1 Introduction

This lecture is to focus on Motion Planning. The lecture starts with introduction of Forward and Inverse kinematics, and then illustrates how to use Newton Method to solve Inverse Kinematics. The second part of the lecture is to concentrate on the algorithms of Motion Planning.

2 Forward Kinematics and Inverse Kinematics

2.1 Definitions

Forward kinematics (FK) refers to the use of the kinematic equations to compute the position of the end-effector from specified values for the joint parameters.

Inverse Kinematics (IK) is the use of the kinematics equations to determine the joint parameters that provide a desired position of the end-effector.

2.2 Example to Illustrate Forward and Inverse Kinematics

\[ FK(q) = X_g \]

(2)
$q = \begin{pmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{pmatrix} \in C$ \hfill (1)

The function $FK(q)$ takes in one particular configuration of the arm and produces the output $x, y$ where the end effector is to be positioned.

Using Figure 1, we can mathematically derive the equations for the Forward Kinematics as follows

\begin{align*}
x &= l_1 \cdot \cos \theta_1 + l_2 \cos(\theta_1 + \theta_2) + l_3 \cdot \cos(\theta_1 + \theta_2 + \theta_3) \hfill (3) \\
y &= l_1 \cdot \sin \theta_1 + l_2 \sin(\theta_1 + \theta_2) + l_3 \cdot \sin(\theta_1 + \theta_2 + \theta_3) \hfill (4)
\end{align*}

Forward Kinematics is an injective function that is a one to one mapping where for every position of arm then there is a singular corresponding position of the end effector.

For Inverse Kinematics, $X_g$ is the position goal that end-effector should reach, and $q_s$ is the initial point where end-effector is.

$Q_g = \{q \in C | FK(q) = X_g \}$ \hfill (5)

The redundant manipulator is one in which $N > M$ implying that there are more parameters we can control than the number of target values desired. This provides more freedom to create a better posture or avoid obstacles.

### 2.3 Method of solving IK

#### 2.3.1 Newton’s Method

Use same example in part 2.

In order to get joint configuration, we try to solve equation

$$\frac{FK(q) - X_G}{f(x)} = 0$$ \hfill (6)
The general solution for this problem is

\[ f(q) \approx f(q_0) + \left. \frac{df}{dq} \right|_{q = q_0} (q - q_0) \]  

(7)

The \( \frac{df}{dq} \) is consider to be Jacobian Matrix.

The Jacobian matrix is the matrix of all first-order partial derivatives of a vector or scalar-valued function with respect to another vector. In the context of manipulators, the Jacobian is defined as the change in linear velocity of the end effector when angular velocity of one joint angle is one radian per second. The Jacobian matrix acts as a transform from the configuration space to the task space.

Jacobian Matrix in this case is

\[
J = \begin{bmatrix}
\frac{\partial x}{\partial \theta_1} & \frac{\partial x}{\partial \theta_2} \\
\frac{\partial y}{\partial \theta_1} & \frac{\partial y}{\partial \theta_2}
\end{bmatrix}
\]  

(8)

The dimension of the Jacobian Matrix is \( 2 \times 3 \) which implies that this is redundant manipulator.

2.3.2 Example of using Newton method

\[ q = \begin{pmatrix} \theta_1 \\ \theta_2 \end{pmatrix} \in C \]  

(9)

\[ FK(q) = FK \begin{pmatrix} \theta_1 \\ \theta_2 \end{pmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \cos(\theta_1) + \cos(\theta_1 + \theta_2) \\ \sin(\theta_1) + \sin(\theta_1 + \theta_2) \end{bmatrix} \]  

(10)

\[ \frac{df}{dq} = \begin{bmatrix}
\frac{\partial x}{\partial \theta_1} & \frac{\partial x}{\partial \theta_2} \\
\frac{\partial y}{\partial \theta_1} & \frac{\partial y}{\partial \theta_2}
\end{bmatrix} = \begin{bmatrix}
-sin\theta_1 - sin(\theta_1 + \theta_2) & -sin(\theta_1 + \theta_2) \\
\cos \theta_1 + \cos(\theta_1 + \theta_2) & \cos(\theta_1 + \theta_2)
\end{bmatrix} \]  

(11)

Examples of calculating Jacobian matrix:

\[ \theta = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad J = \begin{bmatrix} 0 & 0 \\ 2 & 1 \end{bmatrix} \]  

(12)

\[ \theta = \begin{pmatrix} \pi/2 \\ \pi/2 \end{pmatrix} \quad J = \begin{bmatrix} -1 & 0 \\ -1 & -1 \end{bmatrix} \]  

