1a

When there is just translation

$$^{A}P = {}^{B}P + {}^{A}P_{B_{org}}$$

When there is just rotation

$$^{A}P = {}^{A}_{B}R \, {}^{B}P \quad - \quad {}^{A}_{B}R = [\, {}^{A}\hat{X}_B \;\; {}^{A}\hat{Y}_B \;\; {}^{A}\hat{Z}_B \,] = \begin{bmatrix} {}^{B}\hat{X}_A \\ {}^{B}\hat{Y}_A \\ {}^{B}\hat{Z}_A \end{bmatrix}^T = {}^{B}_{A}R^T = {}^{B}_{A}R^{-1}$$

When both

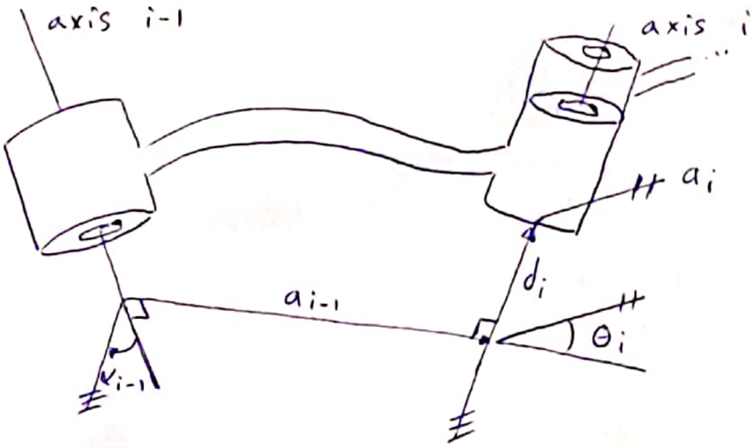
$$^{A}P = {}^{A}_{B}R \, {}^{B}P + {}^{A}P_{B_{org}} = {}^{A}_{B}T \, {}^{B}P \rightarrow \begin{bmatrix} {}^{A}P \\ 1 \end{bmatrix} = \underbrace{\left[\begin{array}{c|c} {}^{A}_{B}R & {}^{A}P_{B_{org}} \\ \hline 0\;0\;0 & 1 \end{array}\right]}_{4\times 4} \begin{bmatrix} {}^{B}P \\ 1 \end{bmatrix}$$

$$^{A}_{B}T = \left[\begin{array}{ccc|c} r_1 & r_2 & r_3 & Dx \\ r_4 & r_5 & r_6 & Dy \\ r_7 & r_8 & r_9 & Dz \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

Concatenation

$$^{B}P = {}^{B}_{C}T \, {}^{C}P$$

$$^{A}P = {}^{A}_{B}T \, {}^{B}P = {}^{A}_{B}T \, {}^{B}_{C}T \, {}^{C}P = {}^{A}_{C}T \, {}^{C}P \rightarrow {}^{A}_{C}T = \begin{bmatrix} {}^{A}_{B}R \, {}^{B}_{C}R & {}^{A}P_B + {}^{A}_{B}R \, {}^{B}P_C \\ 0\;0\;0 & 1 \end{bmatrix}$$



α_{i-1} : link twist around X_{i-1}

a_{i-1} : link length along X_{i-1}

d_i : link offset along Z_i (variable if prismatic)

θ_i : joint angle around Z_i (variable if revolute)

- Z_i along axis i
- X_i along a_i (direction from joint i to $i+1$) or to be normal to the plane in case $a_i = 0$
- Origin of frame $\{i\}$ where X_i and Z_i intersect

α_i : angle $(\underline{Z_i}, Z_{i+1})$ about $\underline{X_i}$ $\qquad$ d_i : distance $(X_{i-1}, \underline{X_i})$ along $\underline{Z_i}$

a_i : distance $(\underline{Z_i}, Z_{i+1})$ along $\underline{X_i}$ $\qquad$ θ_i : angle $(X_{i-1}, \underline{X_i})$ about $\underline{Z_i}$

$$ {}^{i-1}_{\ \ i}T = \begin{bmatrix} c\theta_i & -s\theta_i & 0 & a_{i-1} \\ s\theta_i c\alpha_{i-1} & c\theta_i c\alpha_{i-1} & -s\alpha_{i-1} & -s\alpha_{i-1} d_i \\ s\theta_i s\alpha_{i-1} & c\theta_i s\alpha_{i-1} & c\alpha_{i-1} & c\alpha_{i-1} d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

