Robotics 1

Trajectory planning

Prof. Alessandro De Luca
Trajectory planner interfaces

robot action described as a sequence of poses or configurations (with possible exchange of contact forces)

reference profile/values (continuous or discrete) for the robot controller

* = programming “points”
Trajectory definition
a standard procedure for industrial robots

1. define Cartesian pose points (position+orientation) using the teach-box
2. program an (average) velocity between these points, as a 0-100% of a maximum system value (different for Cartesian- and joint-space motion)
3. linear interpolation in the joint space between points sampled from the built trajectory

examples of additional features
   a) over-fly
   b) sensor-driven STOP
   c) circular path through 3 points

main drawbacks
- semi-manual programming (as in “first generation” robot languages)
- limited visualization of motion

a mathematical formalization of trajectories is useful/needed
Some typical trajectories

- Point-to-point Cartesian motion with an **intermediate** point

Straight lines as Cartesian path

Interpolation with Bezier curves
Some typical trajectories

- Timing laws: Cartesian path with (dis-)continuous tangent

- Square path at constant speed
- Square path with trapezoidal speed profile
Joint and Cartesian trajectories

- assigned task: arm reconfiguration between two inverse kinematic solutions associated to a given end-effector pose

- initial and final configuration
- same Cartesian pose (no change!): the motion cannot be fully specified in the Cartesian space
- to perform this task, the robot should leave the given end-effector pose and then return to it
- a self-motion could be sufficient
  - if the robot starts in a singularity
  - if there is (task) redundancy (m<n)

for “simple” manipulators (e.g., all industrial robots) and m=n, the execution of these tasks will require the passage through a singular configuration
Joint and Cartesian trajectories

- a reconfiguration task (or... passing through singularity)

- three-phase trajectory:
  - circular path + self-motion + linear path

- single-phase trajectory in the joint space (no stops)
From task to trajectory

TRAJECTORY \| GEOMETRIC PATH + TIMING LAW

\{ \text{of motion } p_d(t) \}
\{ \text{of interaction } F_d(t) \}
\{ \text{parameterized by } s: p=p(s) \}
\{ \text{describes the time evolution of } s=s(t) \}

\[ p(s) = \begin{bmatrix} p_x(s) \\ p_y(s) \\ p_z(s) \end{bmatrix} \]

example: TASK planner provides A, B
TRAJECTORY planner generates p(t)
Trajectory planning
operative sequence

1. TASK planning
   - sequence of pose points ("knots") in Cartesian space
     - interpolation in Cartesian space
   - Cartesian geometric path (position + orientation): \( p = p(s) \)

2. Analytic inversion
   - path sampling and kinematic inversion
   - sequence of "knots" in joint space
     - interpolation in joint space
   - geometric path in joint space: \( q = q(\lambda) \)

Additional issues to be considered in the planning process
- obstacle avoidance
- on-line/off-line computational load
- sequence 2 is more "dense" than 1
Example

Cartesian space

joint space

Robotics 1
Trajectory classification

- space of definition
  - Cartesian, joint

- task type
  - point-to-point (PTP), multiple points (knots), continuous, concatenated

- path geometry
  - rectilinear, polynomial, exponential, cycloid, ...

- timing law
  - bang-bang in acceleration, trapezoidal in velocity, polynomial, ...

- coordinated or independent
  - motion of all joints (or of all Cartesian components) start and ends at the same instants (say, t=0 and t=T) = single timing law or
  - motions are timed independently (according to the requested displacement and robot capabilities) – mostly only in joint space
Path and timing law

- after choosing a path, the trajectory definition is completed by the choice of a timing law

\[ p = p(s) \Rightarrow s = s(t) \quad \text{(Cartesian space)} \]
\[ q = q(\lambda) \Rightarrow \lambda = \lambda(t) \quad \text{(joint space)} \]

- if \( s(t) = t \), path parameterization is the natural one given by time

- the timing law
  - is chosen based on task specifications (stop in a point, move at constant velocity, and so on)
  - may consider optimality criteria (min transfer time, min energy,...)
  - constraints are imposed by actuator capabilities (max torque, max velocity,...) and/or by the task (e.g., max acceleration on payload)

**Note:** on parameterized paths, a space-time decomposition takes place

\[ \dot{s}(t) = \frac{dp}{ds} \quad \ddot{s}(t) = \frac{d^2p}{ds^2} \]

e.g., in Cartesian space

- e.g., in Cartesian space
  \[ \dot{p}(t) = \frac{dp}{ds} \dot{s} \]
  \[ \ddot{p}(t) = \frac{d^2p}{ds^2} \dot{s} + \frac{dp}{ds} \ddot{s} \]
Cartesian vs. joint trajectory planning

