is configured using a numerical jacobian-based solver. This plugin is automatically configured by the MoveIt! Setup Assistant.

IKFast Plugin

Often, users may choose to implement their own kinematics solvers, e.g. the PR2 has its own kinematics solvers. A popular approach to implementing such a solver is using the IKFast package to generate the C++ code needed to work with your particular robot.

2D planar robot

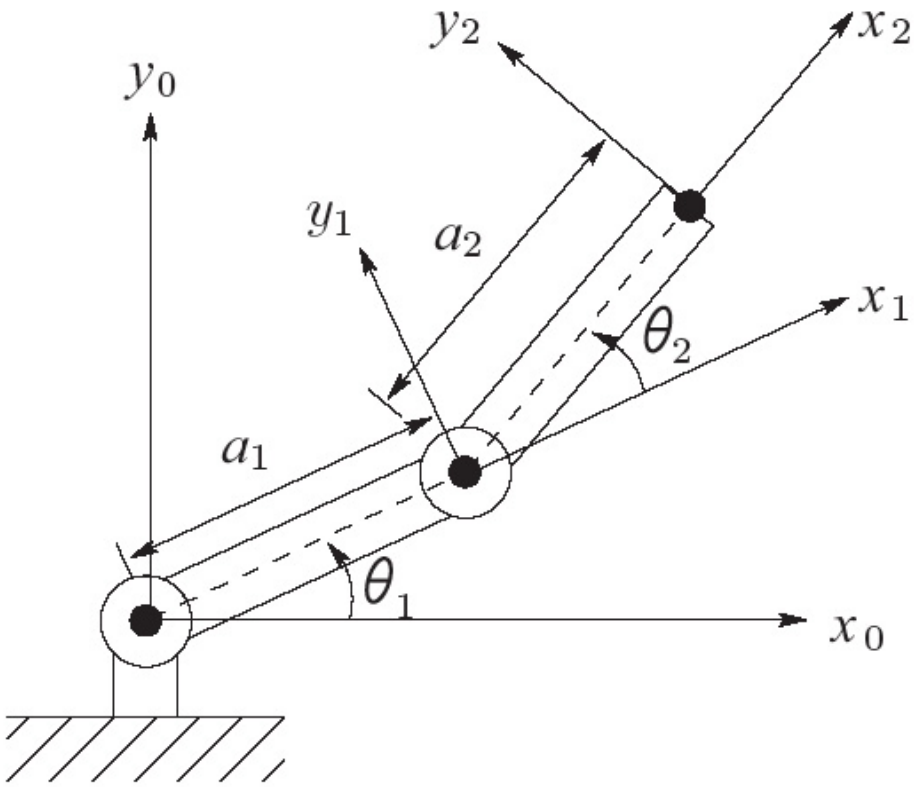
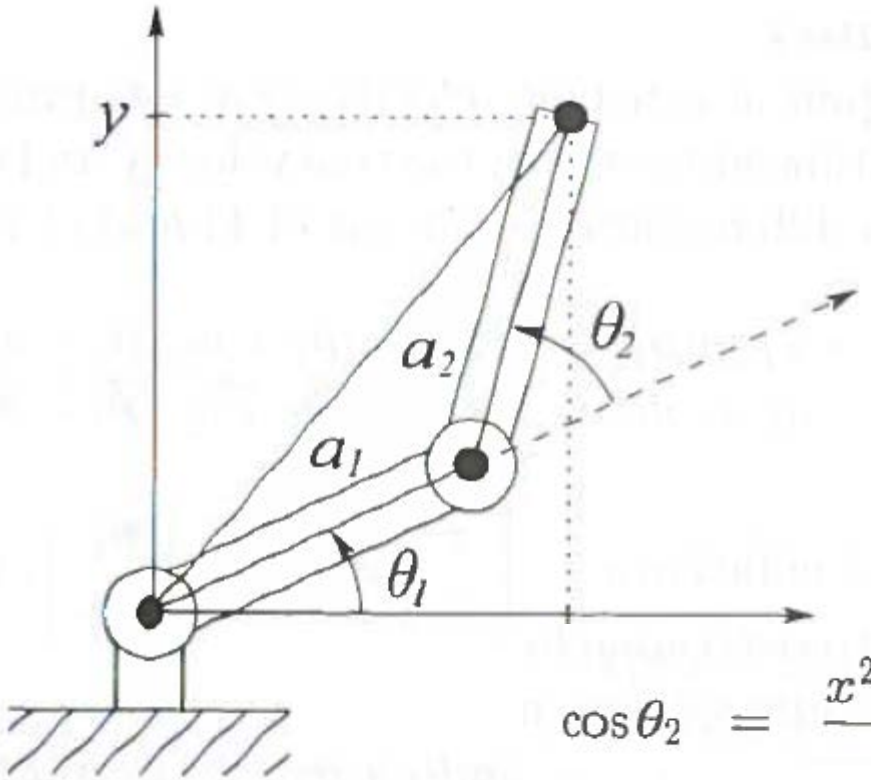


Figure 3.6: Two-link planar manipulator. The z-axes all point out of the page, and are not shown in the figure.