↳ motor and transmission
↳ rigid-body inertial "
↳ kinematic parameters

Matlab Robert Corke Robot Toolbox commands

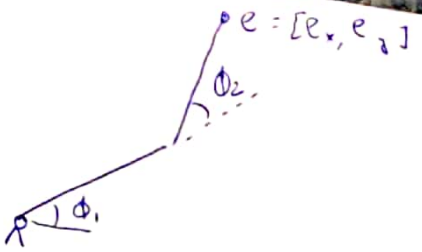
0 : R
1 : P

- Link [theta, d, a, alpha, sigma, mdh, offset, qlim, ...] → object that holds info related to a robot li...
- SerialLink → a class that represents a serial-link arm-type robot using DH parameters
- teach → drive the graphical robot
- plot → display 3D graphical robot model
- base → pose of robot's base (4×4 matrix homogeneous transformation)
- transl → homogeneous transform 4×4 representing pure translation
- trotx, troty, trotz → " " " " " rotation
- fkine → compute forward kinematics
- ikine → " inverse " using iterative numerical method using pseudoinverse of Jacobian
- ikine6s → " " " for 6 axis spherical wrist revolute robot
- ikunc → " " " using optimization
- trplot → draws a 3D coordinate frame represented by the homogeneous transform
- hold on

$$ J(f, x) = \frac{df}{dx} = \underbrace{\begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_3} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \cdots & \frac{\partial f_M}{\partial x_N} \end{bmatrix}}_{M \times N} $$

M : degrees of freedom
N : degrees of mobility
↳ # of joints

1b



$$J(e,\phi) = \begin{bmatrix} \frac{\partial e_x}{\partial \phi_1} & \frac{\partial e_x}{\partial \phi_2} \\ \frac{\partial e_y}{\partial \phi_1} & \frac{\partial e_y}{\partial \phi_2} \end{bmatrix}$$

If small change in ϕ values $\rightarrow \frac{\partial e}{\partial \phi} \approx \frac{\Delta e}{\Delta \phi} \rightarrow \Delta e \approx \frac{\partial e}{\partial \phi} \Delta \phi = J(e,\phi)\Delta\phi = \boxed{J\Delta\phi}$

Valid near the current configuration

↳ if small change in e $\rightarrow \Delta\phi \approx J^{-1}\Delta e$

Forward kinematics is a non-linear function as involves trigonometry

↳ Small steps → Iterative process

Inverse Kinematics technique

while (e is too far from g):

- $J(e,\phi)$ for the current pose ϕ
- J^{-1}
- $\Delta e = \beta(g-e)$
- $\Delta\phi = J^{-1}\Delta e$
- $\phi = \phi + \Delta\phi$
- Forward kinematics → new e

e : position
g : goal
β : step (0.01)

Solvable condition (sufficient):

A manipulator with 6 revolute joints will have a closed form solution if

- 3 neighbouring joint axes intersect at a point

7 degree of mobility has a non-square J matrix, so non-invertible but pseudo inverse:

$$J^{*-1} = J^T(JJ^T)^{-1}$$

↳ Kinematic redundancy

- ↳ increased mobility
- ↳ collision avoidance

Linear velocity → attribute of a point

Angular velocity → attribute of a body (~~with~~ → of the frame attached to it)

Scanned by CamScanner

$$ {}^{i+1}\omega_{i+1} = {}^{i+1}_{i}R\,{}^{i}\omega_i + \underbrace{\dot{\theta}_{i+1}\,{}^{i+1}\hat{Z}_{i+1}}_{\to\ {}^{i+1}\begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_{i+1} \end{bmatrix}} $$

$$ {}^{i+1}v_{i+1} = {}^{i+1}_{i}R\,({}^{i}v_i + {}^{i}\omega_i \times {}^{i}P_{i+1}) $$