- planning in **Cartesian space**
  - allows a more direct visualization of the generated path
  - obstacle avoidance, lack of “wandering”

- planning in **joint space**
  - does not need on-line kinematic inversion

- issues in kinematic inversion
  - \( \dot{q} \) e \( \ddot{q} \) (or higher-order derivatives) may also be needed
  - Cartesian task specifications involve the geometric path, but also bounds on the associated timing law
  - for redundant robots, choice among \( \infty^{n-m} \) inverse solutions, based on optimality criteria or additional auxiliary tasks
  - off-line planning in advance is not always feasible
    - e.g., when interaction with the environment occurs or sensor-based motion is needed
Relevant characteristics

- computational **efficiency** and **memory** space
  - e.g., store only the coefficients of a polynomial function
- **predictability** and **accuracy**
  - vs. “wandering” out of the knots
  - vs. “overshoot” on final position
- **flexibility**
  - allowing concatenation of primitive segments
  - over-fly
  - ...
- **continuity**
  - in space and/or in time
  - at least $C^1$, but also up to jerk = third derivative in time
A robot trajectory with bounded jerk

video

Robotics 1
Trajectory planning in joint space

- \( q = q(t) \) in time or \( q = q(\lambda) \) in space (then with \( \lambda = \lambda(t) \))
- it is sufficient to work component-wise (\( q_i \) in vector \( q \))
- an implicit definition of the trajectory, by solving a problem with specified boundary conditions in a given class of functions
- typical classes: polynomials (cubic, quintic,...), (co)sinusoids, clothoids, ...
- imposed conditions
  - passage through points = interpolation
  - initial, final, intermediate velocity (or geometric tangent for paths)
  - initial, final acceleration (or geometric curvature)
  - continuity up to the k-th order time (or space) derivative: class \( \mathcal{C}^k \)

many of the following methods and remarks can be directly applied also to Cartesian trajectory planning (and vice versa)!
Cubic polynomial in space

\[
q(\lambda) = q(0) + \Delta q [a \lambda^3 + b \lambda^2 + c \lambda + d]
\]

4 conditions

\[
\Delta q = q(1) - q(0)
\]
\[
\lambda \in [0,1]
\]

\[
q_N(0) = 0 \Leftrightarrow d = 0
\]
\[
q_N(1) = 1 \Leftrightarrow a + b + c = 1
\]
\[
q_N'(0) = \frac{dq_N}{d\lambda}|_{\lambda=0} = c = v_0/\Delta q
\]
\[
q_N'(1) = \frac{dq_N}{d\lambda}|_{\lambda=1} = 3a + 2b + c = v_1/\Delta q
\]

$doubly normalized$ polynomial $q_N(\lambda)$

4 coefficients

special case: $v_0 = v_1 = 0$ (zero tangent)

\[
q_N'(0) = 0 \Leftrightarrow c = 0
\]
\[
q_N(1) = 1 \Leftrightarrow a + b = 1
\]
\[
q_N'(1) = 0 \Leftrightarrow 3a + 2b = 0
\]

\[
a = -2
\]
\[
b = 3
\]
Cubic polynomial in time

\[ q(0) = q_{\text{in}} \quad q(T) = q_{\text{fin}} \quad \dot{q}(0) = v_{\text{in}} \quad \dot{q}(T) = v_{\text{fin}} \]

4 conditions

\[ q(\tau) = q_{\text{in}} + \Delta q \left[ a \tau^3 + b \tau^2 + c \tau + d \right] \]

\[ \Delta q = q_{\text{fin}} - q_{\text{in}} \quad \tau = t/T, \quad \tau \in [0,1] \]

4 coefficients \rightarrow “doubly normalized” polynomial \( q_N(\tau) \)

\[ q_N(0) = 0 \iff d = 0 \quad q_N(1) = 1 \iff a + b + c = 1 \]

\[ q_N'(0) = \left. \frac{d q_N}{d \tau} \right|_{\tau=0} = c = v_{\text{in}} T/\Delta q \quad q_N'(1) = \left. \frac{d q_N}{d \tau} \right|_{\tau=1} = 3a + 2b + c = v_{\text{fin}} T/\Delta q \]

special case: \( v_{\text{in}} = v_{\text{fin}} = 0 \) (rest-to-rest)

\[ q_N'(0) = 0 \iff c = 0 \]

\[ q_N(1) = 1 \iff a + b = 1 \]

