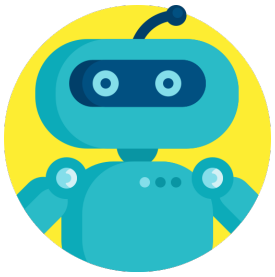


MEC4127F: Introduction to Robotics

Chapter 7: Trajectory generation

Table of Contents

7.1 Introduction

Given a high-level objective, such as moving a robot from one defined point in 2D space to another, the task of defining the corresponding motion required may seem trivial. A straight line that joins the two points will give the shortest path. However, the question still remains, how quickly one must along the path (in terms of velocity and acceleration) in order to be energy efficient. Additionally, the particular robotic platform may not be capable of performing the simple straight line movement as a result of obstacles, workspace limitations, or kinematic

constraints (imagine a car trying to parallel park) — see Figure 7.1 as an example. There may also be time requirements related to moving from the initial location to the final location.

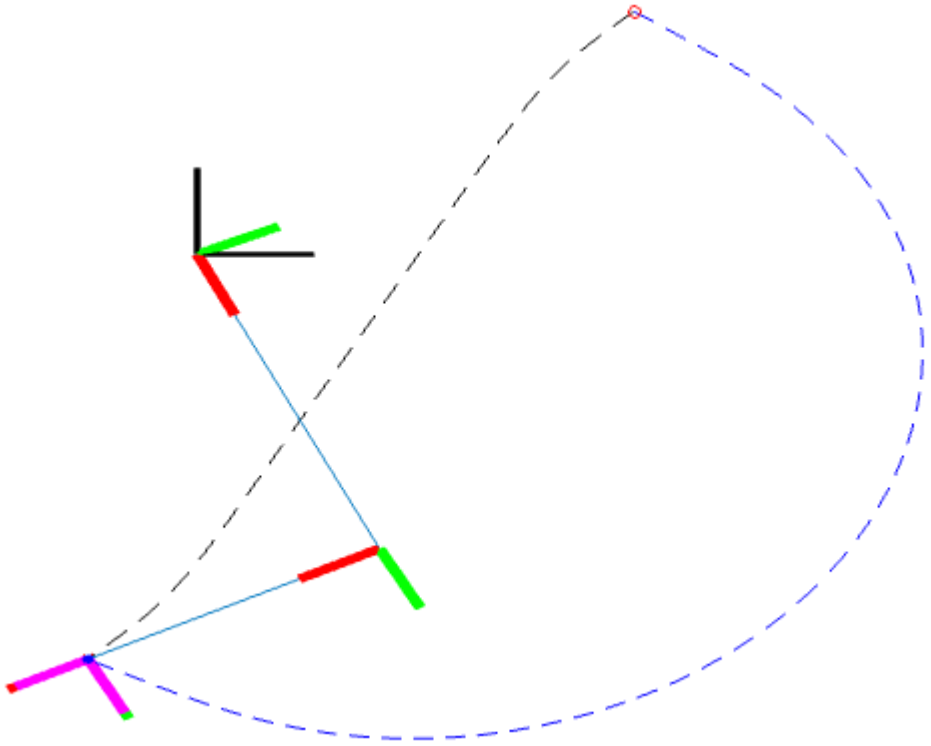


Figure 7.1: Two different end-effector position trajectories that have common start and end points. The black trajectory is the result of a task space trapezoidal velocity profile, whereas the blue trajectory is a result of a joint space trapezoidal velocity profile.

The first consideration should be whether the robot is physically capable of moving from the initial pose to the final pose. The underlying *path* that joins the two poses should exist within the robot workspace in order to avoid unrealistic expectations of the robot.

A *trajectory* is a path on which a timing law is specified. If our path was described on a plane, the corresponding trajectory can be thought of as a 3D description of the 2D plane, where a time axis has been added. In this way, the path is parameterised by a time variable and this inherently induces derivative information from our original path (e.g. velocity and acceleration trajectories). We are no longer just interested in where the path is, but also *when* it is.

When dealing with the trajectory generation problem, it is important to consider the dynamic and kinematic limitations of the robotic platform. A servomotor, for example, has a rated maximum slew rate (rate of change of position), which means that it cannot be used to reliably track trajectories that require angular velocities above its rated limit. Trajectories should therefore be generated with the underlying saturation limits of the actuators in mind. Violating said limits may not seem catastrophic for tame single-input-single-output systems, but larger multi-input-multi-output systems can experience an irrecoverable instability if saturation occurs at the wrong time (we will see this later when controlling wheeled robots and quadcopters).