From link to link we can compute the rotational and linear velocities of the end-effector

$$ {}^{0}V = \begin{bmatrix} {}^{0}v \\ {}^{0}\omega \end{bmatrix} = {}^{0}J(\theta)\,\dot{\theta} \qquad A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \qquad A^{-1} = \frac{1}{\det A}\,\mathrm{adj}\,A^{T} = $$

$$ = \frac{1}{ad-bc}\begin{bmatrix} d & -c \\ -b & a \end{bmatrix}^{T} $$

As long as J is non-singular ($\det J \neq 0$) then it's invertible

$$ \dot{\theta} = J^{-1}(\theta)\,V $$

Singularities

- Workspace boundary singularities
  ↳ when the manipulator is fully stretched or folded back on itself
- Workspace interior singularities
  ↳ when two or more joint axes line up

In a singular configuration, one or more degrees of freedom is lost (if all, then movement is impossible)

Static Forces Balance

$$ {}^{i}f_i - {}^{i}f_{i+1} = 0 \longrightarrow {}^{i}f_i = {}^{i}f_{i+1} \to \boxed{{}^{i}f_i = {}^{i}_{i+1}R\,{}^{i+1}f_{i+1}} $$

$$ {}^{i}n_i - {}^{i}n_{i+1} - {}^{i}P_{i+1} \times {}^{i}f_{i+1} = 0 \to {}^{i}n_i = {}^{i}n_{i+1} + {}^{i}P_{i+1} \times {}^{i}f_{i+1} $$

$$ \hookrightarrow \boxed{{}^{i}n_i = {}^{i}_{i+1}R\,{}^{i+1}n_{i+1} + {}^{i}P_{i+1} \times {}^{i}f_{i+1}} $$

Torques needed at the joints to balance these forces and moments acting on the links

$$ \hookrightarrow \left.\begin{array}{l} \tau_i = {}^{i}n_i^{T}\,{}^{i}\hat{Z}_i \\ \tau_i = {}^{i}f_i^{T}\,{}^{i}\hat{Z}_i \end{array}\right\} $$

$$ F\delta x = \tau\delta\theta \to F^{T}\delta x = \tau^{T}\delta\theta $$

$$ \delta x = J\delta\theta \longrightarrow F^{T}J\delta\theta = \tau^{T}\delta\theta \to F^{T}J = \tau^{T} \to $$

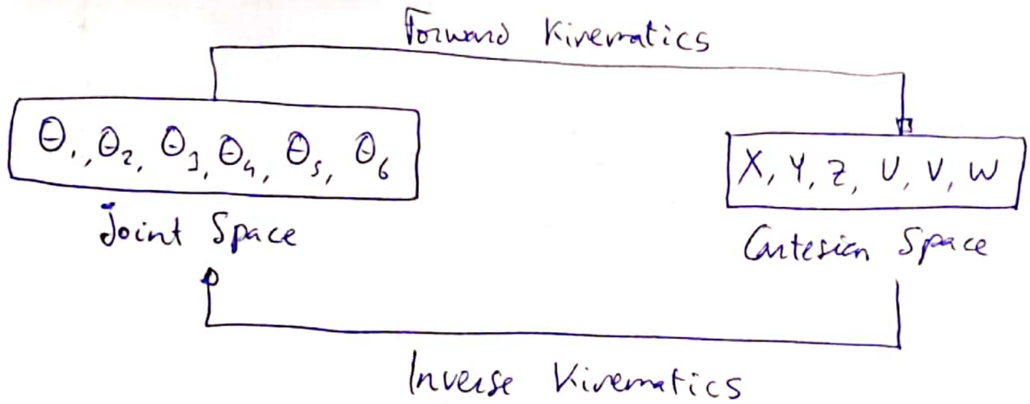
$$ \to \tau = J^{T}F $$

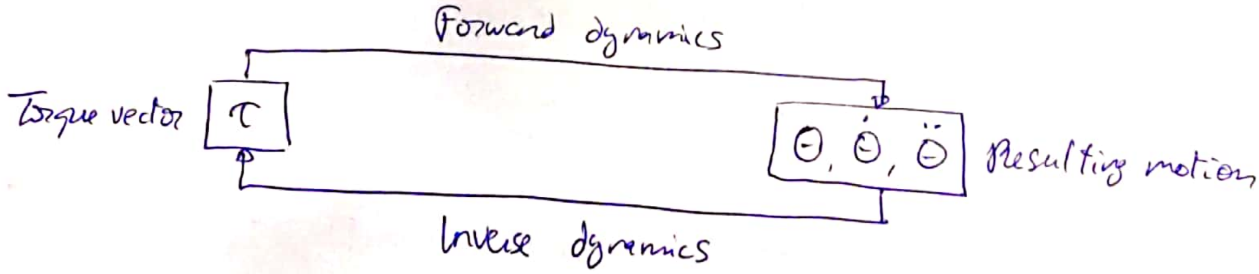
General velocity $V = \begin{bmatrix} v \\ \omega \end{bmatrix}_{6\times1}$ General force $F = \begin{bmatrix} F \\ N \end{bmatrix}_{6\times1}$