\[ q_N'(1) = 0 \iff 3a + 2b = 0 \]

\[ \iff a = -2 \quad b = 3 \]
Quintic polynomial

\[ q(\tau) = a\tau^5 + b\tau^4 + c\tau^3 + d\tau^2 + e\tau + f \]

\[ \tau \in [0, 1] \]

allows to satisfy 6 conditions, for example (in normalized time \( \tau = t/T \))

\[ q(0) = q_0 \quad q(1) = q_1 \quad q'(0) = v_0 T \quad q'(1) = v_1 T \quad q''(0) = a_0 T^2 \quad q''(1) = a_1 T^2 \]

\[ q(\tau) = (1 - \tau)^3[q_0 + (3q_0 + v_0 T)\tau + (a_0 T^2 + 6v_0 T + 12q_0)\tau^2/2] + \tau^3 \left[q_1 + (3q_1 - v_1 T)(1 - \tau) + (a_1 T^2 - 6v_1 T + 12q_1)(1 - \tau)^2/2\right] \]

special case: \( v_0 = v_1 = a_0 = a_1 = 0 \)

\[ q(\tau) = q_0 + \Delta q \left[6\tau^5 - 15\tau^4 + 10\tau^3\right] \quad \Delta q = q_1 - q_0 \]
Higher-order polynomials

- a suitable solution class for satisfying symmetric boundary conditions (in a PTP motion) that impose zero values on higher-order derivatives
  - the interpolating polynomial is always of odd degree
  - the coefficients of such (doubly normalized) polynomial are always integers, alternate in sign, sum up to unity, and are zero for all terms up to the power = (degree-1)/2
- in all other cases (e.g., for interpolating a large number N of points), their use is not recommended
  - N-th order polynomials have N-1 maximum and minimum points
  - oscillations arise out of the interpolation points (wandering)
Numerical examples

9th degree

4 derivatives are zero

14 derivatives are zero!

29th degree

no overshoot nor wandering

2.5

normalized first derivative (velocity in time)

4.5!!

peaking at midpoint
4-3-4 polynomials

three phases (Lift off, Travel, Set down) in a pick-and-place operation in time

q(t) = 4th order polynomial
q_T(t) = 3rd order polynomial
q_S(t) = 4th order polynomial

14 coefficients

boundary conditions

q(t_0) = q_0  \quad q(t_1^-) = q(t_1^+) = q_1  \quad q(t_2^-) = q(t_2^+) = q_2  \quad q(t_f) = q_f

\dot{q}(t_0) = \dot{q}(t_f) = 0  \quad \ddot{q}(t_0) = \ddot{q}(t_f) = 0

\dot{q}(t_i^-) = \dot{q}(t_i^+)  \quad \ddot{q}(t_i^-) = \ddot{q}(t_i^+)  \quad i = 1,2

Robotics 1
Interpolation using splines

- **problem**
  - interpolate N knots, with continuity up to the second derivative

- **solution**
  - *spline*: N-1 cubic polynomials, concatenated so as to pass through N knots and being continuous up to the second derivative at the N-2 internal knots
  - 4(N-1) coefficients
  - 4(N-1)-2 conditions, or
    - 2(N-1) of passage (for each cubic, in the two knots at its ends)
    - N-2 of continuity for first derivative (at the internal knots)
    - N-2 of continuity for second derivative (at the internal knots)
  - 2 free parameters are still left over
    - can be used, e.g., to assign initial and final derivatives, \(v_1\) and \(v_N\)
  - presented next in terms of *time* \(t\), but similar in terms of *space* \(\lambda\)
    - *then*: first derivative = velocity, second derivative = acceleration
Building a cubic spline

\[ q = \theta(t) = \{\theta_k(t), \ t \in [t_k, t_{k+1}]\} \]

\[ \theta_k(\tau) = a_{k0} + a_{k1} \tau + a_{k2} \tau^2 + a_{k3} \tau^3 \]

\[ \tau \in [0, h_k], \ \tau = t - t_k \quad (k = 1, \ldots, N-1) \]

continuity conditions for velocity and acceleration

\[ \dot{\theta}_k(h_k) = \dot{\theta}_{k+1}(0) \]
\[ \ddot{\theta}_k(h_k) = \ddot{\theta}_{k+1}(0) \quad k = 1, \ldots, N-2 \]

Robotics 1
An efficient algorithm

1. if all velocities $v_k$ at internal knots were known, then each cubic in the spline would be uniquely determined by

