IMPERIAL

Hamlyn Centre for Robotic Surgery

HARMS Lab
Human-centred Automation, Robotics and Monitoring in Surgery

Week 8
Path Planning

MRes in Medical Robotics and Instrumentation

Dr George Mylonas

Contents of the lecture

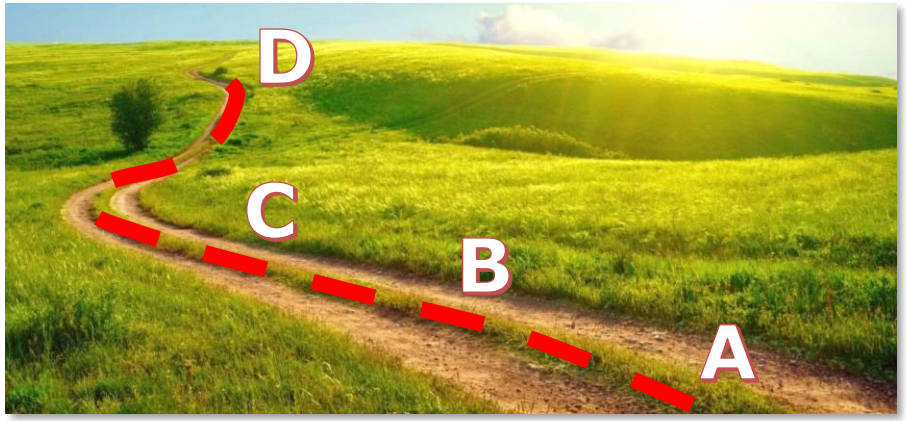
- Path and trajectory
- Trajectory planning in robotics
- Smooth and continuous trajectories (**Hint:** Helpful for your coursework)

- John J Craig, Introduction to robotics – mechanics and control, **Chapter 7**

Path and Trajectory

What is a path?

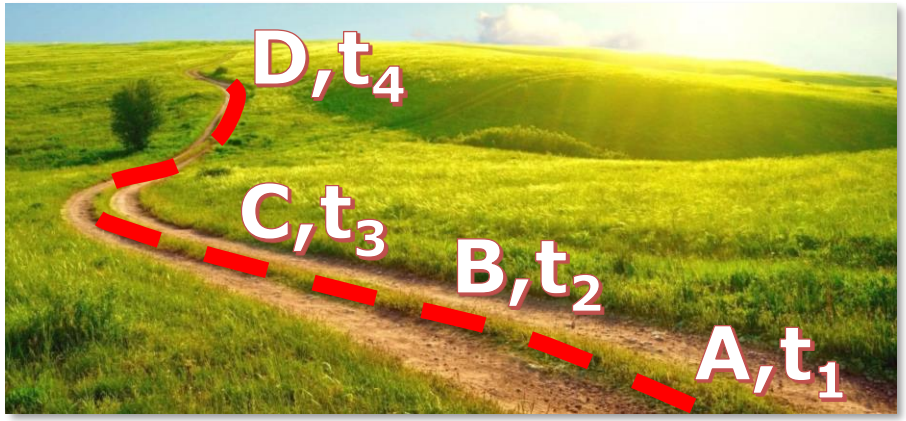
- The way of getting from a start point to an end point,
- By following a set of intermediate points (**waypoints or via points**) that connect start to end;
- It does not include any information on the speed or acceleration we have used along the path;
- Hence, time is not of interest.



cedarscobblehill.com/stay-path-recovery/

What is a trajectory?

- Is a path followed while also observing time, i.e., having a schedule while moving along a path;
- Speed and acceleration need to be accounted for.



Trajectory Planning in Robotics

- **Trajectory planning is a description of how a robot is required to follow a path over time**

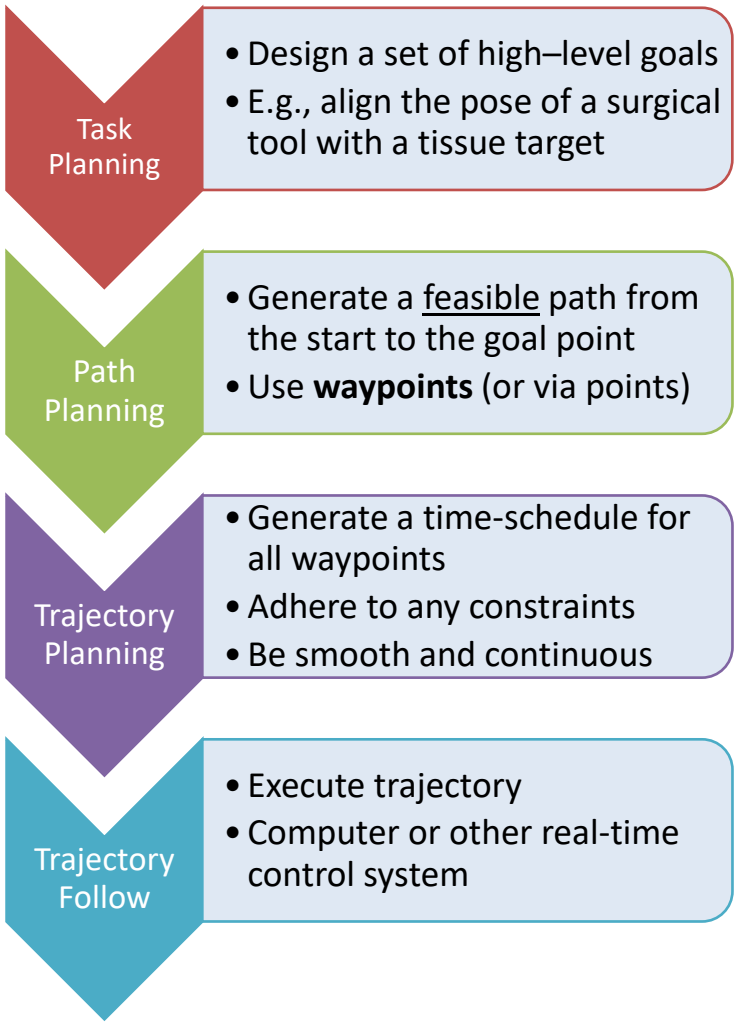
- **Aim:** To move a robot or its end-effector from an initial pose to a final pose, whilst adhering to motion/kinematic **constraints**, such as:
  - Staying within the physical limits of the joints
  - Avoiding collision or self-collision or compromise of structural integrity, e.g., through resonance, high velocities and accelerations etc
  - Considering task-imposed constraints, e.g., conform to a dynamic environment
  - Avoiding robot singularities

- Involves following the planned **waypoints** over time
  - Interpolate for all other points in between

NOTE: Waypoints are in reality "**wayframes**", i.e., both position and orientation!

- Trajectory planning can be performed in either:
  - **Task/Cartesian/Operational Space:** waypoints and interpolations are considered between poses of the end-effector.
  - **Joint/Configuration Space:** waypoints and interpolations are directly on the joint position (angle for revolute, displacement for prismatic joints)

