Resolved-Rate Motion Control.

Assume a low-level joint controller exists.
It can accurately execute trajectories designed by methods of Chapter 3.

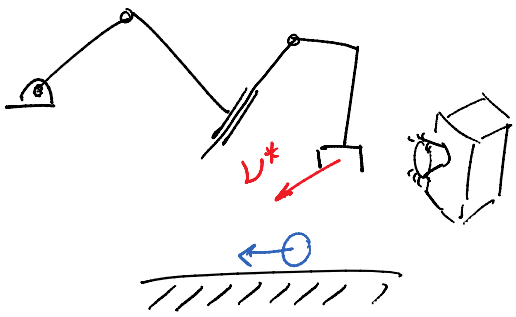
Suppose recent sensor input suggests that the robot end-effector should move in a particular direction at a given speed.

How would one implement this high-level controller.

$$v = J(q)\,\dot{q}$$

v^* – desired end-effector twist

Grab rolling ball

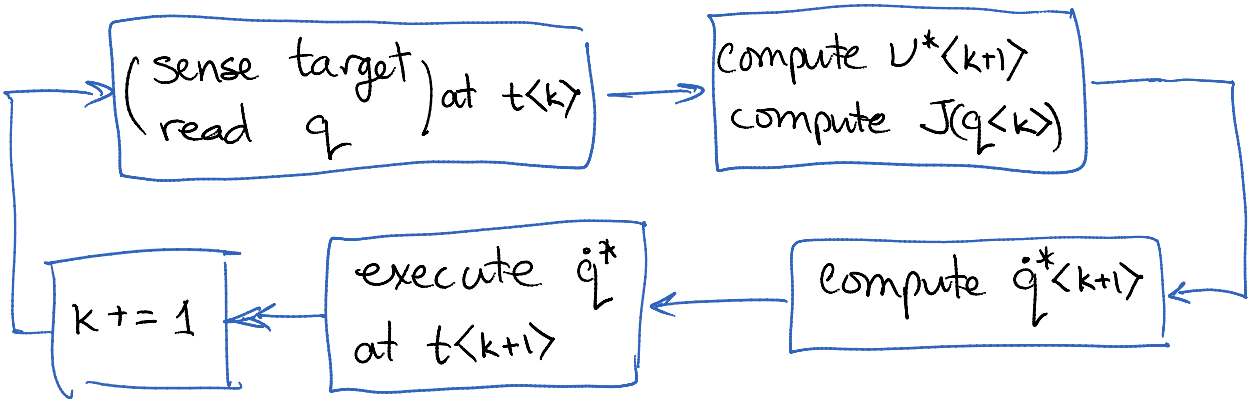


What is desired joint displacement rate, $\dot{q}^*$?

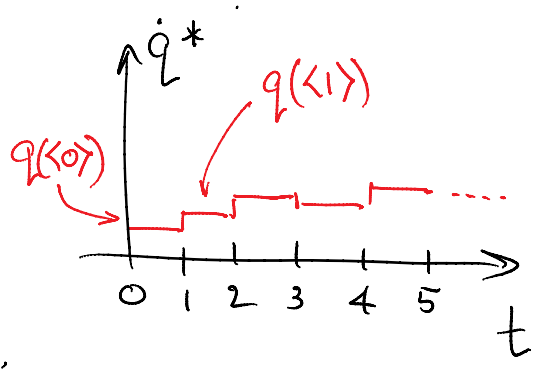
$$\dot{q}^* = J^{-1} v^*$$

Controller is runs on real-time OS.
Let k be the time step.