2D planar robot



$$\cos\theta_2 = \frac{x^2 + y^2 - a_1^2 - a_2^2}{2a_1 a_2} := D \tag{1.5}$$

$$\sin(\theta_2) = \pm\sqrt{1 - D^2} \tag{1.6}$$

$$\theta_2 = \tan^{-1}\frac{\pm\sqrt{1 - D^2}}{D} \tag{1.7}$$

$$\theta_1 = \tan^{-1}(y/x) - \tan^{-1}\left(\frac{a_2 \sin\theta_2}{a_1 + a_2\cos\theta_2}\right) \tag{1.8}$$

Geometric approach – CrustCrawler Position

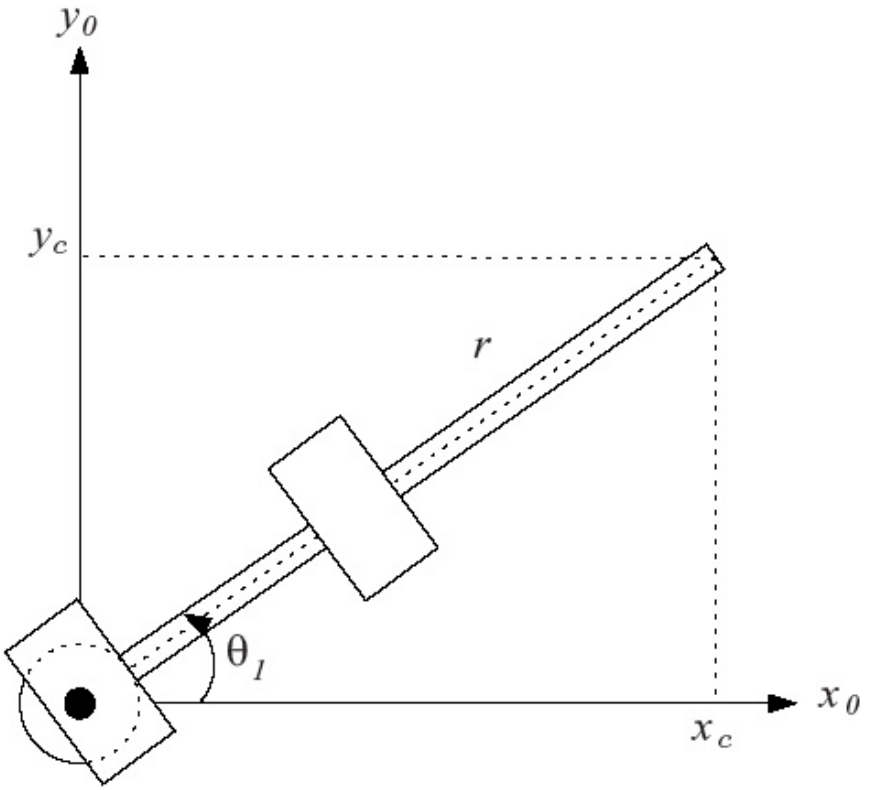


Figure 3.14: Projection of the wrist center onto $x_0 - y_0$ plane.

