Kinematic Control

Topics

- Kinematic Control
  - Setpoint Control
  - Trajectory Following
  - Open-loop vs Closed-loop

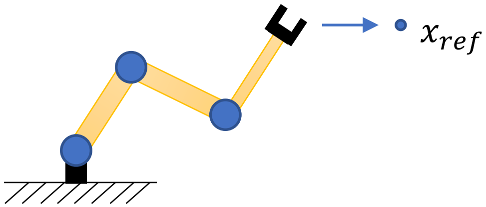
Complementary Reading: J.J. Craig, Introduction to Robotics, Chapter 5,9.

Setpoint vs Trajectory Control

Control can be defined as problem of regulating a setpoint (**setpoint control)** or following a trajectory **(trajectory control).**

- **Setpoint control**

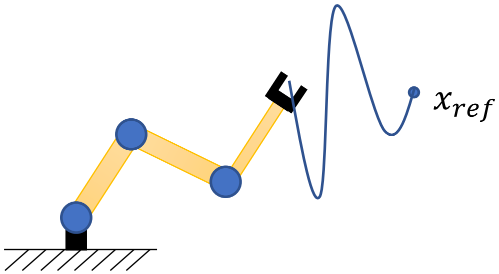
$$x_{ref} = \text{constant}$$



The input to the controller is the end-goal (*setpoint*). A vector is drawn from current position to the setpoint, and the robot moves as quickly as possible to close that gap to the setpoint.

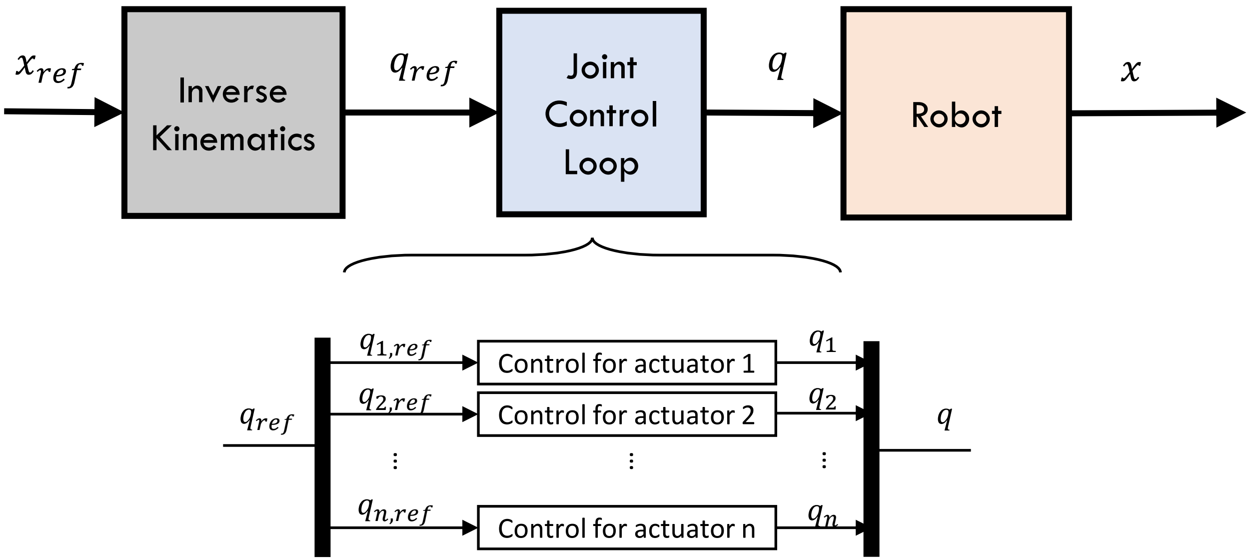
- **Trajectory control**

$$x_{ref}(t) = f(x_{end}, x_0, t)$$



The input to the controller is a *trajectory* described either as a continuous function (rare) or sampled path (typical) that connects the start to the goal.