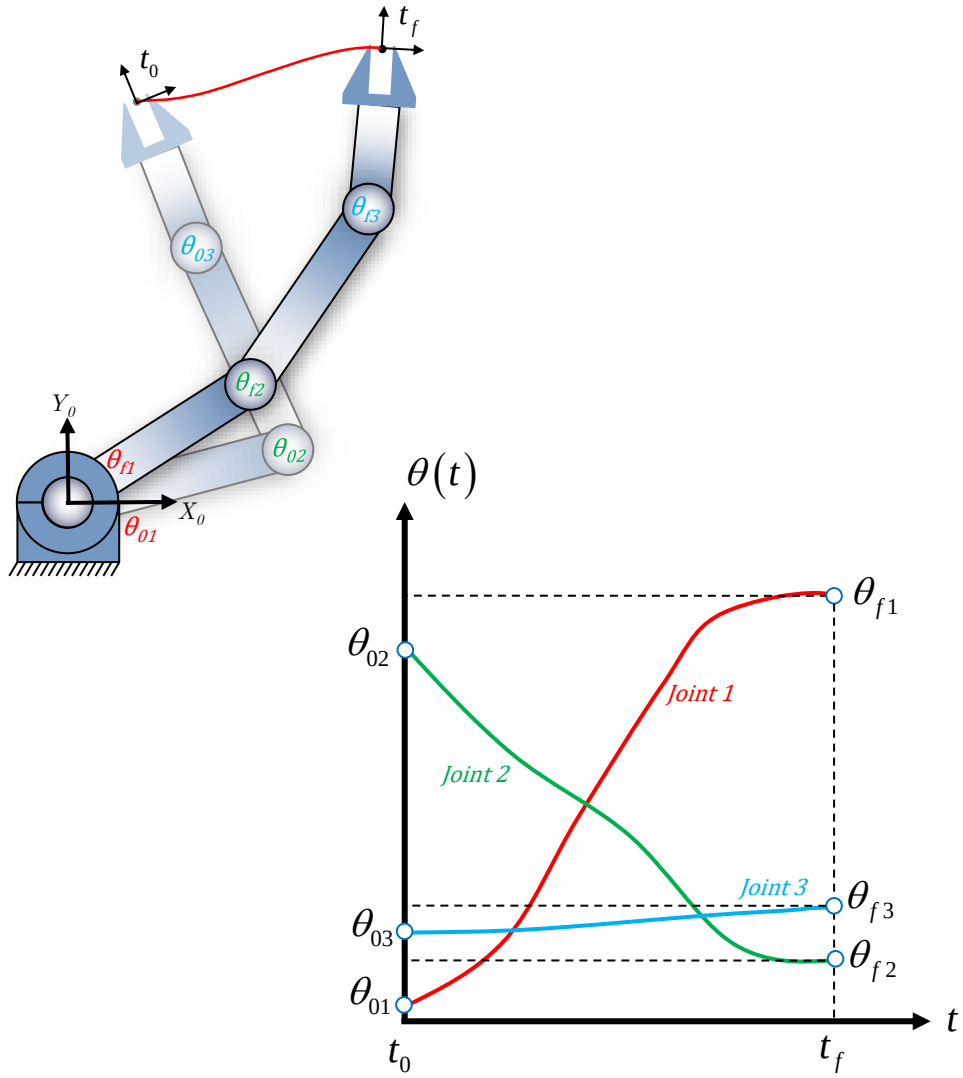
Robot motion typical planning steps



Joint Space Trajectories

Joint Space approach

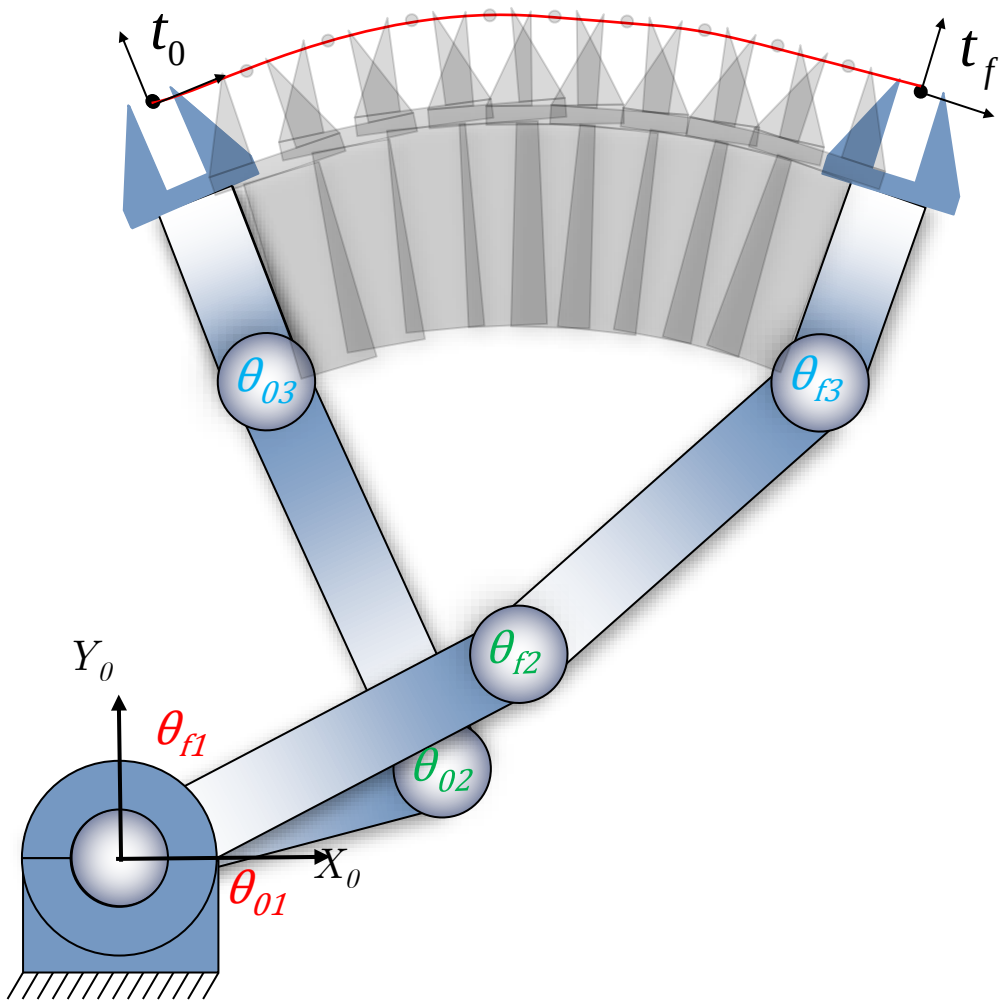
- The desirable initial and final poses of the end-effector are known;
- By using inverse kinematics, **only** initial and final joint positions (angles or displacements) are calculated;
- A **continuous and smooth polynomial function for each joint** is used to calculate the joint positions in between the known initial and final joint positions over a period of time
  - In other words, joint positions are interpolated in between;
  - Constraints and boundary conditions on joint position, velocity, acceleration can be imposed;
- **Note 1:** All joints reach their final position at the same time
- **Note 2:** The interpolated position of a joint, does not depend or consider the position of the other joints. May lead to relative joint position issues (e.g., joint limits).



Task Space Trajectories

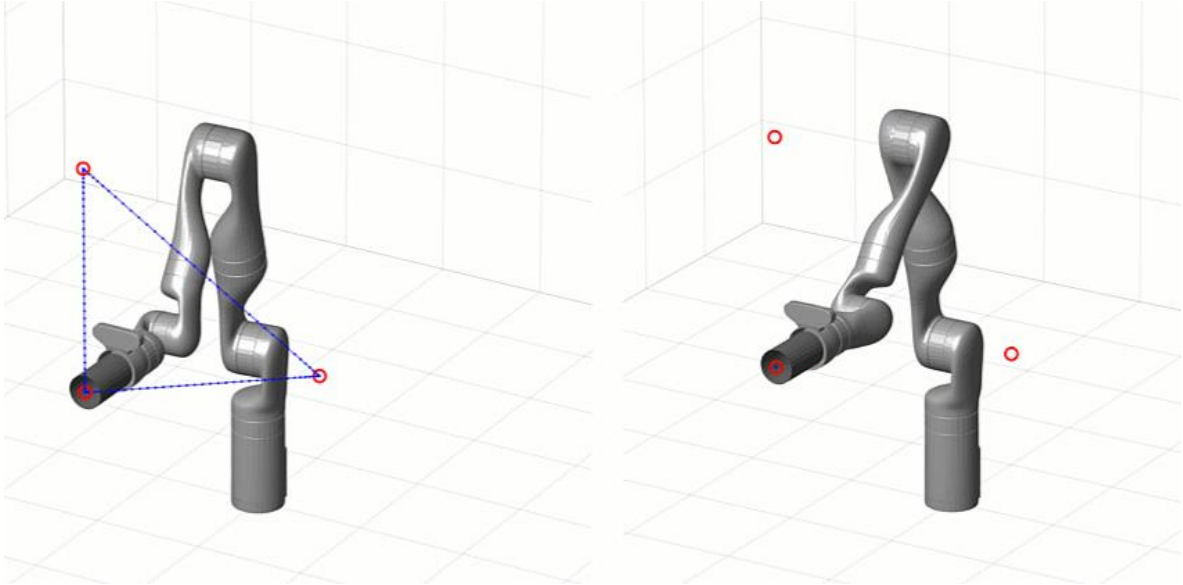
Task Space approach

- Considers directly the spatial position and orientation (pose) of the end-effector;
- Path shapes are expressed in terms of functions that define the end-effector pose as a function of time;
    - Similar polynomial functions as previously are used for interpolation of poses (rather than joint positions) along the defined path in task space
- At run-time, inverse kinematics need to be solved for every defined pose and as the path is updated;
    - I.e., joint positions are continuously calculated
- In practice it is not always possible to follow the path, due to robot workspace limits and singularities (i.e., unsolvable IK).
    - High joint rates near singularities can pose significant safety issues, especially for surgical robots



Joint Space vs. Task Space Trajectories: Pros and Cons

- In both task space or joint space trajectory generation, there are various ways to generate the trajectories
  - Based on interpolating end-effector pose or joint configurations over time
- Task-space trajectories tend to look more “natural” than joint-space trajectories
  - End effector is moving smoothly with respect to the environment even if the joints are not
- Task-space trajectories require solving inverse kinematics more often than a joint-space trajectory
  - Computationally expensive, especially if IK solver is based on optimization.