Geometric approach - CrustCrawler Position

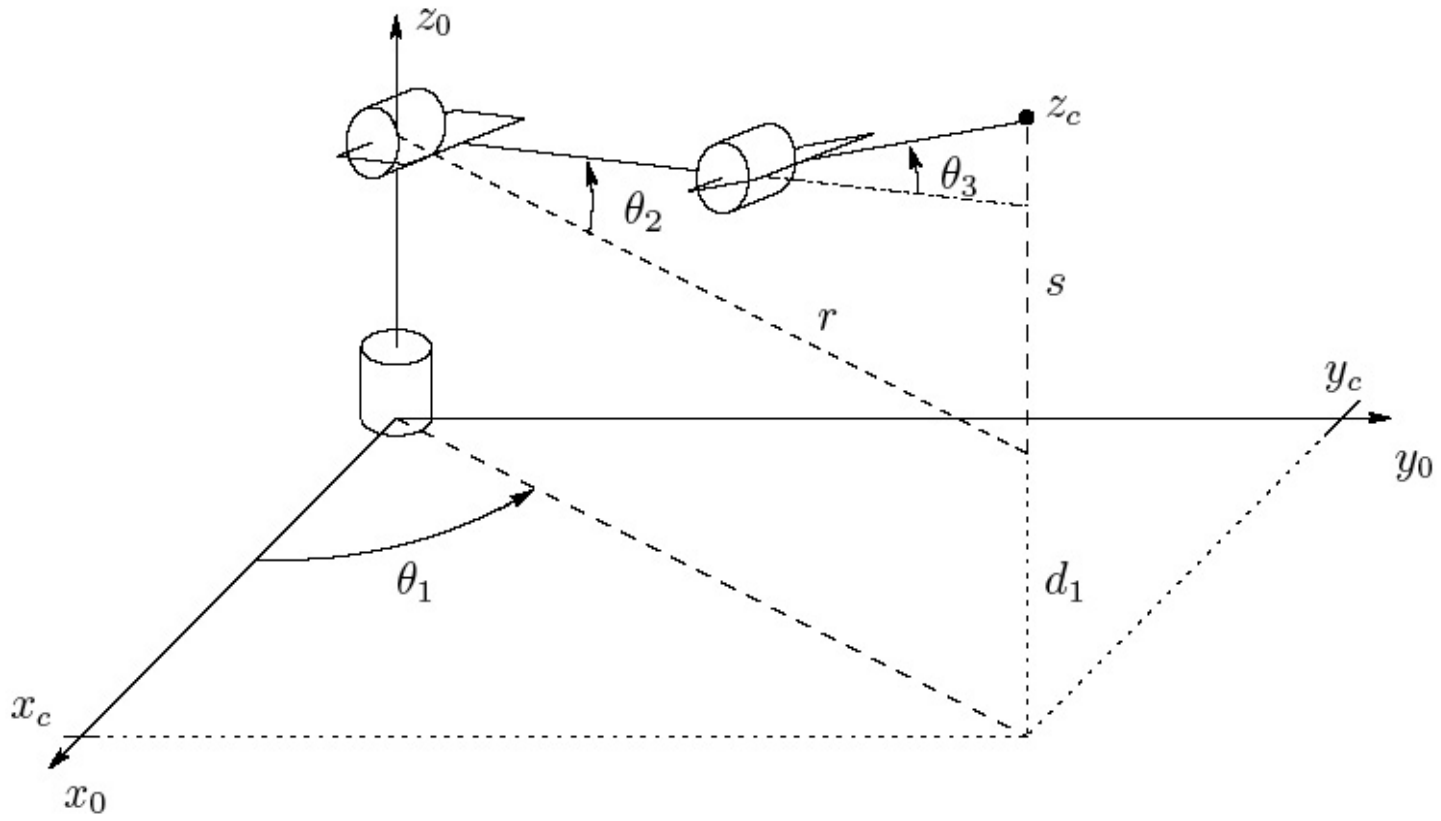


Figure 3.13: Elbow manipulator.