   $\theta_k(0) = q_k = a_k^0$

   $\dot{\theta}_k(0) = v_k = a_k^1$

   

   \[
   \begin{pmatrix}
   h_k^2 & h_k^3 \\
   2h_k & 3h_k^2 \\
   a_k^2 & a_k^3
   \end{pmatrix}
   \begin{pmatrix}
   q_{k+1} - q_k - v_k h_k \\
   v_{k+1} - v_k
   \end{pmatrix} = 1
   \]

2. impose the continuity for accelerations (N-2 conditions)

   \[\ddot{\theta}_k(h_k) = 2a_k^2 + 6a_k^3 h_k = \ddot{\theta}_{k+1}(0) = 2a_{k+1,2}\]

1. expressing the coefficients $a_{k2}$, $a_{k3}$, $a_{k+1,2}$ in terms of the still unknown knot velocities (see step 1.) yields a linear system of equations that is always (easily) solvable

   \[
   \begin{pmatrix}
   A(h) \\
   \uparrow \text{tri-diagonal} \quad \text{always invertible}
   \end{pmatrix}
   \begin{pmatrix}
   v_2 \\
   v_3 \\
   \vdots \\
   v_{N-1}
   \end{pmatrix}
   =
   \begin{pmatrix}
   b(h,q,v_1,v_N)
   \end{pmatrix}
   \]

   unknown

   known vector

   to be substituted then back in 1
Structure of $A(h)$

diagonally dominant matrix (for $h_k > 0$)
[the same matrix for all joints]
Structure of $b(h,q,v_1,v_N)$

\[
\begin{align*}
\frac{3}{h_1h_2} [h_1^2(q_3 - q_2) &+ h_2^2(q_2 - q_1)] - h_2v_1 \\
\frac{3}{h_2h_3} [h_2^2(q_4 - q_3) &+ h_3^2(q_3 - q_2)] \\
\quad &\vdots \\
\frac{3}{h_{N-3}h_{N-2}} [h_{N-3}^2(q_{N-1} - q_{N-2}) &+ h_{N-2}^2(q_{N-2} - q_{N-3})] \\
\frac{3}{h_{N-2}h_{N-1}} [h_{N-2}^2(q_N - q_{N-1}) &+ h_{N-1}^2(q_{N-1} - q_{N-2})] - h_{N-2}v_N
\end{align*}
\]
Properties of splines

- A spline (in space) is the solution with minimum curvature among all interpolating functions having continuous second derivative.
- For cyclic tasks \( q_1 = q_N \), it is preferable to simply impose continuity of first and second derivatives (i.e., velocity and acceleration in time) at the first/last knot as “squaring” conditions.
  - Choosing \( v_1 = v_N = v \) (for a given \( v \)) doesn’t guarantee in general the continuity up to the second derivative (in time, of the acceleration).
  - In this way, the first = last knot will be handled as all other internal knots.
- A spline is uniquely determined from the set of data \( q_1, ..., q_N, h_1, ..., h_{N-1}, v_1, v_N \).
- In time, the total motion occurs in \( T = \sum_k h_k = t_N - t_1 \).
- The time intervals \( h_k \) can be chosen so as to minimize \( T \) (linear objective function) under (nonlinear) bounds on velocity and acceleration in \([0,T]\).
- In time, the spline construction can be suitably modified when the acceleration is also assigned at the initial and final knots.
A modification handling assigned initial and final accelerations

- two more parameters are needed in order to impose also the initial acceleration $a_1$ and final acceleration $a_N$
- two “fictitious knots” are inserted in the first and last original intervals, increasing the number of cubic polynomials from N-1 to N+1
- in these two knots only continuity conditions on position, velocity and acceleration are imposed
  \[ \Rightarrow \text{two free parameters are left over (one in the first cubic and the other in the last cubic), which are used to satisfy the boundary conditions on acceleration} \]
- depending on the (time) placement of the two additional knots, the resulting spline changes
A numerical example

- $N = 4$ knots (3 cubic polynomials)
  - joint values $q_1 = 0$, $q_2 = 2\pi$, $q_3 = \pi/2$, $q_4 = \pi$
  - at $t_1 = 0$, $t_2 = 2$, $t_3 = 3$, $t_4 = 5$ (thus, $h_1 = 2$, $h_2 = 1$, $h_3 = 2$)
  - boundary velocities $v_1 = v_4 = 0$
- 2 added knots to impose accelerations at both ends (5 cubic polynomials)
  - boundary accelerations $\alpha_1 = \alpha_4 = 0$
  - two placements: at $t_1' = 0.5$ and $t_4' = 4.5$ (×), or $t_1'' = 1.5$ and $t_4'' = 3.5$ (*)

---

Robotics 1