	Task Space	Joint Space
Pros	• Motion is predictable, as interpolation takes place in task space • Better handling of obstacles and collisions	• Faster execution, as IK evaluated only at waypoints • Joint motion is smooth and continuous and easy to validate
Cons	• Computationally expensive due to use of IK in every time step • Joint motion not necessarily smooth and more difficult to validate	• Intermediate/interpolated points not guaranteed to respect joint limits • Cannot deal with task space obstacles explicitly

From: Mathworks

Polynomial Joint-Space Method: Zero Start and Finish Joint Velocities

- Consider known initial and final poses of a manipulator at times t_0 and t_f, respectively;
- Inverse kinematics provide the robot's joint angles at both poses;
- We are looking for a function $\theta(t)$ for each joint, to smoothly move that joint from its initial position θ_0 at t_0 to its final position θ_f at t_f.

$$\begin{cases} \theta(0)=\theta_0 & \dot{\theta}(0)=0 \\ \theta(t_f)=\theta_f & \dot{\theta}(t_f)=0 \end{cases}$$

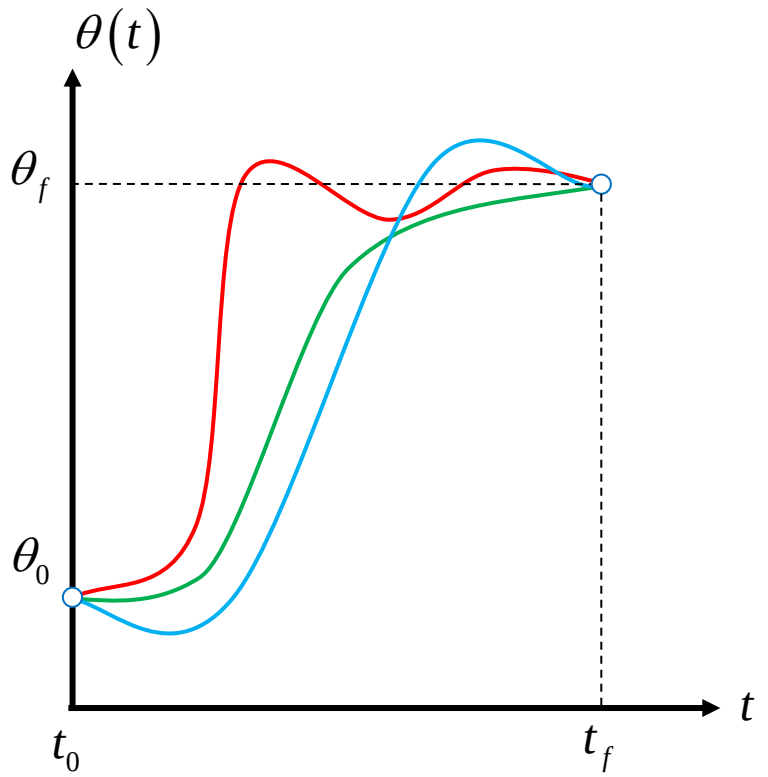
- Four boundary conditions (constraints) are set on $\theta(t)$

$$\begin{cases} \theta(t)=a_0+a_1t+a_2t^2+a_3t^3 \\ \dot{\theta}(t)=a_1+2a_2t+3a_3t^2 \\ \ddot{\theta}(t)=2a_2+6a_3t \end{cases}$$

- Which can be satisfied by a **cubic polynomial**, which has as many coefficients as the boundary conditions

$$\begin{cases} \theta(0)=a_0 \\ \theta(t_f)=a_0+a_1t_f+a_2t_f^2+a_3t_f^3 \\ \dot{\theta}(0)=a_1 \\ \dot{\theta}(t_f)=a_1+2a_2t_f+3a_3t_f^2 \end{cases}$$

$$\begin{cases} a_0=\theta_0 \\ a_1=0 \\ a_2=3(\theta_f-\theta_0)/t_f^2 \\ a_3=-2(\theta_f-\theta_0)/t_f^3 \end{cases}$$



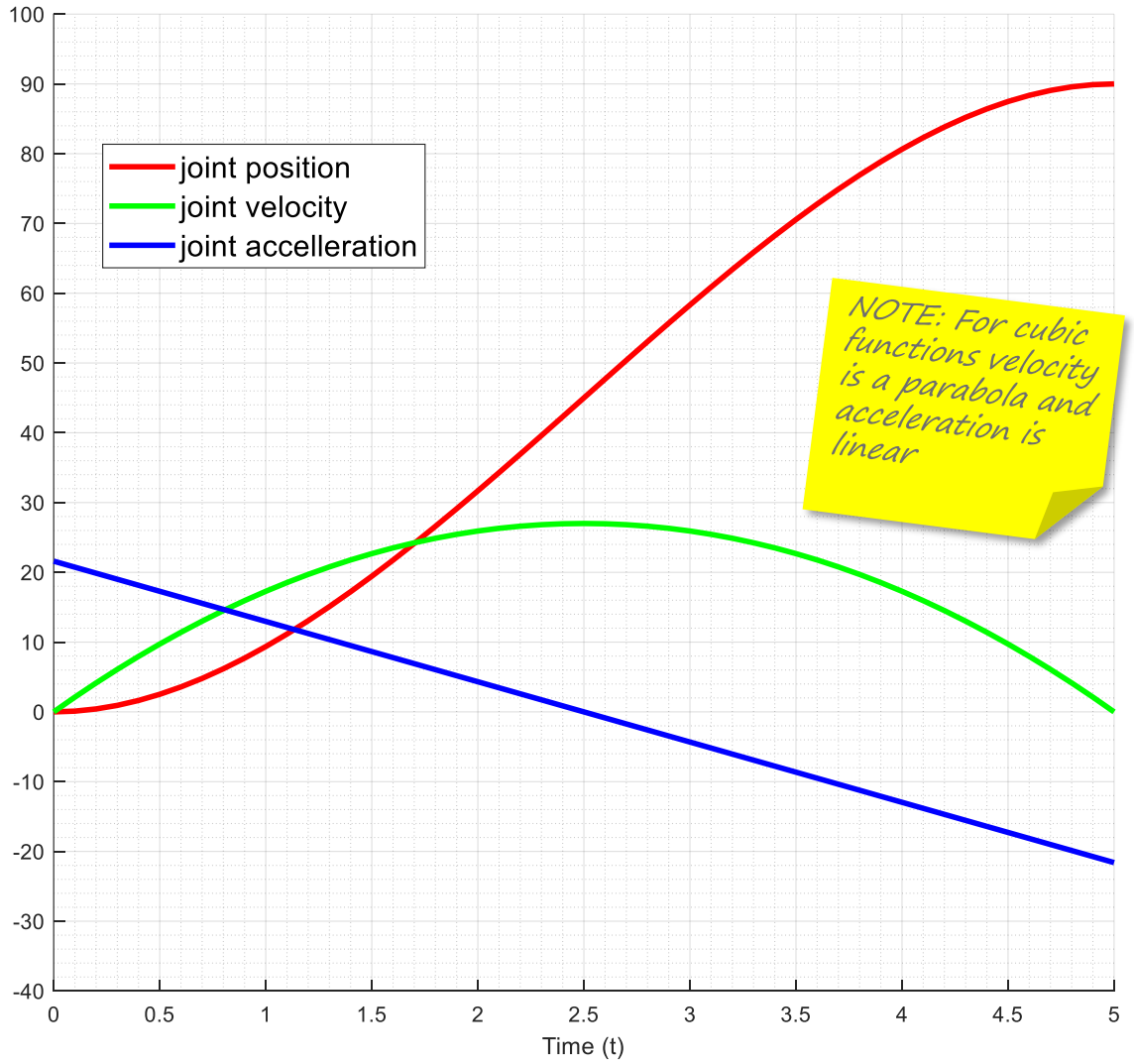
Polynomial Joint-Space Method: Zero Start and Finish Joint Velocities