Dynamics are the study of forces / torques required to cause motion

The dynamic equation is function of :

- Mass of each link
- Mass distribution for each link $\rightarrow$ inertia tensor
- length of each link
- Joint type
- Manipulator configuration and joint locations





Two approaches

- Newton-Euler dynamic formulation
  - Outward iterations
  - Inward iterations
- Lagrangian formulation of manipulato dynamics
  - Kinetic energy
  - Potential energy

Newton - Euler equations

$$F = m\,\dot{v}_c$$

$$N = {}^{c}I\,\dot{\omega} + \omega \times {}^{c}I\,\omega$$

Method used to compute torques given a trajectory $\rightarrow$ Inverse dynamics

- Outward iterations

Compute velocities and acceleration, forces and torques at links centre of mass

↳ done iteratively link-by-link starting by link 1 until link n

starting by $i = 0$

$${}^{i+1}\omega_{i+1} = {}^{i+1}_{i}R\,{}^{i}\omega_i + \underbrace{\dot{\theta}_{i+1}\,{}^{i+1}\hat{Z}_{i+1}}_{\begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_{i+1} \end{bmatrix}}$$

$\rightarrow$ this would be zero in case of prismatic joint

$${}^{i+1}\dot{\omega}_{i+1} = {}^{i+1}_{i}R\,{}^{i}\dot{\omega}_i + {}^{i+1}_{i}R\,{}^{i}\omega_i \times \dot{\theta}_{i+1}\,{}^{i+1}\hat{Z}_{i+1} + \ddot{\theta}_{i+1}\,{}^{i+1}\hat{Z}$$

$${}^{A}\Omega_C = {}^{A}\Omega_B + {}^{A}_{B}R\,{}^{B}\Omega_C$$

$${}^{i+1}\dot{v}_{i+1} = {}^{i+1}_{i}R\left({}^{i}\dot{\omega}_i \times {}^{i}P_{i+1} + {}^{i}\omega_i \times \left({}^{i}\omega_i \times {}^{i}P_{i+1}\right) + {}^{i}\dot{v}_i\right)$$

$${}^{i+1}\dot{v}_{C_{i+1}} = {}^{i+1}\dot{\omega}_{i+1} \times {}^{i+1}P_{C_{i+1}} + {}^{i+1}\omega_{i+1} \times \left({}^{i+1}\omega_{i+1} \times {}^{i+1}P_{C_{i+1}}\right) + {}^{i+1}\dot{v}_{i+1}$$

$${}^{i+1}F_{i+1} = m_{i+1}\,{}^{i+1}\dot{v}_{C_{i+1}}$$

$${}^{i+1}N_{i+1} = {}^{C_{i+1}}I_{i+1}\,{}^{i+1}\dot{\omega}_{i+1} + {}^{i+1}\omega_{i+1} \times {}^{C_{i+1}}I_{i+1}\,{}^{i+1}\omega_{i+1}$$

• Inward iterations

Compute forces and torques at joints
Outward iterations are required to be computed previously

Force balance relationship → ${}^iF_i = {}^if_i - {}^i_{i+1}R\,{}^{i+1}f_{i+1}$

Summing torques about center of mass and setting them equal to zero

→ $${}^iN_i = {}^in_i - {}^in_{i+1} + (-{}^iP_{c_i}) \times {}^if_i - ({}^iP_{i+1} - {}^iP_{c_i}) \times {}^if_{i+1}$$
$$= {}^in_i - {}^i_{i+1}R\,{}^{i+1}n_{i+1} - {}^iP_{c_i} \times {}^iF_i - {}^iP_{i+1} \times {}^i_{i+1}R\,{}^{i+1}f_{i+1}$$

