MECEE 4602: Introduction to Robotics, Fall 2019

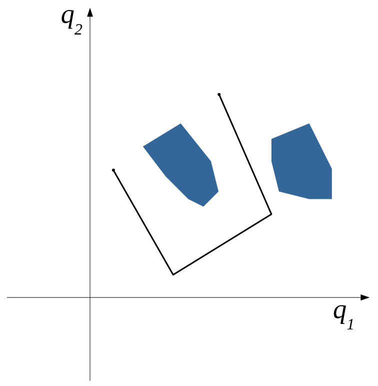
Trajectory Execution

In the previous lecture, we've looked at the problem of motion planning, or generating trajectories. In most cases, the trajectories we computed were a succession of points in the robot's configuration space that the robot must go through. In this lecture, we look in more detail at how we **execute** those trajectories.

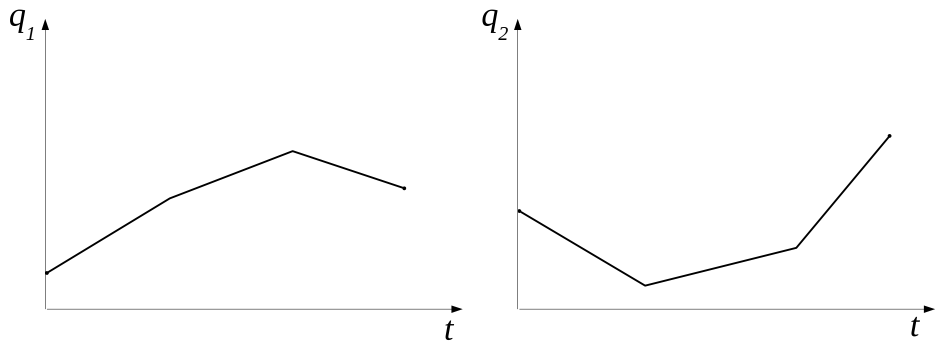
At this point, we also start explicitly considering time in our handling of robots. In general, a trajectory needs some finite time for execution; joints also have real-life constraints, such as a maximum velocity they can move at, a maximum acceleration, etc. All these constraints must be taken into account.

1 Timing laws for trajectory segments

Here is an example for a 2D configuration space, with a trajectory to be executed.



The configuration space plot is intuitive for representing the trajectory, but the time element is completely missing. A representation of the trajectory that also captures the time element would be:

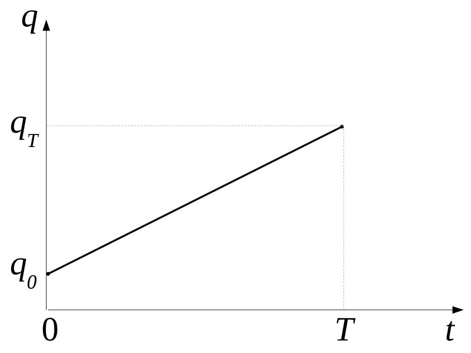


We see that, for each segment, each DOF value evolves as a function of time. The function will also depend on the DOF value that we request for the beginning of the segment (q_0 and the end of the segment (q_T), and can also depend on other constraints which we will use later.

$$q = q(t, q_0, q_T, \ldots) \tag{1}$$

This function is sometimes referred to as the **timing function** of the trajectory. Even if the waypoints suffice to plot a trajectory, it can not be executed until we also specify a particular timing function to be used.