Before going into the mathematical detailing of the problem, we will first introduce some important definitions and terminology to better understand the problem at hand.

7.1.1 Path

A generalised **path** $X(s)$ describes the mapping of a scalar path parameter, s (which is not the Laplace variable in this instance!), to a point in the robot's configuration space $\mathscr{C}$. Scalar parameter s is assumed to start at 0 at the start of the path, and equal 1 at the end, with $s \in [0, 1]$. As s increases monotonically from 0 to 1, the robot moves along the specified path. As we will show later, $X(s)$ can refer to a rectilinear path, or a rotational path.

7.1.2 Time scaling

The focus of this Chapter will be to determine the appropriate **time scaling**, $s(t)$, which assigns a value s to each time time point $t \in [0, T]$, where T is the completion time of the manoeuvre. Therefore, the function $s(t)$ performs the following mapping:

$$s : [0, T] \rightarrow [0, 1].$$

For example, at the start of the path, $t = 0$, which maps to $s(t) = 0$, whereas at the end of the manoeuvre, $t = T$, and $s(t) = 1$. The following sections will look at defining the mapping for all points in $t \in (0, T)$.

7.1.3 Trajectory

When a path and time scaling are combined, the result is defined as a **trajectory**, $X(s(t))$, or $X(t)$ in short. A trajectory therefore has spatial and temporal meaning. The robot is no longer just required to be in a particular configuration, but rather also be in that particular configuration at a specified time.

The velocity and acceleration along a trajectory can be defined, respectively, using chain rule as

$$\dot{X} = \frac{dX}{ds}\dot{s},$$

$$\ddot{X} = \frac{dX}{ds}\ddot{s} + \frac{d^2X}{dX^2}\dot{s}^2.$$

We require the robot's acceleration (and by extension, dynamics) to be well defined. As such, both $X(s)$ and $s(t)$ must be twice differentiable.

7.2 Straight-line paths

A **straight-line path** is the simplest form of joining a start configuration, X_0, and end configuration, X_T — be it a straight line in translation or rotation (or both).

For rectilinear motion and rotations about a single axis, straight line equations can be written as

$$X(s) = X_0 + s(X_T - X_0),$$

where $s \in [0, 1]$. Notably,

- $s = 0 \Rightarrow X(s) = X_0$,
- $s = 1 \Rightarrow X(s) = X_T$.

The partial derivatives of $X(s)$ with respect to s can be calculated as

$$\frac{dX}{ds} = X_T - X_0,$$

$$\frac{d^2X}{ds^2} = 0.$$

The result above implies that a trajectory using straight line paths will have velocity and acceleration profiles of

$$\begin{aligned} \dot{X} &= \frac{dX}{ds}\dot{s}, \\ &= (X_T - X_0)\dot{s}, \end{aligned}$$

$$\begin{aligned} \ddot{X} &= \frac{dX}{ds}\ddot{s} + \frac{d^2X}{dX^2}\dot{s}^2, \\ &= \frac{dX}{ds}\ddot{s}, \\ &= (X_T - X_0)\ddot{s}. \end{aligned}$$

7.2.1 Linear interpolation

When applied to translational paths, the equation $X(s) = X_0 + s(X_T - X_0)$ is often also referred to as **linear interpolation** (LERP). This formulation can be used to construct an n-dimensional path for the translation (position) of a particular robot. For example, if a robot's desired position in $\{W\}$ is given by ${}^W\mathbf{p}^*(s) \in \mathbb{R}^3$, LERP can construct an associated straight-line path of

$${}^W\mathbf{p}^*(s) = {}^W\mathbf{p}_0^* + s\left({}^W\mathbf{p}_T^* - {}^W\mathbf{p}_0^*\right),$$

where ${}^W\mathbf{p}_0^*$ and ${}^W\mathbf{p}_T^*$ are the desired start and end positions of the robot in the 3D world frame, respectively. The only unknown would the function s, and how performs the mapping from start time at $t = 0$ to completion time $t = T$.

7.2.2 Spherical linear interpolation

Spherical linear interpolation (SLERP) is the rotational counterpart to linear interpolation, which is required when multi-dimensional rotations are of interest. One can picture SLERP as a path that is drawn along a unit-radius great circle arc. The equation for LERP of $X(s) = X_0 + s(X_T - X_0)$ cannot be applied in general to rotation motion, as shown in **Chapter 3**. We instead can make use of exponential coordinates when applying the time scaling s and then convert back to rotation matrices or quaternions when performing the rotation.