Consider 1 time step in a visual servoing task:



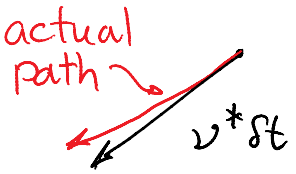
Note: $\dot{q}^*\langle k+1\rangle$ is executed until $t\langle k+2\rangle$



If API gives access to positions of joints but not velocities, then use known controller period δt to convert $\dot{q}^*$ to $\delta q^* = \dot{q}^* \delta t$.

$$q\langle k+1\rangle = J^{-1}(q\langle k\rangle)\, v^* \delta t + q\langle k\rangle$$

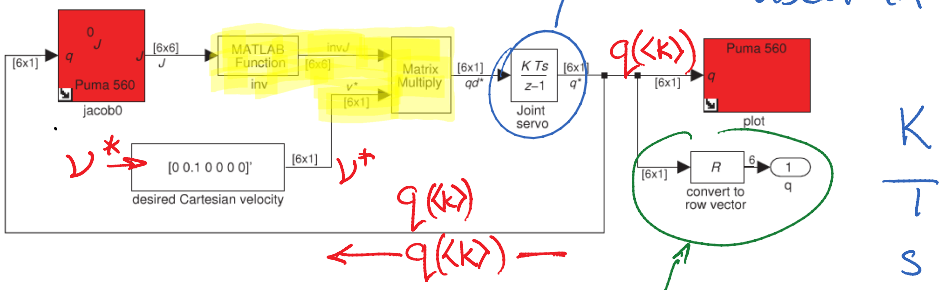
Note: For both approaches, J's dependence on q $\Rightarrow$ trajectories will drift.



Note: Drift will be worst near singular configurations.

Simulink block diagram

```
>> sl_rrmc
>> r = sim('sl_rrmc')
```



z is a discrete-time variable used in control analysis.

$\frac{K}{1}$

s

Data output

$J \backslash v^*$ is faster than $J^{-1} v^*$

If one used above approach to execute long preplanned trajectories, drift can be significant.

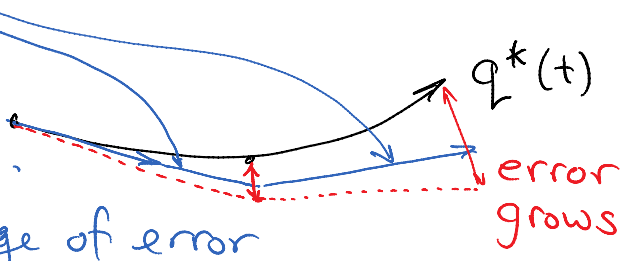
How do we eliminate drift?

Compute error and insert correction.

Consider two steps along $q^*(t)$, an underlying desired trajectory in $SE(3)$.

First method: $\delta q^* = \bar{J}^{-1}(q(\langle k \rangle))\, v^*$

1. compute tangent.
2. try to move along it.
3. repeat (w/o knowledge of error



Second method (w/error correction)

$$q^*(\langle k+1 \rangle) = \bar{J}^{-1}(q\langle k \rangle) \cdot \delta q$$

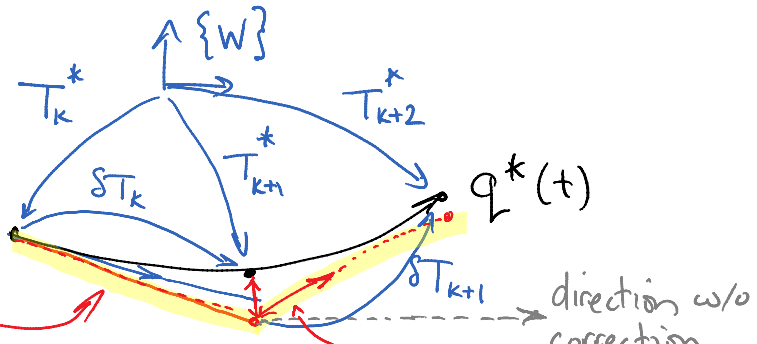
but δq is computed differently:

$$T_k^* \, \delta T_k = T_{k+1}^*$$

$$\delta T_k = (T_k^*)^{-1} \, T_{k+1}^*$$

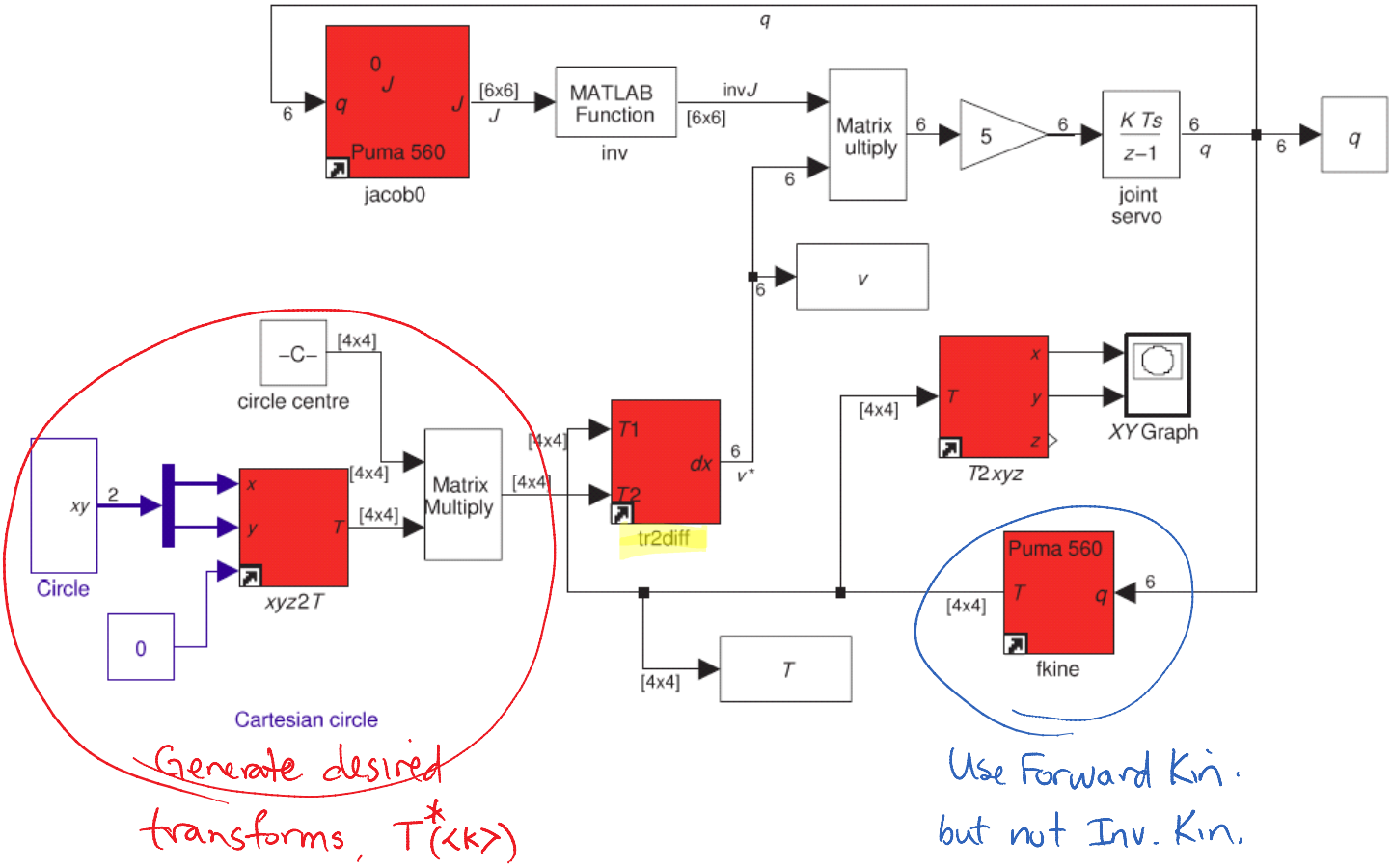
$v^* \delta t$ is replaced by

tr2diff(δT) δt



Step with no initial error is the same

Step with initial error has component to remove error.