Consider a single trajectory segment for a single joint:



The plot we have here illustrates the case where the timing function is simply linear interpolation between q_0 and q_T:

$$q = \frac{T-t}{T} q_0 + \frac{t}{T} q_T = \frac{q_T - q_0}{T} t + q_0 \tag{2}$$

We notice that in this case velocity ($\dot{q}$) is constant, and acceleration ($\ddot{q}$) is zero:

$$\dot{q} = \frac{q_T - q_0}{T} \tag{3}$$

$$\ddot{q} = 0 \tag{4}$$

Linear interpolation has severe shortcomings in practice. What if, at the start of the segment, the joint has some other velocity? For example, we might start with the joint standing still ($\dot{q}_0 = 0$). Also, what if we would like to end the segment with some desired velocity? For example, we might want the joint to reach a stop at the end ($\dot{q}_T = 0$). Linear interpolation assumes the joint can instantly achieve the specified velocity at the beginning and end of the trajectory, which is never feasible in practice (as it would have to assume infinite acceleration).

To formalize, linear interpolation provides continuous position, but discontinuous velocity and acceleration. In other words, it is C^0-continuous. Therefore, it can not match any specified velocity and acceleration at its endpoints, and can not "fit" with a segment coming before or after it.

Real robots with finite acceleration capabilities will require at least continuous velocity profiles, hence timing functions that are at least C^1 continuous. For even smoother applications, we might sometimes require C^2 continuity (continuous acceleration) or maybe even C^3 continuity (continuous jerk).

2 Polynomial interpolation

2.1 Cubic polynomials

If we'd like to be able to specify desired $\dot{q}_0$ and $\dot{q}_T$ we must choose a different function for q. One possibility is a polynomial. For specifying start and end velocities, a cubic suffices:

$$\begin{aligned} q(t) &= at^3 + bt^2 + ct + d && (5)\\ \dot{q}(t) &= 3at^2 + 2bt + c && (6)\\ \ddot{q}(t) &= 6at + 2b && (7) \end{aligned}$$

In order to compute the values for a, b, c and d, we use the desired constraints:

$$\begin{aligned} q_0 &= q(0) = d && (8)\\ q_T &= q(T) = aT^3 + bT^2 + cT + d && (9)\\ \dot{q}_0 &= \dot{q}(0) = c && (10)\\ \dot{q}_T &= \dot{q}(T) = 3aT^2 + 2bT + c && (11) \end{aligned}$$

Or, in matrix form:

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ T^3 & T^2 & T & 1 \\ 0 & 0 & 1 & 0 \\ 3T^2 & 2T & 1 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} q_0 \\ q_T \\ \dot{q}_0 \\ \dot{q}_T \end{bmatrix} \qquad (12)$$

which we can solve for $[a, b, c, d]^T$.

Example. Fit the cubic polynomial that achieves $q_0 = 1rad$, $q_T = 3rad$, $\dot{q}_0 = 0rad/s$ (start from rest), $\dot{q}_T = 1rad/s$, for a time interval $T = 1s$. Compute the position, velocity and acceleration at $t = 0s$, $t = 0.5s$ and $t = 1s$. Plot position, velocity and acceleration as a function of time for the entire trajectory.

2.2 Quintic polynomials

If, in addition, we'd like to specify desired endpoint accelerations $\ddot{q}_0$ and $\ddot{q}_T$, we must move to a quintic polynomial:

$$\begin{aligned} q(t) &= at^5 + bt^4 + ct^3 + dt^2 + et + f && (13) \\ \dot{q}(t) &= 5at^4 + 4bt^3 + 3ct^2 + 2dt + e && (14) \\ \ddot{q}(t) &= 20at^3 + 12bt^2 + 6ct + 2d && (15) \end{aligned}$$

The six constraints are:

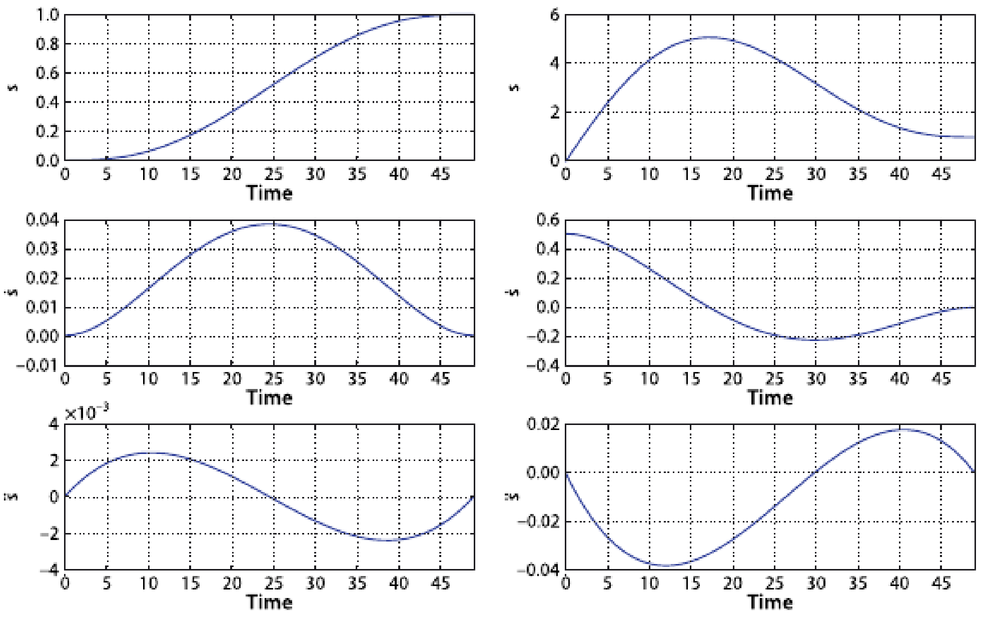
$$\begin{aligned} q_0 &= q(0) = f && (16) \\ q_T &= q(T) = aT^5 + bT^4 + cT^3 + dT^2 + eT + f && (17) \\ \dot{q}_0 &= \dot{q}(0) = e && (18) \\ \dot{q}_T &= \dot{q}(T) = 5aT^4 + 4bT^3 + 3cT^2 + 2dT + e && (19) \\ \ddot{q}_0 &= \ddot{q}(0) = 2d && (20) \\ \ddot{q}_T &= \ddot{q}(T) = 20aT^3 + 12bT^2 + 6cT + 2d && (21) \end{aligned}$$

or, in matrix form:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 & \\ T^5 & T^4 & T^3 & T^2 & T & 1 \\ 0 & 0 & 0 & 1 & 0 & \\ 5T^4 & 4T^3 & 3T^2 & 2T & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & \\ 20T^3 & 12T^2 & 6T & 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix} = \begin{bmatrix} q_0 \\ q_T \\ \dot{q}_0 \\ \dot{q}_T \\ \ddot{q}_0 \\ \ddot{q}_T \end{bmatrix} \qquad (22)$$

which we can again solve for $[a, b, c, d, e, f]^T$.

Here are example quintic trajectories (from Peter Corke, "Robotics, Vision and Control"):