(13)
Each column of Jacobian Matrix represents velocity of target for unit velocity of joint.
If assume the initial joint configuration is \((0; \pi/2)\), and the length of link 0 and link 1 are both 1. Therefore
the we can set up equation and solve it by using Newton Method

\[
f(g) = f(g_0) + \frac{df}{dq} \cdot (q - q_0)
\]

(15)

\[
0 = \left[ \begin{pmatrix} 1 \\ 1 \end{pmatrix} - X_g \right] + \begin{pmatrix} -1 & -1 \\ 1 & 0 \end{pmatrix} \cdot \left[ q - \begin{pmatrix} 0 \\ \pi/2 \end{pmatrix} \right]
\]

(16)

### 2.3.3 Pseudoinverse Jacobian

Generally, the Pseudoinverse is a generalization of the inverse of a matrix.
The Moore-Penrose pseudoinverse is most widely used, and in relation to the Jacobian matrix it describes
\(J^+\), the inverse of the Jacobian which finds the least squares solution to the goal position according to equation

\[
q^* = q_0 - J^+(x_g - x_0)
\]

(17)

**Motivation** When presented with a redundant manipulator, you are usually put in a situation where an
infinite number of joint angles could solve the IK problem. Mathematically there are endless possibilities for
moving the robot to the goal, but in the real world we care only for a single solution so we can command our
robot accordingly. The pseudoinverse of the Jacobian can be used to find a single IK solution that minimizes
overall joint motion. The Jacobian is not readily invertible because it is not square (more columns than rows
in the redundant case), but the pseudoinverse can be found via SVD. This may not always be the “best”
solution, with regards to other metrics, but is often useful.

**Example** Take a two link manipulator with a goal of reaching \(x = 1.5\) and no restrictions on \(y\), as seen in
Figure 3. We have 2 DOFs but only 1 goal variable, so our manipulator is redundant. The set of possible
solutions may look like Figure 4, where there are an infinite number of \(\theta_1\) and \(\theta_2\) positions lying on a line
that will solve the problem. At point 1 and point 2, you can reach the goal using only one joint, but at point
3 you minimize the distance to the origin. $\theta_1^2 + \theta_2^2$ is minimized, and in theory the energy used to get to the goal is also minimized. This solution can be found with the pseudoinverse.

Performing SVD on a matrix returns $U$, $S$, and $V^*$. The pseudoinverse can be found by the equation: $A^+ = VS^+U^*$, where $S$ is diagonal and so $S^+$ can be found by taking the reciprocal of each element of $S$.

2.3.4 Null Space

In the general case, the Null Space of a linear mapping $A$ between two vector spaces is the set of elements $x$ such that $Ax = 0$.

Specifically, in our robot arm joint example the null space is the set of internal motions that keep the end effector fixed, or otherwise result in no progress towards the goal.

While the jacobian described one solution of a redundant manipulator, the null space $N$ can be used to describe the all solutions, as in equation

$$q_{all} = q^* + N\lambda$$

(18)

The null space can also describe amount of freedom a manipulator has. For instance, if a 3 link arm is to reach a goal in $\mathbb{R}^2$, 3 independent, unique DOFs - 2 goal variables = 1 Null Space.

3 Motion planning

How can we get from our starting position to any goal position? For a simple motion planning algorithm to be considered “Complete”, we follow these steps:

Make it to a goal position while satisfying the constraints in finite time Return failure if no path exists in finite time Return success and give path if one exists Resolution complete

There are other notions of completeness, such as Resolution Complete (discretize world and get close to goal within some resolution) and Probabilistically Complete (as time approaches infinity, we asymptotically approach complete solution).

3.1 Algorithm 1

Say we are given the configuration space shown on Figure 5

![Figure 5: Configuration space with an obstacle](image)

The simplest possible algorithm would be to try connecting straight lines from the start to the goal:

1. Generate an IK solution
2. Connect start and go by straight line
3. Collision check path if fails
4. A failure, then repeat from step 1
3.2 Improvement

We can use cleverness with our IK to improve our solution a bit. That is, we ought to try distant IKs if we encounter a failure. This is motivated by the idea that “Obstacles are close to other obstacles,” so the probability that the point very close to an obstacle is another obstacle is relatively high.

An effective method for parsing the space in such a way as to select distant IKs is to use the Bisection Method. We can recursively test the midpoints of gap samples, throwing away obstacles, until a desired resolution is reached.