MATLAB Code:

```
syms th0 thf a0 a1 a2 a3 tf th thdot thdotdot t

th0 = 0; %initial joint position
thf = 90; %final joint position
tf = 5; %motion duration

%Derived expressions of polynomial coefficients
a0 = th0;
a1 = 0;
a2 = 3*(thf-th0)/tf^2;
a3 = -2*(thf-th0)/tf^3;

%Substitute back into the cubic polynomial expression
th = a0 + a1*t + a2*t.^2 + a3*t.^3; %joint position
thdot = a1 + 2*a2*t + 3*a3*t.^2; %joint velocity
thdotdot = diff(th,t,2); %joint acceleration

%Evaluate symbolic expressions for t
t = [0:0.1:tf];
th = subs(th,t);
thdot = subs(thdot,t);
thdotdot = subs(thdotdot,t);

%Plot curves
hold on;
grid on;
grid minor;
plot(t,th,'Color','r','LineWidth',2.5);
plot(t,thdot,'Color','g','LineWidth',2.5);
plot(t,thdotdot,'Color','b','LineWidth',2.5);
```

$$\begin{cases} a_0 = \theta_0 \\ a_1 = 0 \\ a_2 = 3\left(\theta_f - \theta_0\right)/t_f^2 \\ a_3 = -2\left(\theta_f - \theta_0\right)/t_f^3 \end{cases}$$

$$\begin{cases} \theta(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 \\ \dot{\theta}(t) = a_1 + 2a_2 t + 3a_3 t^2 \\ \ddot{\theta}(t) = 2a_2 + 6a_3 t \end{cases}$$



Polynomial Joint-Space Method: Zero Start and Finish Joint Velocities and Acceleration

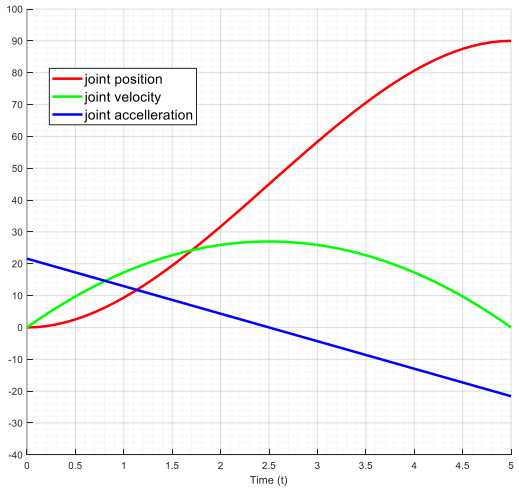
- In previous example we had four boundary conditions: desirable start/finish joint positions and zero start/finish velocities;
- We can include additional constraints, such as zero start/finish acceleration to achieve smoother motion

$$\begin{cases}\theta(0)=\theta_0\\ \theta(t_f)=\theta_f\end{cases}\quad\begin{cases}\dot\theta(0)=0\\ \dot\theta(t_f)=0\end{cases}\quad\begin{cases}\ddot\theta(0)=0\\ \ddot\theta(t_f)=0\end{cases}$$

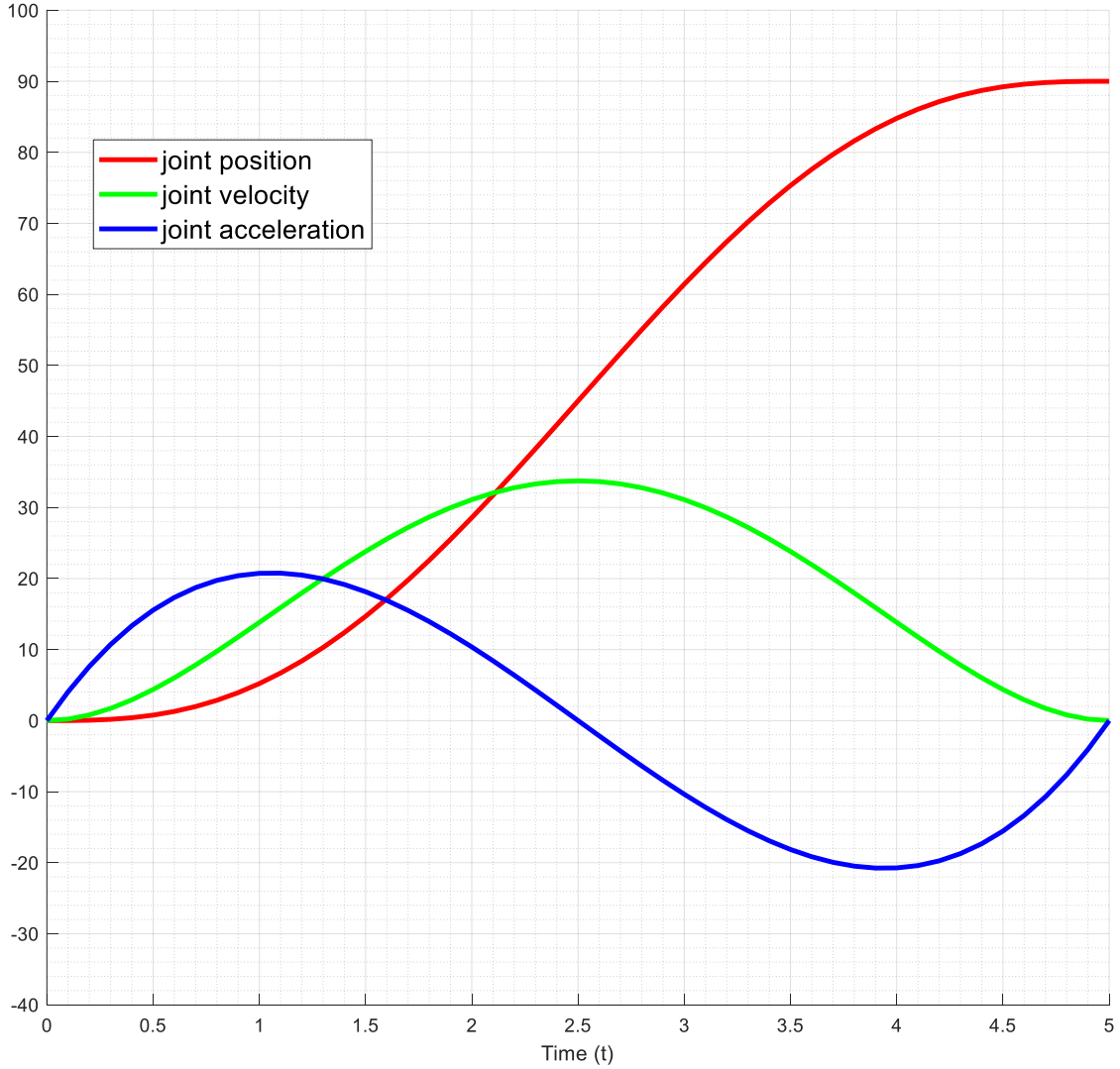
- For six constraints we need a **quintic polynomial** which has six coefficients

$$\begin{cases}\theta(t)=a_0+a_1t+a_2t^2+a_3t^3+a_4t^4+a_5t^5\\ \dot\theta(t)=a_1+2a_2t+3a_3t^2+4a_4t^3+5a_5t^4\\ \ddot\theta(t)=2a_2+6a_3t+12a_4t^2+20a_5t^3\end{cases}$$

- Like previously, we can derive six equations with six unknowns for coefficients a_0 to a_5