Geometric approach - orientation

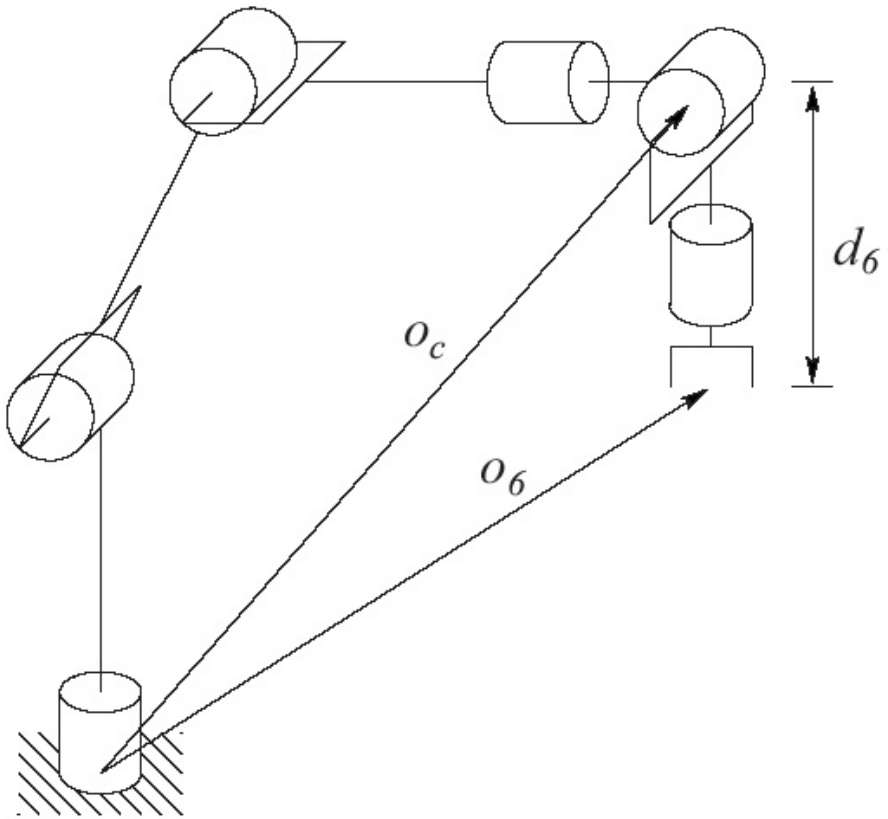


Figure 3.12: Kinematic decoupling.

Manipulator Jacobian

- Relation between (spatial) velocity of end frame and velocity of each joint

$$\begin{pmatrix} v \\ \omega \end{pmatrix} = \begin{pmatrix} J_v \\ J_\omega \end{pmatrix} \dot{q} = J\dot{q}$$

- Rewritten :

$$v = J_v \dot{q} \quad \text{and} \quad \omega = J_\omega \dot{q}$$

  - v = velocity of end frame
  - ω = angular velocity of end frame
  - J = manipulator Jacobian

Manipulator Jacobian
2D Planar Robot

$$x = a_1 \cos\theta_1 + a_2 \cos(\theta_1 + \theta_2) \quad (1.1)$$

$$y = a_1 \sin\theta_1 + a_2 \sin(\theta_1 + \theta_2) \quad (1.2)$$

$$\begin{aligned} \dot{x} &= -a_1 \sin\theta_1 \cdot \dot{\theta}_1 - a_2 \sin(\theta_1 + \theta_2)(\dot{\theta}_1 + \dot{\theta}_2) \\ \dot{y} &= a_1 \cos\theta_1 \cdot \dot{\theta}_1 + a_2 \cos(\theta_1 + \theta_2)(\dot{\theta}_1 + \dot{\theta}_2) \end{aligned} \quad (1.9)$$

Using the vector notation $x = \begin{bmatrix} x \\ y \end{bmatrix}$ and $\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$

$$\begin{aligned} \dot{x} &= \begin{bmatrix} -a_1 \sin\theta_1 - a_2 \sin(\theta_1 + \theta_2) & -a_2 \sin(\theta_1 + \theta_2) \\ a_1 \cos\theta_1 + a_2 \cos(\theta_1 + \theta_2) & a_2 \cos(\theta_1 + \theta_2) \end{bmatrix} \dot{\theta} \quad (1.10) \\ &= J\dot{\theta} \end{aligned}$$