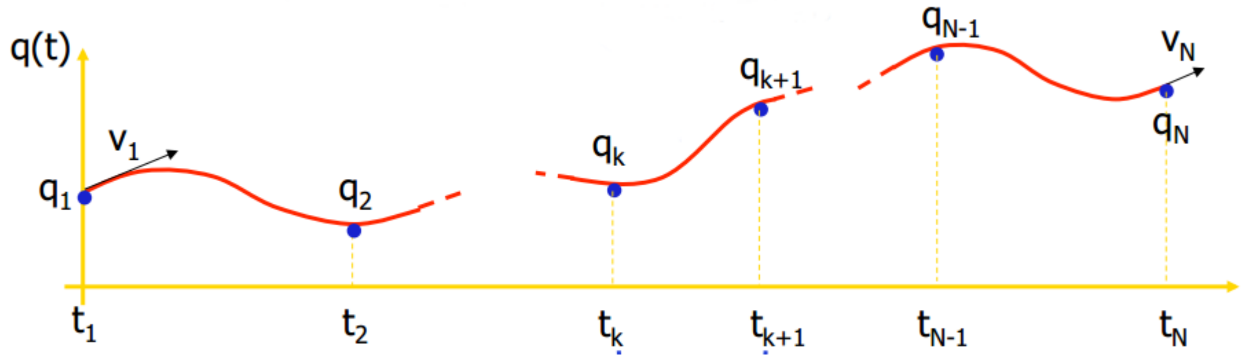


2.3 Multi-segment polynomial interpolation

Going back to our original motion planning problem, we notice that the overall joint trajectory the robot is executing has multiple segments. This means that each joint must execute multiple trajectory segments in succession. We will now take advantage of the ability of polynomial interpolation to hit specified starting and ending velocities (and accelerations) in order to compute a multi-segment trajectory.

Let's assume our goal is to hit all the waypoints with a continuous velocity profile. We could attempt to fit a polynomial to the trajectory (as we did in the single-segment case earlier), but the resulting polynomial would be of very high order.

One solution is to fit a separate low-order polynomial to each segment, and constrain these polynomials so that velocities (and possibly accelerations) at the waypoints are continuous (image from class notes by Alessandro de Luca):



Assume a total of N waypoints, connected by $N-1$ segments. If we fit a cubic polynomial to each segment, we have a total of $4N-4$ coefficients to play with. We can impose the following constraints:

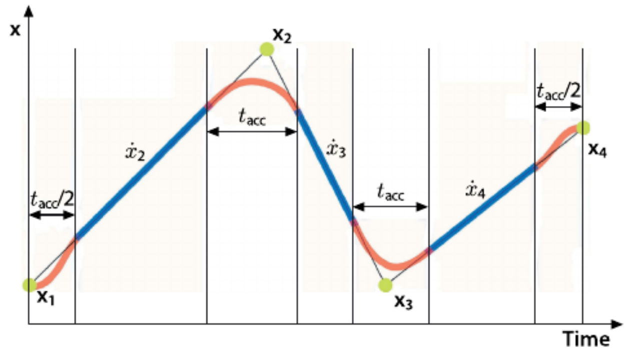
- each segment must start and end at a waypoint ($2N-2$) constraints)
- velocities at all internal waypoints must be continuous ($N-2$ constraints)
- accelerations at all internal waypoints must be continuous ($N-2$ constraints)

We thus have a total of $4N-6$ constraints, leaving us with 2 free coefficients. We can add 2 more constrains (e.g. specify desired initial and final velocity) in order to have a fully determined system.

Example. Assume the trajectory for a joint comprises 3 waypoints: $q_0 = 0rad$, $q_1 = 3rad$, $q_2 = 2rad$. The time spent between each waypoint must be $1s$. We would like to start and stop at rest $\dot{q}_0 = \dot{q}_2 = 0rad/s$. For each segment, the timing function must be a cubic polynomial. Write down the matrix equation that determines the coefficients of these polynomials.

2.4 Blending

The disadvantages of splines are similar to those of polynomials for single-segment execution. While they are continuous and hit all waypoints exactly, they allow little control over the velocity and acceleration used between them. One solution is to divide the trajectory into a set of constant-velocity motions and a set of blending motions, which cut around corners (example from Peter Corke, "Robotics, Vision and Control"):



We will use a simpler version of this algorithm which uses a pre-defined time interval for the "cornering" part of the trajectory (for example, 0.1s on each side of the waypoint). We also pre-define how "far" from the waypoint we are willing to cut (for example, 0.1 rad). These pre-set values can be chosen conservatively, to ensure the robot does not hit its limits during the blended trajectory.

Once these parameters are set, a polynomial can be used to fit the blended trajectory. After the polynomial is completed, it can be sampled in order to check if either the velocity or the acceleration exceeds a joint limit; if so, the polynomial is re-computed allowing more execution time.