TODO: Derived in Tutorial problem-sheet Q1



Polynomial Joint-Space Method: Non-Zero Start and Finish Joint Velocities

- If joint velocity at start and finish points is non-zero, the boundary conditions are:

$$\begin{cases} \theta(0) = \theta_0 \\ \theta(t_f) = \theta_f \end{cases} \quad \begin{cases} \dot{\theta}(0) = \dot{\theta}_0 \\ \dot{\theta}(t_f) = \dot{\theta}_f \end{cases}$$

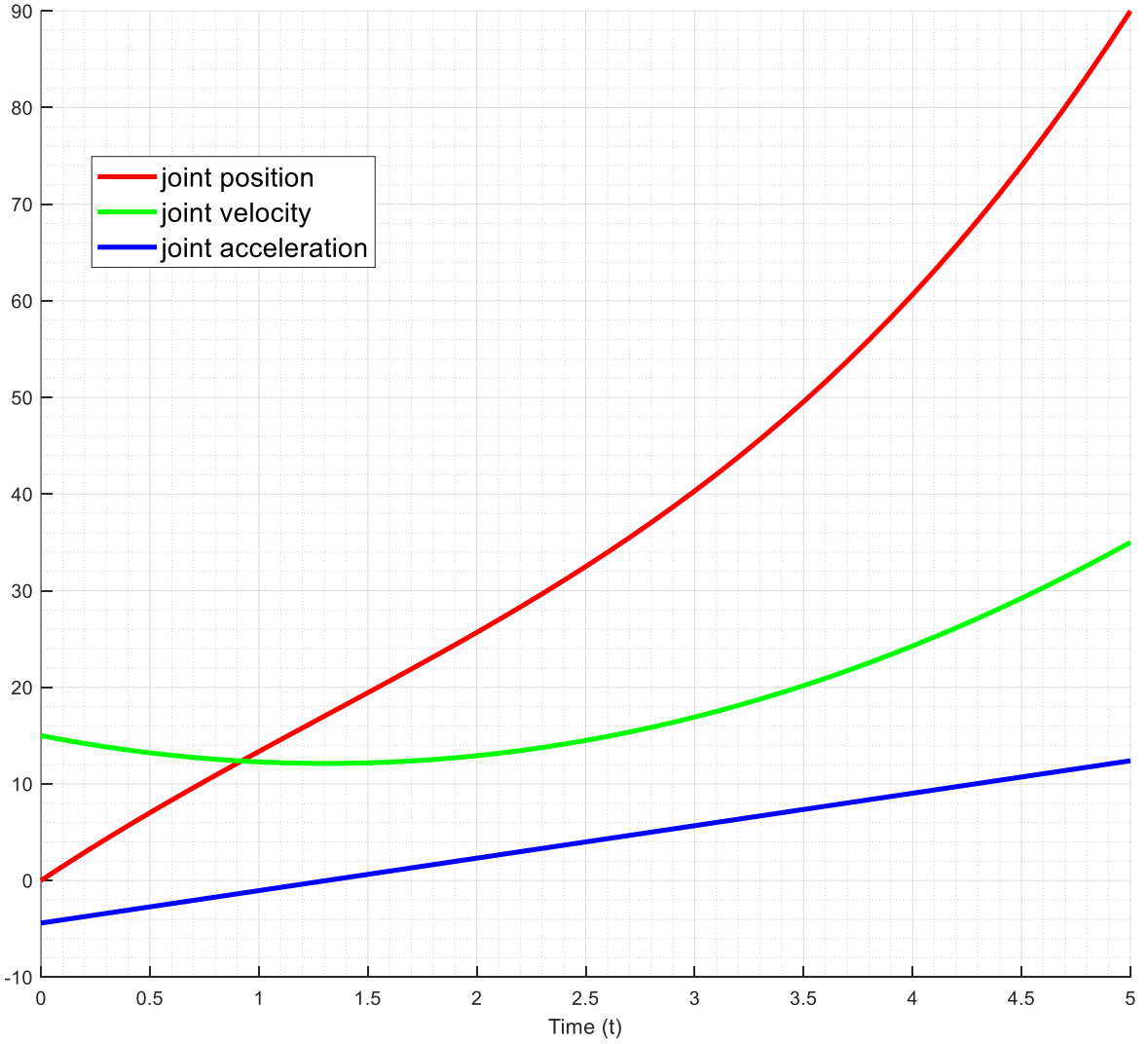
- A generic **cubic polynomial** will suffice for four boundary conditions;

$$\begin{cases} \theta(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 \\ \dot{\theta}(t) = a_1 + 2a_2 t + 3a_3 t^2 \\ \ddot{\theta}(t) = 2a_2 + 6a_3 t \end{cases}$$

- Solving the following equations we derive the polynomial coefficients that allow us to connect any initial and final positions with any initial and final velocities;

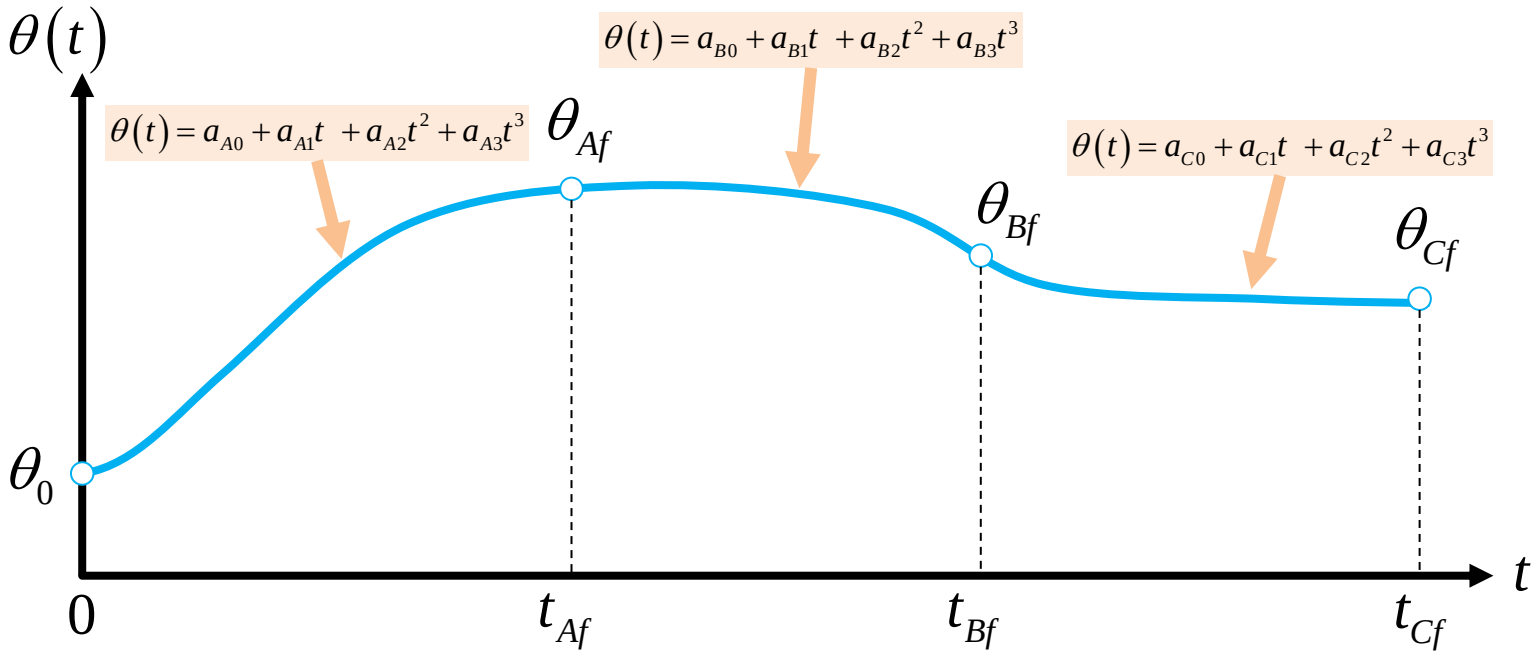
$$\begin{cases} \theta(0) = a_0 \\ \theta(t_f) = a_0 + a_1 t_f + a_2 t_f^2 + a_3 t_f^3 \\ \dot{\theta}(0) = a_1 \\ \dot{\theta}(t_f) = a_1 + 2a_2 t_f + 3a_3 t_f^2 \end{cases}$$