Avoiding singularities

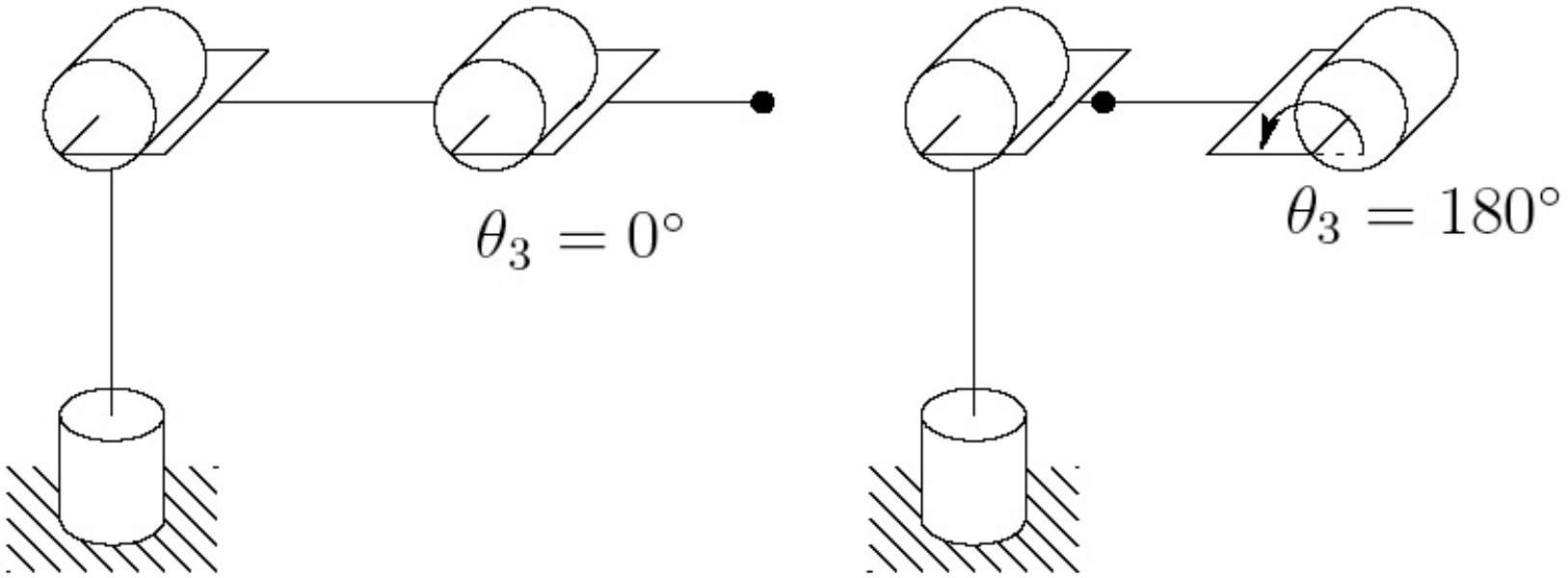


Figure 4.6: Elbow singularities of the elbow manipulator.

Manipulability

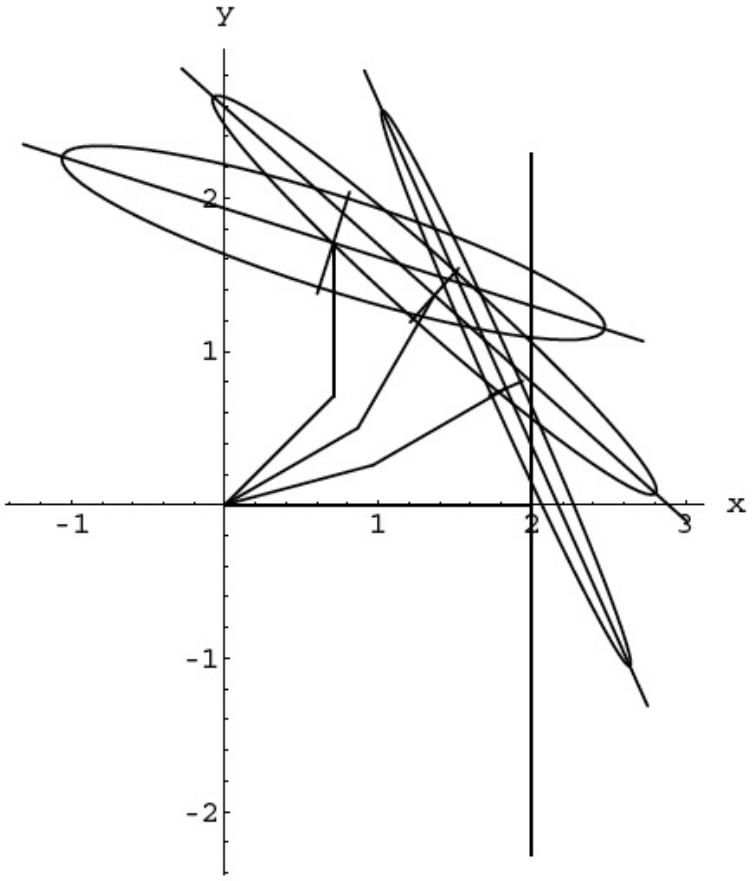


Figure 4.12: Manipulability ellipsoids are shown for several configurations of the two-link arm.

Resolved-rate motion control

- Uses Jacobian
- $q(t+1) = q(t) + J^{-1}\begin{pmatrix} v \\ \omega \end{pmatrix}\Delta t$

Final Project

Your Tasks..

- Matlab exercises – crustcrawler inverse kinematics
- Mandatory exercise - Inverse kinematics in ROS / Python
- Mandatory Final project proposal