Local Control: *Open-Loop Setpoint* Control



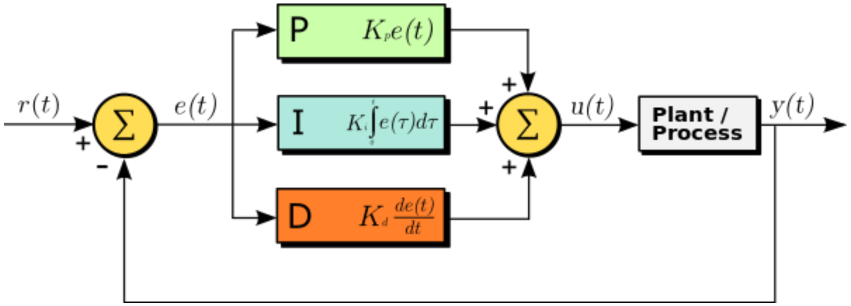
High level premise:

Assume you have an accurate model of your robot $x = f(q)$ such that by setting q you can expect x to match nearly-perfectly to real life.

Therefore, given x_{ref} to track,

1. Use inverse kinematics on x_{des} to find q_{des}
2. Move joint i to q_i $\forall i$ under some single-joint-angle control loop.

Proportional – Integral – Derivative (PID) Controller



- Used in almost all actuator and local control applications.

Continuous: $u(t) = K_p e(t) + K_i \int^t e(t)dt + K_d \dot{e}(t)$

Discrete: $u[k] = K_p \mathrm{e}[k] + K_i \sum_{i=0}^{k} \mathrm{e}[i] + K_d(\mathrm{e}[k] - \mathrm{e}[k-1])$

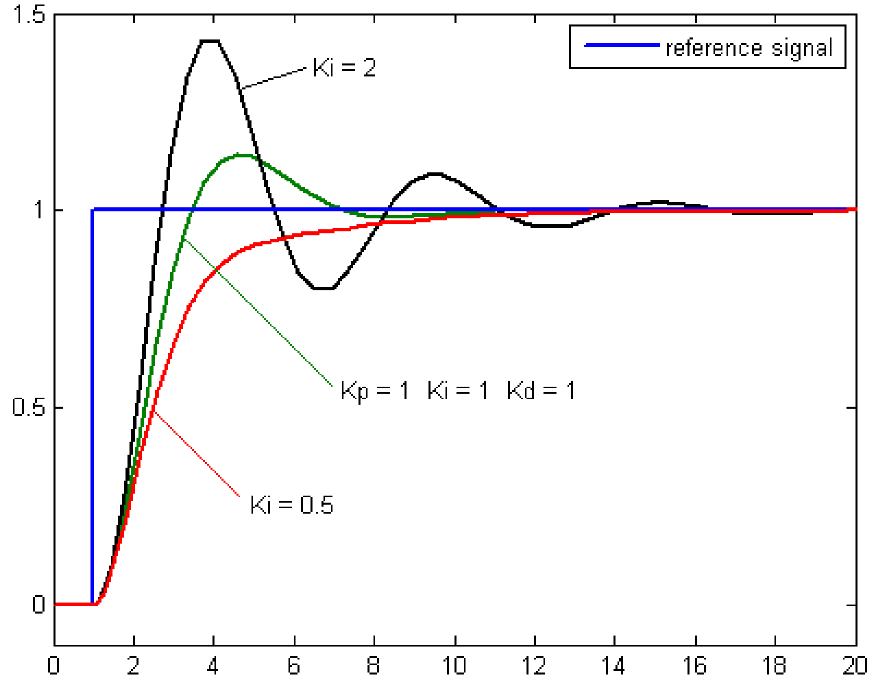
where $\mathrm{e}[k] = q_{ref}[k] - q[k]$

  - K_p, K_i, K_d are constants
  - u typically input to actuator (i.e. current or voltage)

- Can apply PID to vectors, i.e. $e(t) = [x(t), y(t), z(t)]$ per axis

Intuition on PID control:

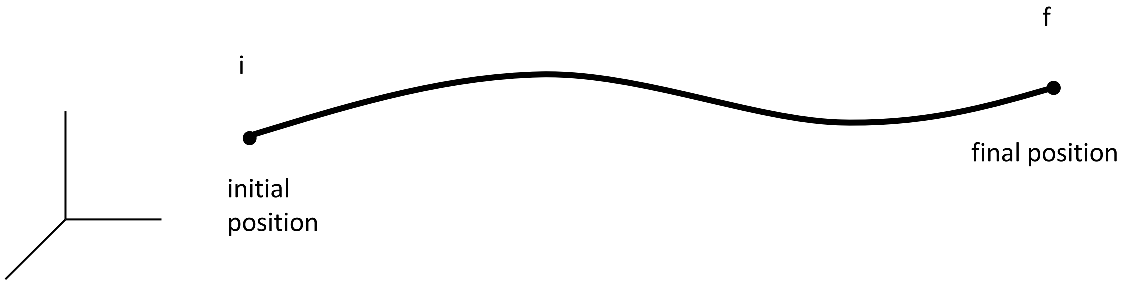
- Controlled spring-like behavior pulling x closer to x_{ref} with K_p
- Controlled damping with K_d
- Controlled zeroing of error with K_i



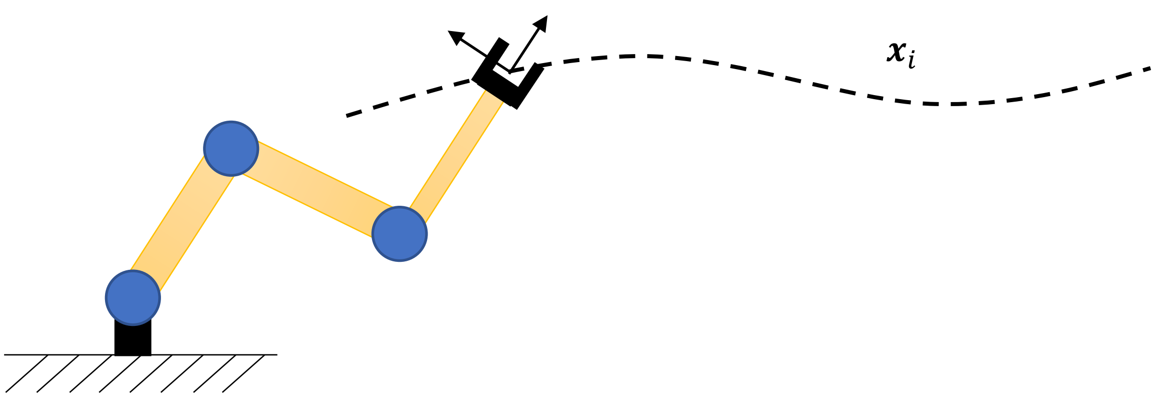
Local Control: *Open-Loop Trajectory* Control

Similar to setpoint control, but now x is a function of time.