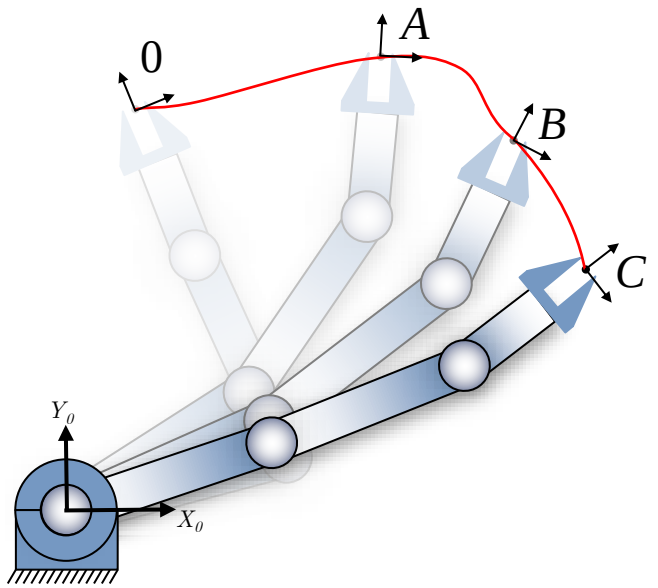
$$\begin{cases} a_0 = \theta_0 \\ a_1 = \dot{\theta}_0 \\ a_2 = 3(\theta_f - \theta_0)/t_f^2 - 2\dot{\theta}_0/t_f - \dot{\theta}_f/t_f \\ a_3 = -2(\theta_f - \theta_0)/t_f^3 + (\dot{\theta}_f + \dot{\theta}_0)/t_f^2 \end{cases}$$



Polynomial Joint-Space Method: Trajectory Planning using Waypoints

- Using a **suitable polynomial according to constraints**, we can generate longer trajectories by joining together several smaller ones connected at waypoints;
- Typically, we require traversing waypoints <u>smoothly</u> and <u>without stopping</u>;
- At each waypoint, inverse kinematics are used to calculate the required joint angles to be interpolated in between.



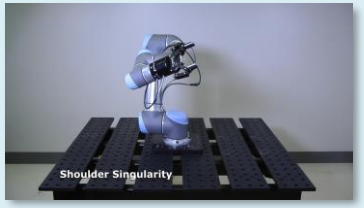


$$\begin{cases} \theta(0) = a_0 \\ \theta(t_f) = a_0 + a_1 t_f + a_2 t_f^2 + a_3 t_f^3 \\ \dot{\theta}(0) = a_1 \\ \dot{\theta}(t_f) = a_1 + 2a_2 t_f + 3a_3 t_f^2 \end{cases} \quad \begin{cases} a_0 = \theta_0 \\ a_1 = \dot{\theta}_0 \\ a_2 = 3(\theta_f - \theta_0)/t_f^2 - 2\dot{\theta}_0/t_f - \dot{\theta}_f/t_f \\ a_3 = -2(\theta_f - \theta_0)/t_f^3 + (\dot{\theta}_f + \dot{\theta}_0)/t_f^2 \end{cases}$$

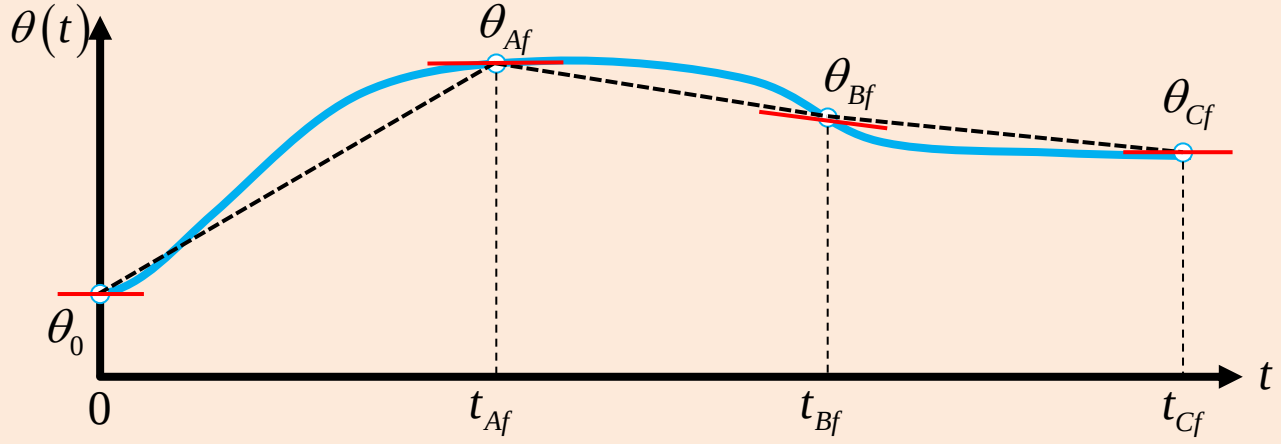
Polynomial Joint-Space Method: Smooth Trajectory Planning using Waypoints

- **Problem:** How can we specify joint velocity through each of the waypoints (as we don't want to stop)? There are different ways:

- **Way 1:** User-specified desired velocities at waypoints in task space (based on desired linear and angular velocity of end-effector)
  - Joint velocities are then calculated by the inverse of the Jacobian $\dot{\mathbf{P}} = \mathbf{J}(\vec{\theta})\dot{\vec{\theta}} \quad \Rightarrow \quad \dot{\vec{\theta}} = \mathbf{J}^{-1}(\vec{\theta})\dot{\mathbf{P}}$
  - Does not guarantee smooth joint motion due to discontinuities in acceleration (esp. close to singularities)
  - Cumbersome, as we must specify the velocity at every waypoint



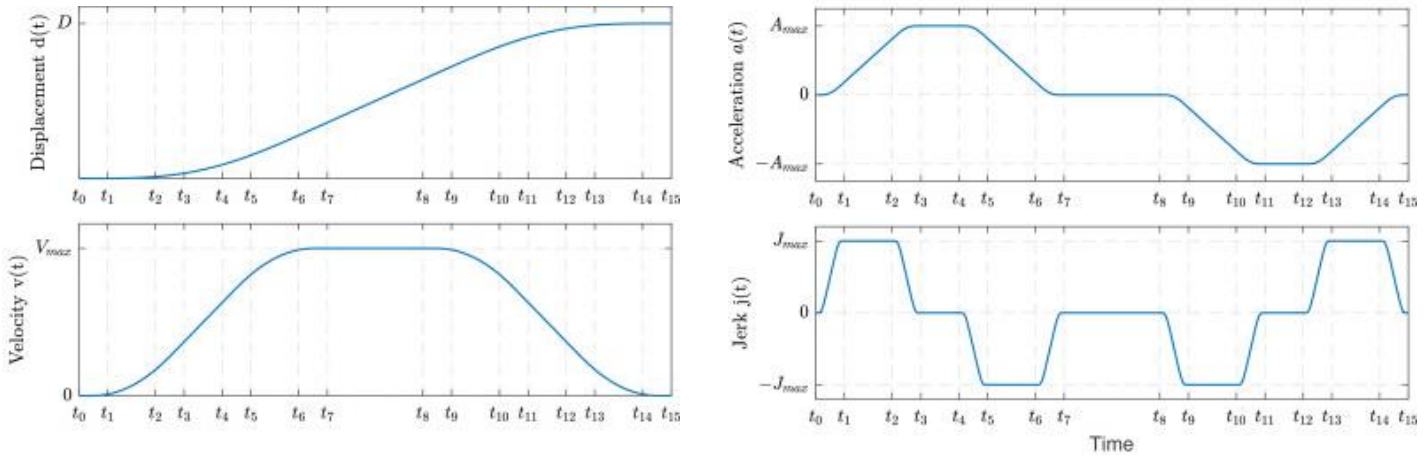
- **Way 2:** Automatically choose joint velocity based on some heuristic (a simple and practical) approach:
  - *Rule 1:* If slope of lines (first derivative of joint position, i.e., velocity) connecting the waypoints change sign, choose zero velocity
  - *Rule 2:* If slope does not change sign, choose average of two slopes as the velocity