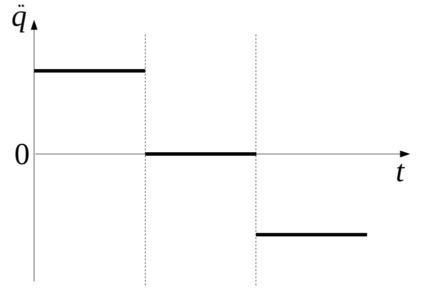
For the constant-velocity parts of the trajectory, we can simply use a specified constant velocity for each segment, and calculate the minimum time needed to get to the destination.

3 Bang-coast-bang profile

The main shortcoming of the polynomial fitting method is that the velocity and acceleration that result might be below what the joint can actually do. In practice, we often want a path to be executed as fast as possible (or at least close to that), with the joint accelerating and moving at its (safe) limits.

One can obtain higher velocity and accelerations by simply requesting a smaller time interval T for executing the segment; however, the relationship between maximum velocity, acceleration, and T is not straightforward.

Another common approach is referred to as using a "bang-coast-bang" or "trapezoidal" strategy: the joint accelerates as fast as possible until reaching its max velocity, coasts at max velocity, then decelerates as fast as possible to get to the destination.



We assume the following problem definition and joint specs:

- a_M: maximum allowable joint acceleration. This is a limit imposed by the robot's manufacturer for safety purposes, regardless of the trajectory being executed. Exceeding it might damage the robot. For simplicity, we assume the same absolute value for acceleration and deceleration.
- v_M: maximum allowable joint velocity. This is a limit imposed by the robot's manufacturer for safety purposes, regardless of the trajectory being executed. Exceeding it might damage the robot.
- v_{max}: the maximum velocity achieved by the joint during execution of this trajectory. Obviously, $v_{max} \leq v_M$.
- t_a: acceleration time (time spent at a_M)
- t_d: deceleration time (time spent at $-a_M$)
- t_c: coasting time (time spent at v_{max})
- T: total segment time. We note that $t_a + t_c + t_d = T$.
- Δq: total distance traveled by the joint during the segment

Ultimately, computing this trajectory boils down to the following question: what is the value of v_{max}, the higher velocity the joint will achieve during the trajectory? If the joint accelerates from v_0 all the way to v_{max}, the time needed to do that is

$$t_a = \frac{v_{max} - v_0}{a_M} \tag{23}$$

During this time, the distance traveled will be:

$$\Delta q_a = v_0 t_a + a_M \frac{t_a^2}{2} = \frac{v_{max}^2 - v_0^2}{2a_M} \tag{24}$$

Similarly, the time needed to decelerate from v_{max} to v_T will be:

$$t_d = \frac{v_{max} - v_T}{a_M} \tag{25}$$

The distance traveled while decelerating will be:

$$\Delta q_d = v_{max} t_d - a_M \frac{t_d^2}{2} = \frac{v_{max}^2 - v_T^2}{2a_M} \tag{26}$$

The total distance traveled during the ramp phases will thus be:

$$\Delta q_{ramp} = \Delta q_a + \Delta q_d = \frac{2v_{max}^2 - v_0^2 - v_T^2}{2a_M} \tag{27}$$

Now we can check if the joint actually has time to reach v_M, the highest velocity allowable by construction. Assuming $v_{max} = v_M$, we use the relationship above to compute the distance traveled during ramp time, and compare that to the the total distance to travel(Δq)

Case 1

$$\Delta q_{rampmax} = \frac{2v_M^2 - v_0^2 - v_T^2}{2a_M} \leq \Delta q \tag{28}$$