1. Given a trajectory $x(t)$ in task space

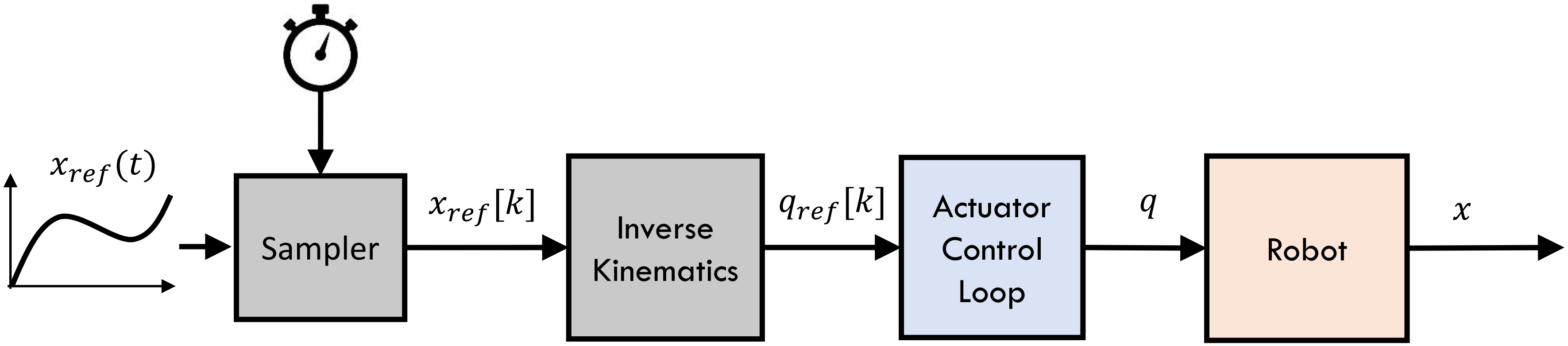


2. Interpolate Trajectory into small segments, $x(t) \rightarrow x_i$ for $i = 1 \ldots K$



For open-loop trajectory control,

1. $x_{ref}(t)$ is sampled at time t_k, i.e. $x[k]$
2. Use inverse kinematics on $x[k]$ to find $q[k]$
   - Can use Jacobian inverse or seed an IK solver with $q[k-1]$ to find $q_{ref}[k]$
3. Move joint i to q_i $\forall i$ under some single-joint-angle control loop.

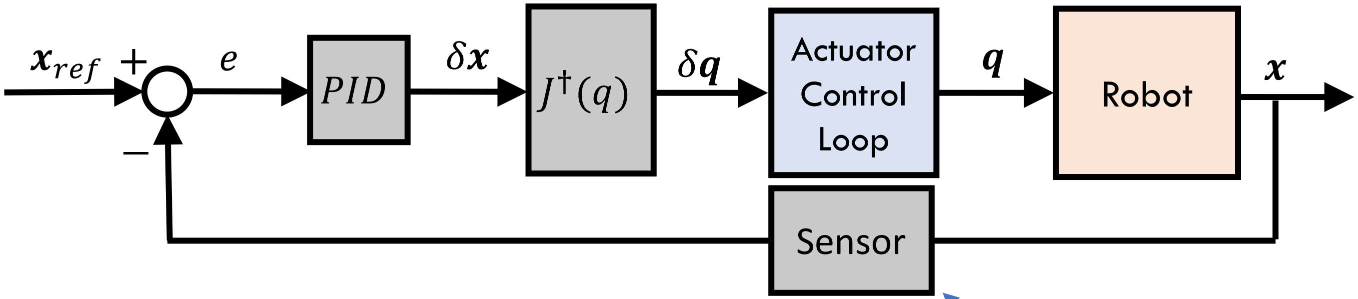


Remarks on *open-loop* control strategies:

- No feedback, so no idea if you are moving correctly towards the Cartesian setpoint
- You must have a very accurate model $x = f(q)$
- Works for high-rigidity robots

Local Control: *Closed-Loop Setpoint* Control

Presumes that you have a way to measure your output cartesian state x so it can correct for model inaccuracies or disturbances in your system.



Often left out of block diagram, assumed present.

Therefore, given $x_{ref} \in \mathbb{R}^6$ to track,

1. Find $e = x_{ref} - x$ to determine error from target

2. Convert e to a desired step towards target
   e.g., assume just a P-controller. Then, $\delta x = \mathrm{K}\Delta x$ where