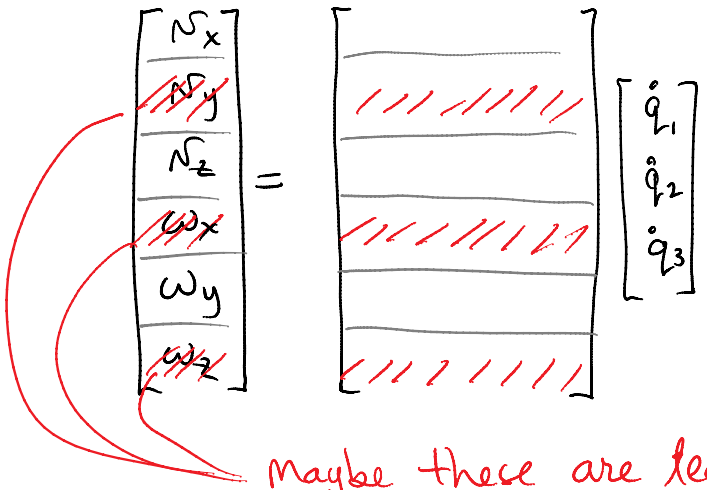


Note: When $J^{-1} \nexists$ method must be modified.

The simplest idea....

Underactuated: $\dot{v} = J\dot{q}$ delete some rows of J!

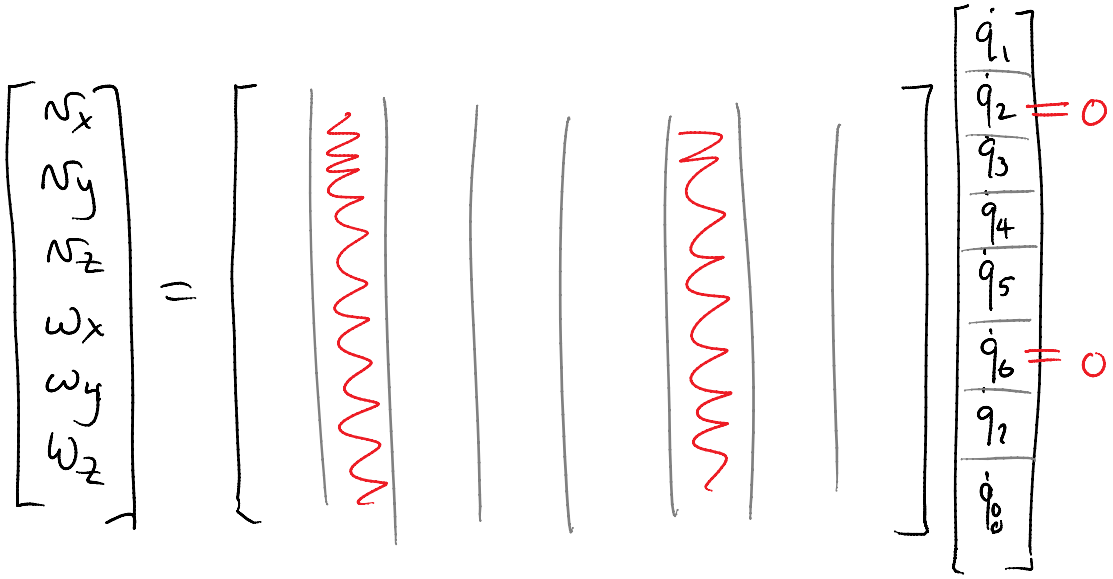
Redundant: (over-actuated) $\dot{v} = J\dot{q}$ delete some columns of J!



What should be eliminated?

Maybe these are least important.

Resulting J is (3×3).

