What is inverse kinematics (IK)?

Inverse kinematics is the mathematical process that takes a desired position and orientation for a robot's end-effector and calculates the joint angles needed to achieve it.

This is necessary for multi joint robotic systems. It can be thought of as end-effector movement since we want, for example, the robot's hand at some point in space, and need to calculate what its shoulder, elbow and wrist joints should do in order to accomplish this.

You may be thinking "*well if inverse kinematics exist what is it the other way around?*" – and this is called *forward* kinematics (FK). It's where you have all the joint positions and what to figure out from there where the end-effector, or in our running example, hand, is.

Some complications:

- There are often many solutions to reach the same spot but not always in the best way
- Sometimes there is no solution if the desired placement is out of range
- IK needs to respect physical limits like servo or stepper limits and obstacle avoidance

There are two main approaches to solving IK.

The analytical approach uses trigonometry and geometry to derive a closed-form equation. This is typically fast and accurate for simple robot systems

The numerical approach uses iterative methods like the Jacobian Inverse, Gradient Descent or Pseudo Inverse. Using these methods may yield results with significantly more complex robots and constraints but has its limitations. Computationally it may be much more demanding, you may have convergence issues, and can expect a generally slower response time. This method involves some very spooky linear algebra.

Beyond this simple explanation, the math behind IK will not be significantly covered within the scope of these notes, and the LAIKA program project

****

History

IK has its roots in robotics, biomechanics and computer graphics. Denavit-Hartenberg parameters standardized how to model joints in the 80s, Analytical IK expanded in the 90s with robots like the Stanford arm, and in the 2000's boston dynamics began development of their robotic systems well known today. Libraries like KDL, Trac-IK MoveIt! Snd BioIK were developed in recent years with ROS in mind

ROS2 and IK/FK

Inverse kinematics tells the robot how to move its joints to reach a desired position, and ROS provides the framework and tools to implement that movement in real-time.

Concept	Role
Inverse Kinematics (IK)	Math used to calculate **joint angles** needed to reach a **desired position or orientation** (x, y, z, roll, pitch, yaw)
ROS	The **middleware** that lets you send those joint angles to the robot's controllers, simulate motion, listen to sensor feedback, etc.

A high level abstraction of workflow on LAIKA:

1. A target pose is defined in 3D space as a vector eg. "move to <0.4, 0.1, 0.2>"
2. IK Solver (MoveIt!) computes the required joint angles
3. ROS publishes those solved angles to the robots controllers via topics or services
4. The robot moves and a feedback loop confirms the pose

LAIKA has 3 degrees of freedom per leg, and 12 degrees of freedom total. Using IK, LAIKA can walk in a similar motion to real steps of a dog, and can adapt to uneven terrain. Using the IMU onboard the lower computer, positional estimates can be calculated, and foot placment can be understood by the system through forward kinematics, and thus be adjusted via the inverse.

LAIKA's body is based on a widely used quadruped frame in the quadruped community, and its limb dimensions and frame constraints are programmed in the IK/FK algorithm of the lower computer.

To bypass the joint limits enforced by the IK algorithm, a program may be written to control servo joints directly via PWM, thus setting a holdable position. This approach is taken to set and hold a pre-determined "prone" position that holds LAIKA's limbs in place during flight up until the deployment is verified on the ground. The exact join positions of "Prone" and the command for the actition are a sub-function of the LAIKALIB library, and is demonstrated in a later section of these notes.