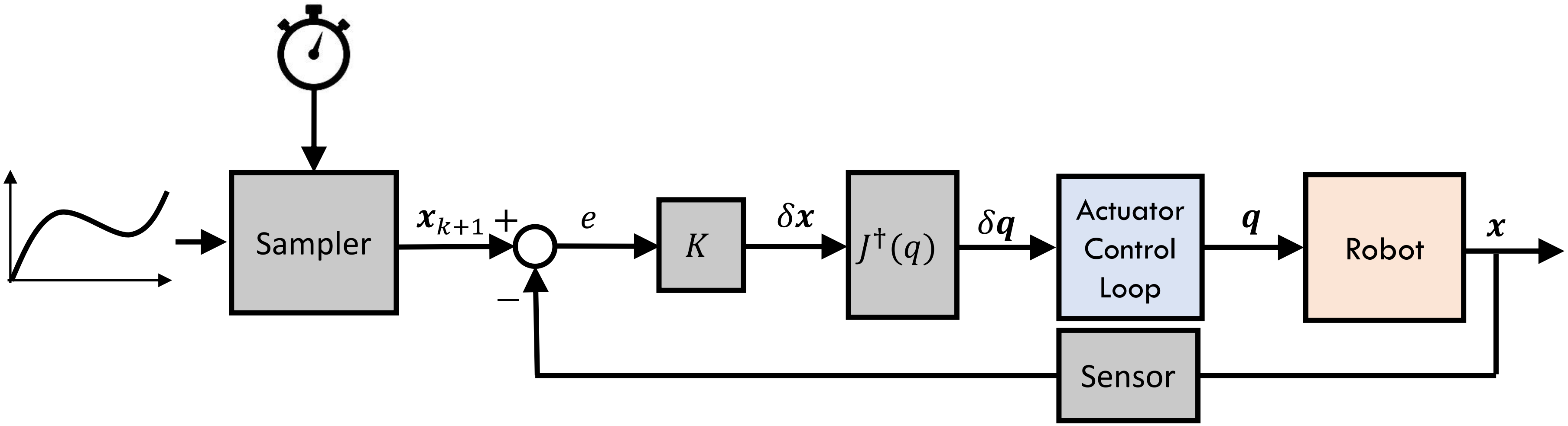
$$K = diag(k_x, k_y, k_z, k_{\omega x}, k_{\omega y}, k_{\omega z})$$

3. Calculate $\delta x = J^{\dagger}\delta q$, and move actuators δq amount
   in this case PID control describes a sub-stepping strategy.

Local Control: *Closed-Loop Trajectory* Control

Also called **Resolved Motion Rate Control**

1. Measure your current position x
2. For the desired position x_{k+1} at time $t = k$, find the step (error) $\delta x = x_{k+1} - x$.
3. Compute $J(q)$ and its pseudoinverse, $J^{\dagger}(q)$ for the current q.
4. Take a step $\delta q = J^{\dagger}(q)\delta x$ in your actuators to move x towards x_{k+1}.
5. Let $k = k + 1$. Repeat step 2.



Thought Questions:

1. Assume your Jacobian is modelled incorrectly;
   - Will tracking converge?

2. If you were to learn a system model, would it be better to learn a Jacobian function, or to learn a parametric form of the kinematics and then derive the Jacobian?

Final Note: Interpolation for Orientation Trajectories

- **Interpolation of a position vector is trivial**
  - Use linear interpolation of each axis x, y, z
- **Interpolation of an orientation vector is non-trivial**
  - Linear interpolation of $\theta_x, \theta_y, \theta_z$ does not produce an expected interpolated path. Reasoning: interpolation of orientations involves walking on a curved surface (i.e. unit ball in SE(3)). Path is not a straight line, so linear interpolation produces incorrect results.
- **Need to use quaternions (axis angle representation):**

  1. Let $q_a = quaternion(\theta_{ax}, \theta_{ay}, \theta_{az})$ and $q_b = quaternion(\theta_{bx}, \theta_{by}, \theta_{bz})$ where $q_a, q_b \in \mathbb{R}^4$
  2. Apply interpolation function SLERP:

$$q(t) = SLERP(q_a, q_b\, , t) = \frac{q_a\, sin\big((1-t)\theta/2\big) + q_b sin(t\theta/2)}{sin(\theta/2)}$$

  3. $\theta(t) = euler\big(q(t)\big)$