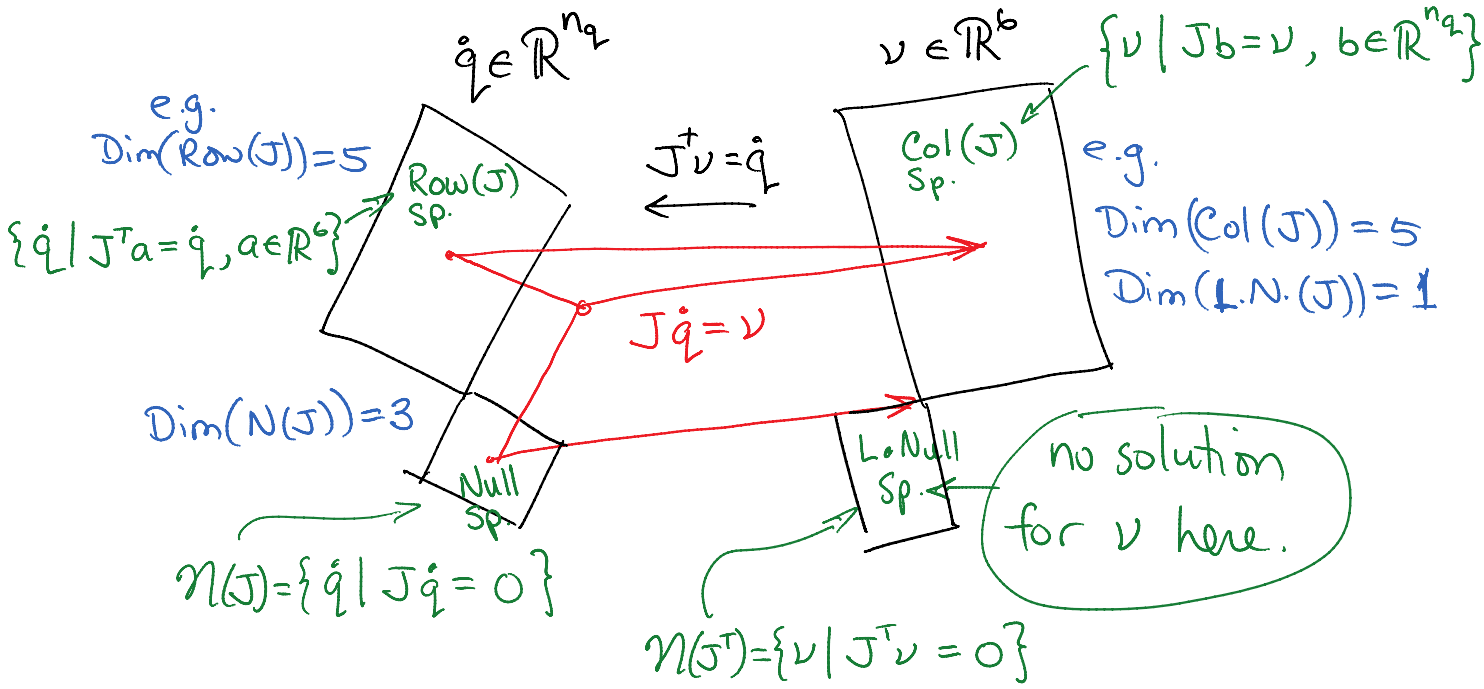


Perhaps lock some joints!

Resulting J is (6×6).

Pseudo-inverse.

When J is singular, $\nexists\ J^{-1}$, and $\dot{q}$ may $\nexists\ \ni\ J\dot{q} = \nu^*$.



Case 1: $Rank(J) < 6 \Rightarrow Dim(L.N(J)) > 0$

No solution for $\nu \in L.N.(J)$

$$Rank(J) \leq \min(nrows, ncols)$$

Two main options:

A.) Eliminate rows of J until submatrix is non-singular

B.) Use $\dot{q}^* = J^+\nu^*$

J^+ ← pinv(J) ← pseudo-inverse

$\dot{q}^*$ minimizes $\|J\dot{q}^* - \nu^*\|$

Case 2: $\text{Rank}(J) = 6 \Rightarrow \text{Dim}(\text{Null}(J)) > 0$

$\text{Dim}(L.N(J)) = 0$

Solution exists $\forall\ v \in \mathbb{R}^6$

$$\dot{q}^* = J^+ v^* + \mathcal{N}(J)\,\dot{q}_{ns}$$

$\dot{q}_{ns}$: arbitrary values

$\mathcal{N}(J)$: Null space basis vectors.

$\text{Dim}(\text{Null}(J)) = \eta$ = "nullity" of J.

$\dot{q}_{ns} \in \mathbb{R}^{\eta}$.

You can choose $\dot{q}_{ns}$ to optimize a desireable motion metric.

e.g. $\min \|\dot{q}^*\| \Rightarrow \dot{q}_{ns} = 0$