Arranging:

$${}^if_i = {}^i_{i+1}R\,{}^{i+1}f_{i+1} + {}^iF_i$$

$${}^in_i = {}^iN_i + {}^i_{i+1}R\,{}^{i+1}n_{i+1} + {}^iP_{c_i} \times {}^iF_i + {}^iP_{i+1} \times {}^i_{i+1}R\,{}^{i+1}f_{i+1}$$

Computed from link n until link 1, inwards toward base of robot

$$\tau_i = {}^in_i^T\,{}^i\hat{z}_i$$

$$\tau_i = {}^if_i^T\,{}^i\hat{z}_i$$

The effect of the gravity loading on the links can be included by setting ${}^0\dot{v}_0 = G = g\hat{Y}_0$

In Matlab toolbox:

m: link mass
r: link COG (center of gravity) [3×1]
I: link inertia matrix [3×3]
G: gear ratio (usually 1)
Jm: motor inertia (usually 0)
accel: joint acceleration
dyn: show dynamic properties of links

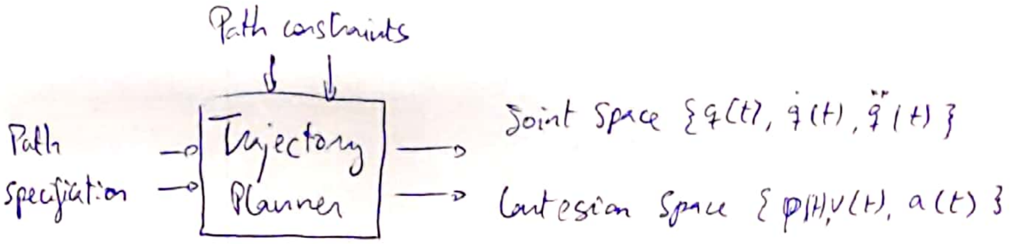
gravload: compute gravity joint force
cinertia: cartesian inertia matrix
rne: inverse dynamics
fdyn: forward dynamics
coriolis: compute centripetal/coriolis force
itorque: compute inertia torque

State - space equation format

$$\tau = M(\theta)\ddot{\theta} + V(\theta, \dot{\theta}) + G(\theta)$$

- $M(\theta)_{n\times n}$: mass matrix
- $V(\theta, \dot{\theta})_{n\times 1}$: centrifugal and Coriolis terms
- $G(\theta)_{n\times 1}$: gravity terms

Trajectory planning



Smoothness in math → movement function is continuous and ~~derivable~~ with non-null derivative

{ Path : Denotes the locus of points in the joint or cartesian space
{ Trajectory : Is a path on which a time law is specified
 L involves : - finding the prescribed path
 - collision avoidance
 - concerns about actuator saturation