In this case, the joint has time to reach its maximum velocity: $v_{max} = v_M$. Since the distance traveled during ramp up and ramp down is less than the total distance to travel, it means the joint must also spend some time "coasting" at max velocity. Coast time can be computed as:

$$t_c = \frac{\Delta q - \Delta q_{rampmax}}{v_{max}} \tag{29}$$

Total segment time will be

$$T = t_a + t_d + t_c \tag{30}$$

Case 2

$$\Delta q_{rampmax} > \Delta q \tag{31}$$

This means that the joint does not have time to reach its max velocity before it must start slowing down. As a result, we must have $v_{max} < v_M$. Also, there is no "coast" phase. We can compute the exact value of v_{max} by noting that the distance traveled during acceleration and deceleration must be equal to total segment distance.

$$\Delta q = \Delta q_{ramp} = \frac{2v_{max}^2 - v_0^2 - v_T^2}{2a_M} \tag{32}$$

From this relationship, we can directly compute v_{max}, which we then use to compute t_a and t_d. Total segment time will be

$$T = t_a + t_d \tag{33}$$

Example 1. A joint is required to go from $q_0 = 1rad$ to $q_T = 3rad$, starting from rest and ending at rest ($\dot{q}_0 = \dot{q}_T = 0$). Joint limits are $v_M = 1rad/s$ and $a_M = 1rad/s^2$.

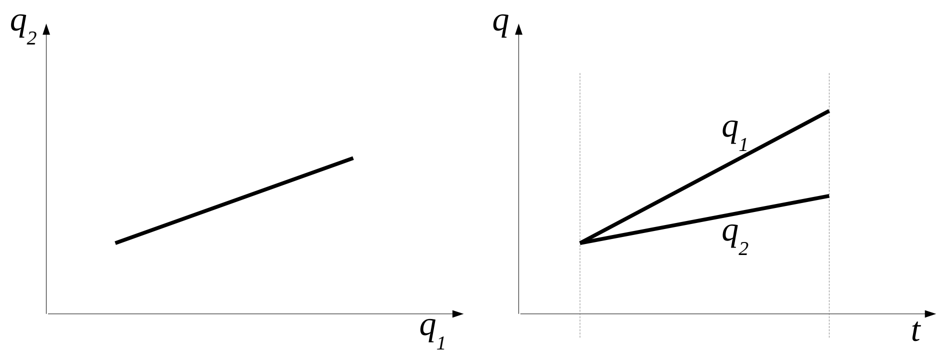
- Compute the total time T required to achieve the goal using a bang-coast-bang trapezoidal timing function, as well as the portions of that time spent accelerating (t_a), decelerating (t_d) and coasting at constant velocity (t_c).
- Plot the position, velocity and acceleration profiles for the trajectory segment.

Example 2. Same problem as above, but assuming a faster joint: $v_{max} = 2rad/s$.

4 Multi-dimensional trajectories

Most trajectories for robots do not involve a single joint. An arm moving through a trajectory means all joints have to move together. Furthermore, they have to move in **synchronized fashion**. If individual joint trajectories are not synchronized, then the arm will deviate from the desired trajectory even if each joint correctly executes its own segment.

It is very important to note that, when looking at plots of multi-dimensional trajectories, straight line motion has a different meaning than when looking at a single joint trajectory plotted against time. For a single joint position plotted against time, straight-line motion means constant velocity. For a 2-joint trajectory plotted in configuration space, straight line motion means that the 2 joints are synchronized, and their velocities are at a constant ratio with respect to each other.



This is one reason why constant-velocity segments are preferred for single joint trajectories: if the velocity is constant, it is much easier to synchronize multiple joints.

In practice, multi-dof trajectories are also allowed to deviate from straight line motion in order to cut corners. One option for implementing this is a version of the blended trajectories we used earlier, with the following additional constraints:

- during constant-velocity segments, joint velocities must be in the ratio dictated by the overall trajectory.
- joints must synchronize when entering and exiting corner segments. During execution of corner segments, velocities are not expected to maintain a desired ratio; as a result, the overall trajectory will deviate from straigh-line motion.

For executing constant-line segments as fast as possible, the key is to identify which joint will need the most time for executing the segment. All other joints will then move at the speeds dictated by the need to synchronize.