7.2.2.1 Rotation matrices

Recall that the rotation matrix $\mathbf{R}$ is related to exponential coordinates $\alpha\hat{\mathbf{v}}$ using the matrix exponential of

$$\mathbf{R} = \mathbf{e}^{\lfloor \alpha\hat{\mathbf{v}} \rfloor} = \mathbf{e}^{\lfloor \hat{\mathbf{v}} \rfloor \alpha}.$$

Exponential coordinates can also be extracted from rotation matrices (in skew-symmetric form) using the matrix logarithm:

$$\alpha\lfloor \hat{\mathbf{v}} \rfloor = \log \mathbf{R}.$$

Considering an initial orientation of ${}^{W}\mathbf{R}(0) = {}^{W}\mathbf{R}_0$ and final orientation of ${}^{W}\mathbf{R}(T) = {}^{W}\mathbf{R}_T$, the incremental rotation matrix that rotates ${}^{W}\mathbf{R}_0$ to ${}^{W}\mathbf{R}_T$, given by ${}^{0}\mathbf{R}_T$, accomplishes

$${}^{W}\mathbf{R}_T = {}^{W}\mathbf{R}_0 {}^{0}\mathbf{R}_T.$$

As the notation implies, ${}^{0}\mathbf{R}_T$ is rotation matrix describing the orientation of the reference frame at the end of the motion, with respect to the initial reference frame. One can then solve for ${}^{0}\mathbf{R}_T$ based on

$$\begin{aligned} {}^{0}\mathbf{R}_T &= {}^{W}\mathbf{R}_0^{T\,W}\mathbf{R}_T, \\ &= {}^{0}\mathbf{R}_W {}^{W}\mathbf{R}_T. \end{aligned}$$

Using the matrix logarithm, we can extract the exponential coordinates from ${}^{0}\mathbf{R}_T$ using

$$\begin{aligned} \Delta\alpha\lfloor {}^{0}\hat{\mathbf{v}} \rfloor &= \log {}^{0}\mathbf{R}_T, \\ &= \log \left({}^{0}\mathbf{R}_W {}^{W}\mathbf{R}_T \right). \end{aligned}$$

The time scaling parameter s can be applied to the exponential form of the incremental rotation using

$$\begin{aligned} (s\Delta\alpha)\lfloor {}^{0}\hat{\mathbf{v}} \rfloor &= s \log {}^{0}\mathbf{R}_T, \\ &= s \log \left({}^{0}\mathbf{R}_W {}^{W}\mathbf{R}_T \right). \end{aligned}$$

A simple way to think of this time scaling is that the effective rotation angle of $s\Delta\alpha$ is interpolated based on the variation of $s \in [0, 1]$, with the rotation vector remaining unchanged.

Finally, the time-scaled exponential coordinates can be mapped back into rotation matrix form using the matrix exponential and then pre-multiplied by the rotation matrix describing the initial orientation, namely ${}^{W}\mathbf{R}_0$, to obtain the time-scaled parameterisation of the trajectory:

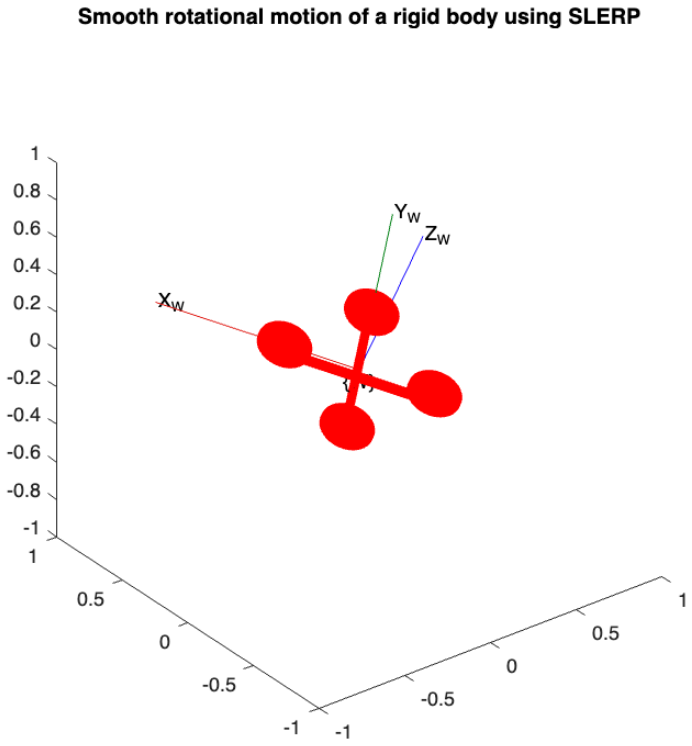
$$\begin{aligned} {}^{W}\mathbf{R}(s) &= {}^{W}\mathbf{R}_0 \mathbf{e}^{s \log {}^{0}\mathbf{R}_T}, \\ &= {}^{W}\mathbf{R}_0 \mathbf{e}^{s \log \left({}^{0}\mathbf{R}_W {}^{W}\mathbf{R}_T \right)}. \end{aligned}$$

The result above represents spherical linear interpolation from ${}^{W}\mathbf{R}_0$ to ${}^{W}\mathbf{R}_T$, which is dependent on the choice of s.