Point - to - point Motion

$\theta(0) = \theta_0 \qquad \theta(t_f) = \theta_f$

$\dot{\theta}(0) = 0 \qquad \dot{\theta}(t_f) = 0$

$q(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3$

$\dot{q}(t) = a_1 + 2a_2 t + 3a_3 t^2$

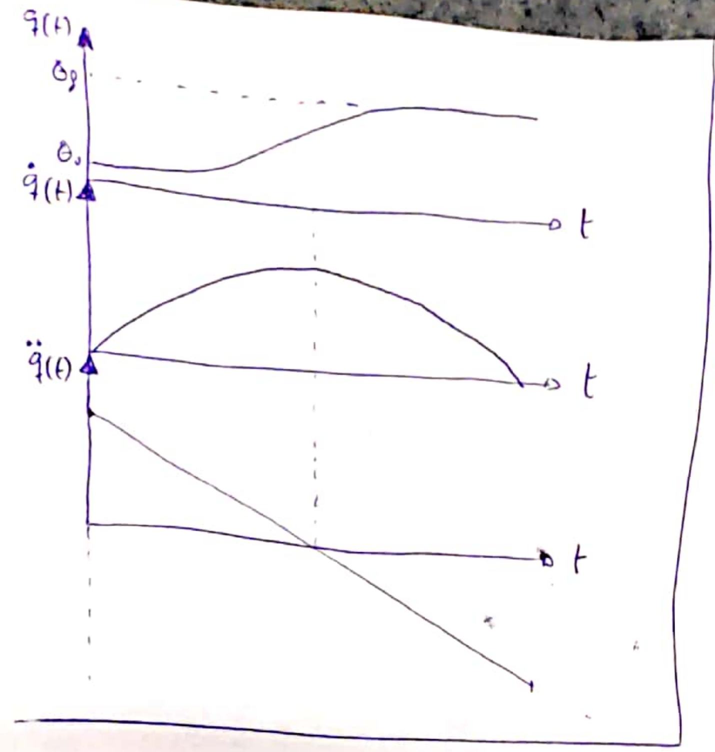
$\ddot{q}(t) = 2a_2 + 6a_3 t$

(1) $q(0) = \theta_0 \equiv a_0$ (2) $q(t_f) = a_0 + \cancel{a_1 t_f}^{0} + a_2 t_f^2 + a_3 t_f^3 \equiv \theta_f$

(2) $\dot{q}(0) = 0 \equiv a_1$ (4) $\dot{q}(t_f) = \cancel{a_1}^{0} + 2a_2 t_f + 3a_3 t_f^2 \equiv 0$

$\hookrightarrow t_f(2a_2 + 3a_3 t_f) = 0 \to a_2 = -\frac{3a_3}{2} t_f = \frac{3}{t_f^2}(\theta_f - \theta_0)$

(3) $\theta_0 + \left(-\frac{3a_3}{2} t_f\right) t_f^2 + a_3 t_f^3 = \theta_f \to \theta_0 - \frac{a_3}{2} t_f^3 = \theta_f \to a_3 = -\frac{2}{t_f^3}(\theta_f - \theta_0)$



If we also want to add as constraints.

$\ddot{\theta}(0)$ and $\ddot{\theta}(t_f)$

we would have 6 constraints.

Order of polynomial = # constraints - 1

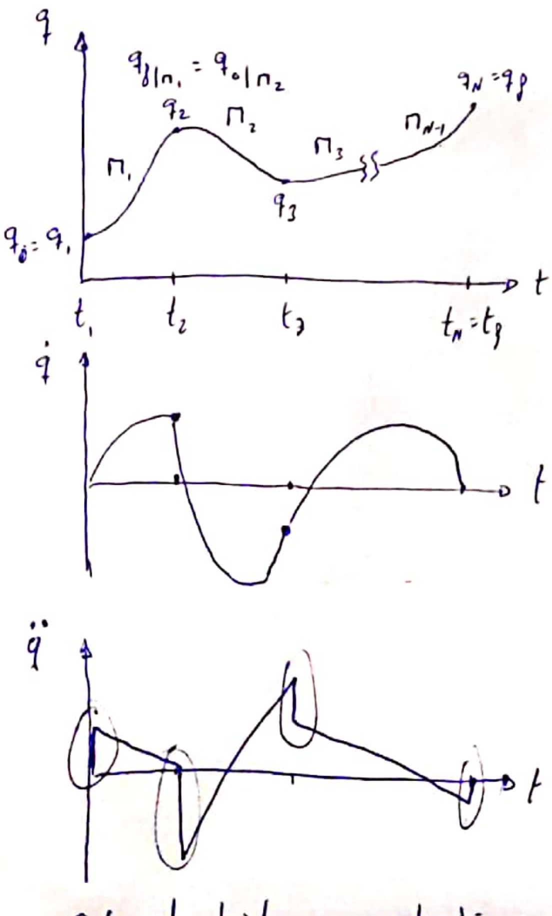
So we need a 5th order polynomial

$$q(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5$$

↳ one per joint

↑ the polynomial order
- ↳ ↑ oscillation
- ↳ ↑ numerical accuracy
- ↳ heavier to solve

Solution → Suitable number of low-order interpolating polynomials



$$\Pi_k(t_k) = q_k$$
$$\Pi_k(t_{k+1}) = q_{k+1}$$
$$\dot{\Pi}_k(t_k) = \dot{q}_k$$
$$\dot{\Pi}_k(t_{k+1}) = \dot{q}_{k+1}$$