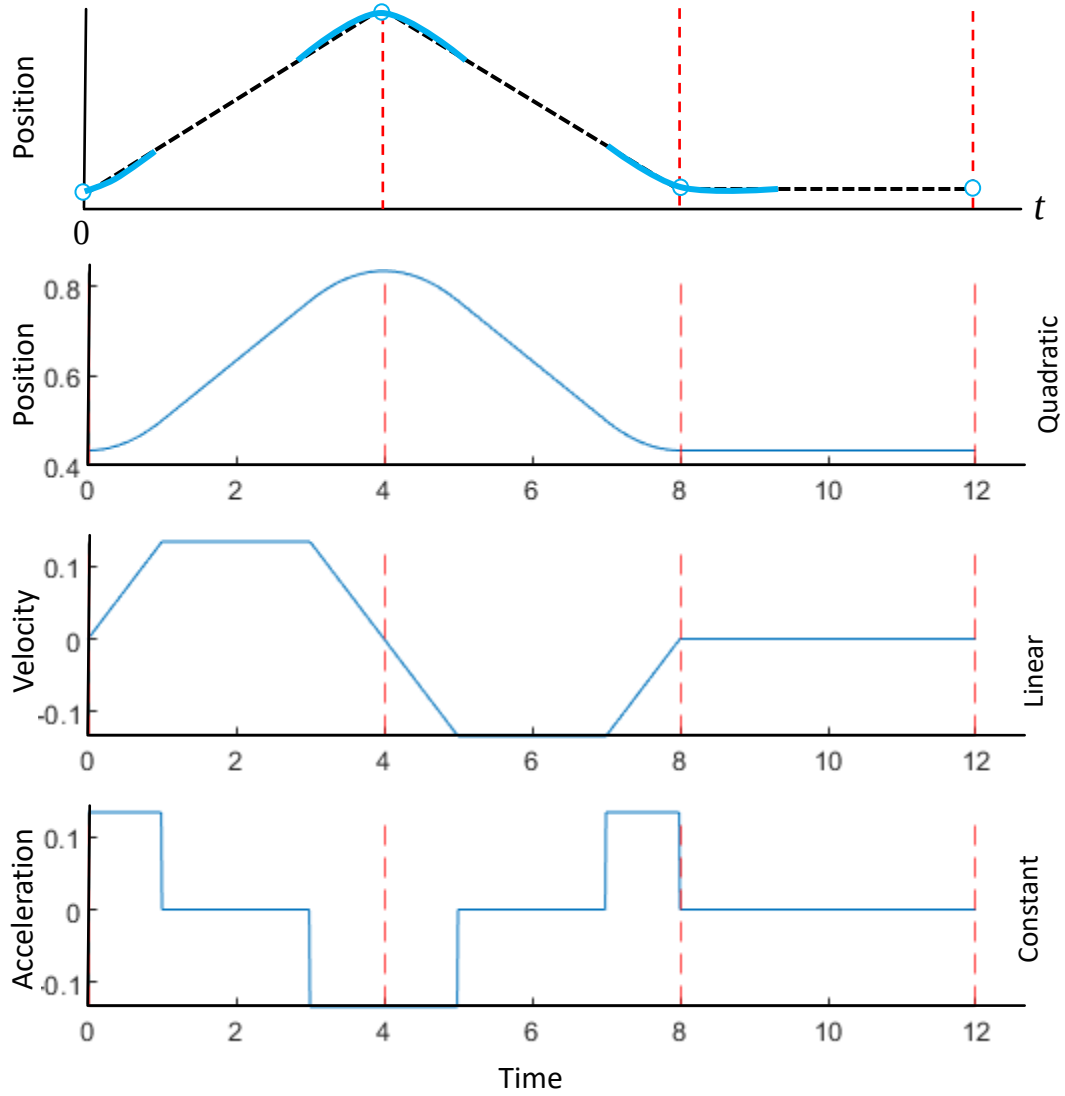


- **Way 3:** Automatically choose joint velocity, in such a way as to cause the joint acceleration (i.e., second derivative of joint position) at the waypoint to be continuous (no jump/jerk)
  - E.g.: enforce identical (but otherwise free) velocities and accelerations before and after the waypoints
  - Or: use higher order polynomials (e.g., quintic) and appropriate constraints
  - Or: Spline/blend the data portion connecting two polynomials connecting waypoints with polynomials continuous in velocity and/or acceleration (see next slide)

Other Ways to Generate Smooth Trajectories: Linear Segments with Parabolic Blending

- Use a linear function to interpolate between waypoints
  - Simple, but velocity would be discontinuous at the beginning and end of motion
- To create a smooth path in position and velocity, we blend/spline together the linear function with a parabolic (quadratic, 2nd order) region at each waypoint
  - velocity is continuous and linear (aka trapezoidal profile)
  - acceleration/deceleration is constant, which ensures smooth change in velocity from linear to blended regions (and vice versa)
- Acceleration is however still discontinuous
  - Alternative trajectory profiles may solve this, e.g., other S-curve profiles



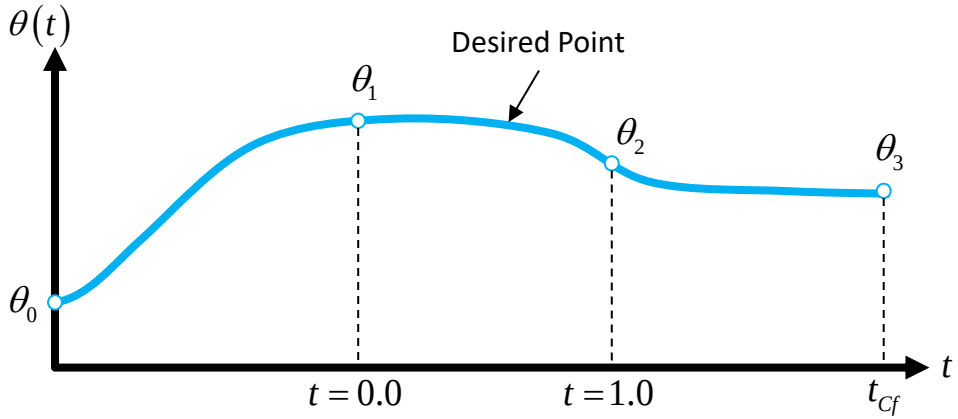


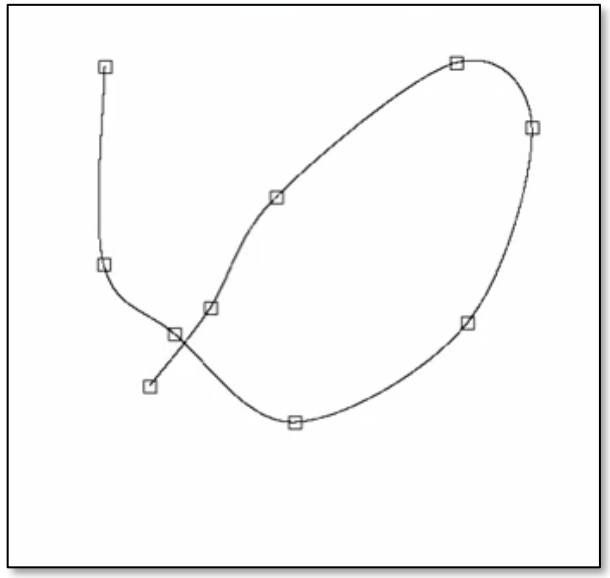
Other Ways to Generate Smooth Trajectories: Splines

- Splines are polynomials in space that can be used to create complex shapes through data interpolation and/or smoothing

Catmull-Rom Splines

- Are continuous in terms of position $\theta(t)$ and velocity $\dot{\theta}(t)$
- Ensure that the path goes through the control points (our trajectory waypoints)





$$\theta(t) = \frac{1}{2}\begin{bmatrix}1 & t & t^2 & t^3\end{bmatrix}\begin{bmatrix}0 & 2 & 0 & 0\\ -1 & 0 & 1 & 0\\ 2 & -5 & 4 & -1\\ -1 & 3 & -3 & 1\end{bmatrix}\begin{bmatrix}\theta_0\\ \theta_1\\ \theta_2\\ \theta_3\end{bmatrix}$$