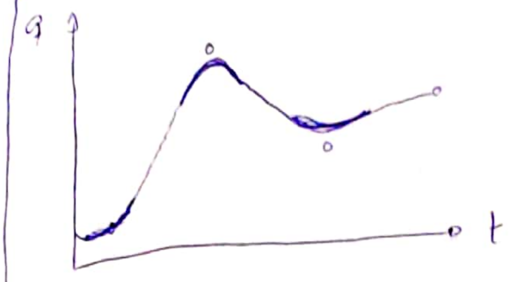
$$\dot{\Pi}_k(t_{k+1}) = \dot{\Pi}_{k+1}(t_{k+1})$$

Extremely high ~~overaccelerations~~ → jerk

Solution → Interpolating using SPLINES
- ↳ Continuous accelerations at Path Points
- ↳ Creation of virtual points

• Sequence of cubic polynomials that indicates smooth functions that interoperate a sequence of points ensuring continuity of the function and its derivatives

Interpolating linear Polynomials with Parabolic Blends



The function $q(t)$ must have a parabolic profile around t_k

↳ The trajectory is a sequence of linear and quadratic polynomials

↳ a discontinuity on $\ddot{q}(t)$ is tolerated

linear interpolation causes discontinuity in velocity

↳ Problems
- ↳ Intermediate points unreachable
- ↳ High joint rates mean singularity

Euler angles

X: Roll (ϕ)

Y: Pitch (θ)

Z: Yaw (ψ)

Quaternions

q_1

q_2

q_3

q_4

RobotWare: Firmware inside the robot controller

Drives are the computers sending signals to the motors, low level language

RobotStudio: Pick and place flow

↳ Design 3D system
- ↳ Define work objects/targets
  - Set/fix orientations
  - ↳ Synchronize controller
    - ↳ Create flow control on RAPID code
      - ↳ Offline testing
        - ↳ Deploy
          - ↳ Fix targets positions by teaching/updating targets

RobotStudio: Path creation flow

↳ Design 3D system
- ↳ Define work objects/targets
  - Set/fix orientations
  - ↳ Create path
    - ↳ Synchronize controller
      - ↳ Modify/create RAPID code
        - ↳ Test
          - ↳ Deploy

TPWrite String [\Num] [\Bool] [\Pos] [\Orient]

$\underbrace{\text{TPWrite}}_{\text{Instruction}}$ $\underbrace{\text{String}}_{\text{Compulsory argument}}$ $\underbrace{\text{[\textbackslash Num] [\textbackslash Bool] [\textbackslash Pos] [\textbackslash Orient]}}_{\text{Optional arguments, Mutually exclusive}}$

Persistent Variable : The same as an ordinary variable but with the difference that it remembers the last value it was assigned, even if the program is reseted

```
PERS __ := X
PROC main()

    __ := Y
ENDPROC
```

```
PERS __ := Y
PROC main()
    __ : Y
END PROC
```

Constant : CONST

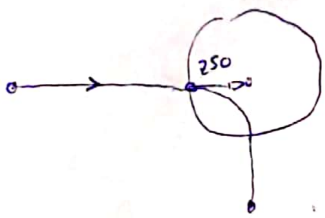
Variable : VAR

RAPID : High level programming language to control robots

MoveL	To Point	Speed	Zone	Tool
	p10	v1000	fine	tool0
Straight line	Position	mm/s		

→ is the mounting flange at the tip of the robot that should go to that p10

→ the robot shall go exactly to the specified position and not cut any corners on its way to the next position



MoveJ ... ... to move the robot quickly from one point to the other when a straight line is not required

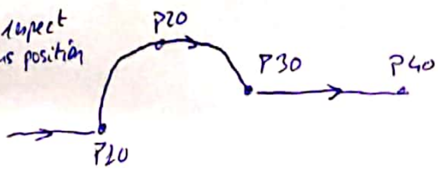
MoveC to move circularly in an arc

→ MoveL p10, v500, fine, tPen;

MoveC p20, p30, v500, fine, tPen; with respect previous position

($p20$: intermediate, $p30$: final)

MoveL p40, v500, fine, tPen



SetDO Rob_Gripper_Set, 1;

↳ Setting digital output

VAR robtarget pre_pick;

pre_pick := Offs (Pick, 0, 0, -100) } Definition of offsets

WaitRob \InPos; → Wait until the robot gets to the position

CONST robtarget Pick := [[600, -100, 800], [1, 0, 0, 0], [0, 0, 0, 0], [9E9, 9E9, 9E9, 9E9, 9E9, 9E9]