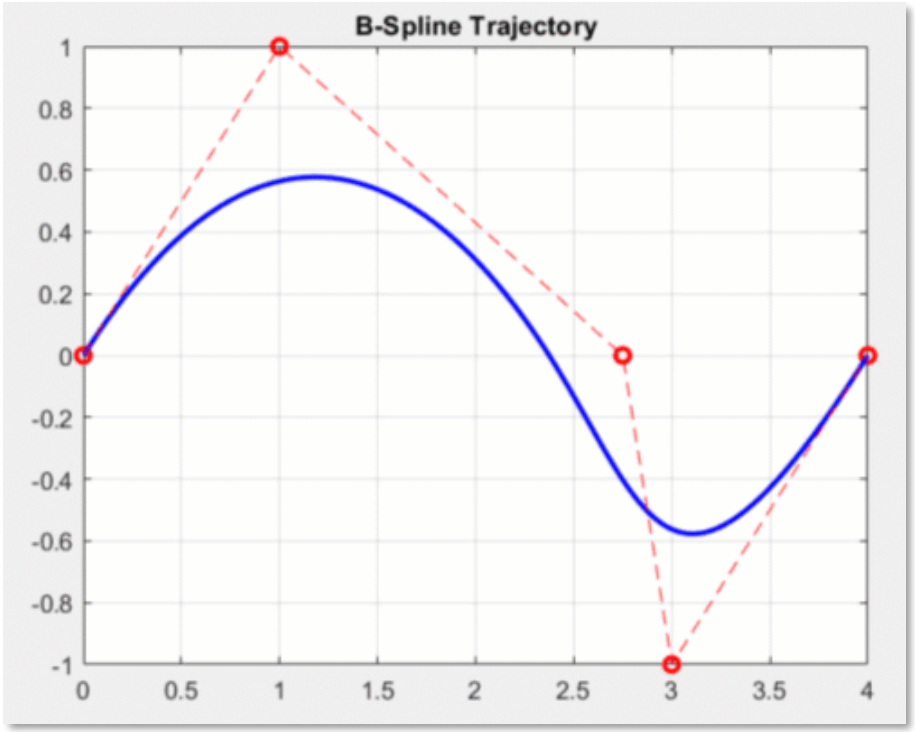
$$\begin{aligned}\theta(t) = 0.5*((&2\theta_1) +\\ &(-\theta_0 + \theta_2)t +\\ &(2\theta_0 - 5\theta_1 + 4\theta_2 - \theta_3)t^2\\ &(-\theta_0 + 3\theta_1 - 3\theta_2 + \theta_3)t^3)\end{aligned}$$

Other Ways to Generate Smooth Trajectories: Splines

B-Splines

- Short for Basis Spline
- Ensures continuity in terms of $\theta(t)$, $\dot{\theta}(t)$, $\ddot{\theta}(t)$
- Path will not pass through the control points

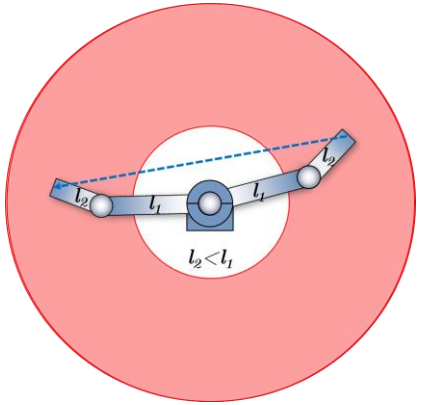
$$\theta(t) = \frac{1}{6}\begin{bmatrix}1 & t & t^2 & t^3\end{bmatrix}\begin{bmatrix}1 & 4 & 1 & 0\\ -3 & 0 & 3 & 0\\ 3 & -6 & 3 & 0\\ -1 & 3 & -3 & 1\end{bmatrix}\begin{bmatrix}\theta_0\\ \theta_1\\ \theta_2\\ \theta_3\end{bmatrix}$$

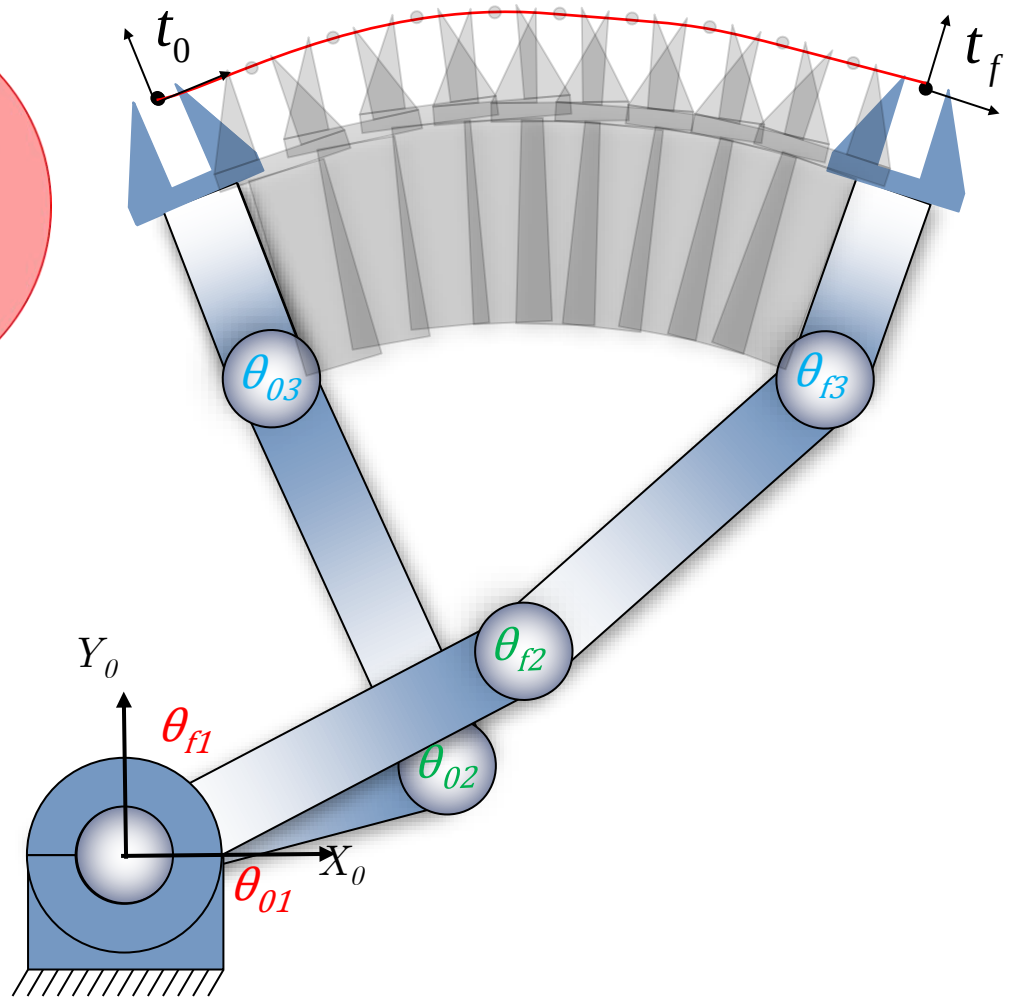


From: Mathworks

Task Space Trajectory Generation

- Considers directly the spatial position and orientation (pose) of the end-effector;
- The same interpolation, blending or spline fitting approaches may be used to the prescribed path;
- In practice it is not always possible to follow the path linearly, due to robot workspace limits or singularities.
- Also, we cannot interpolate the rotation matrix to calculate orientations in between waypoints
- Cartesian position is described with 3 numbers;
- The **angle-axis** representation can be used to specify an orientation with 3 numbers;
- Smoothly interpolate the 6x1 vector of Cartesian position and orientation between waypoints;
- Use inverse kinematics at each of these interpolated points;
- Easy to convert a rotation matrix to the angle-axis representation, and vice versa, e.g.:
  - MATLAB functions angvec2tr, tr2angvec





Conclusions

- Path and trajectory definitions
- Trajectory planning in robotics
- Joint-Space vs. Task-Space trajectory planning
- Polynomial Joint-Space Methods
- Linear Segments with Parabolic Blending
- Splines for smooth continuous trajectory planning
- Task Space interpolation using angle-axis notation