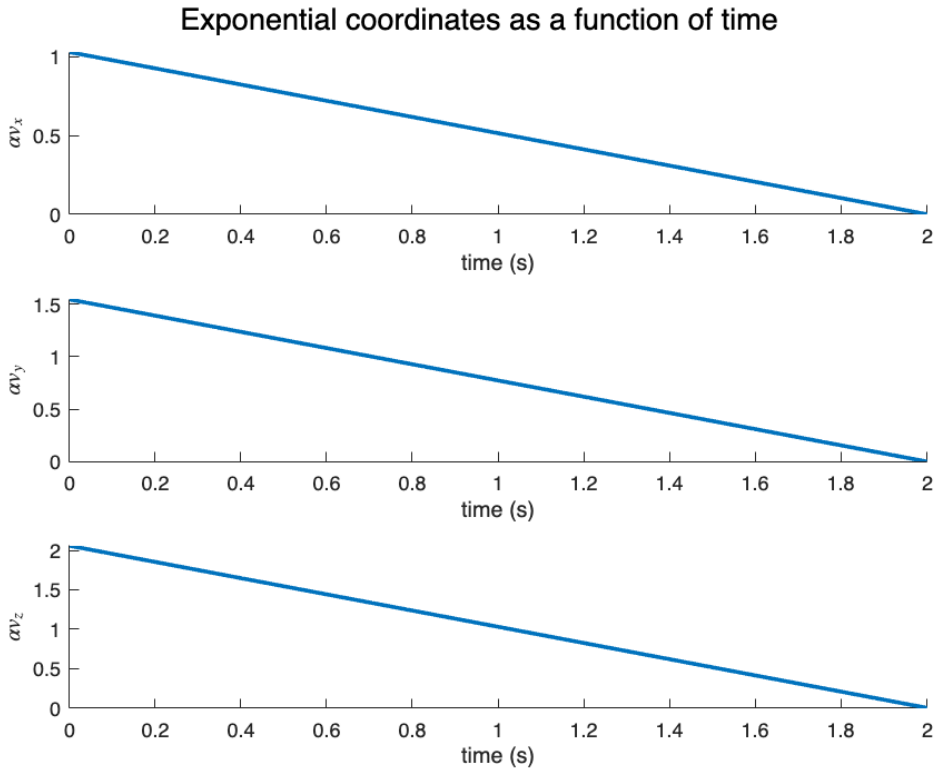
Example: SLERP using rotation matrices

The code block below provides an example of how SLERP can be used to smoothly rotate a rigid body from an arbitrary initial orientation of ${}^{W}\mathbf{R}_0$ to a final orientation of ${}^{W}\mathbf{R}_T = \mathbf{I}$. In this case, a first-order time scaling, based on Section 7.3.1, is used, which can be seen based on how the exponential coordinates change linearly over time. Note that any time scaling can be used in practice.

```
R0 = [ -0.666666666666667   0.133333333333333   0.733333333333334;
         0.666666666666667  -0.333333333333333   0.666666666666667;
         0.333333333333333   0.933333333333334   0.133333333333333];
RT = [1 0 0;
      0 1 0;
      0 0 1];
dR = R0'*RT;

T = 2;
numSamples = 1e2;
t = linspace(0,T,numSamples);
s = t/T;

alphaV = zeros(length(t),3);

figure
for i=1:length(t)
    R = R0*expm(s(i)*logm(dR));

    axang = rotm2axang(R);    %extract current axis-angle elements for sake
of plotting
    alphaV(i,:) = axang(4)*axang(1:3);    %consolidate axis-angle form into
exponential coordinates for plotting

    %3D visualisation of rotational motion
    plotTransforms([0 0
0],rotm2quat(R),"MeshFilePath",'multirotor.stl',"MeshColor","red","FrameAxis
Labels","on","FrameLabel","W")

    axis equal
    xlim([-1 1.00])
    ylim([-1 1.00])
    zlim([-1 1.000])
    title('Smooth rotational motion of a rigid body using SLERP')
    pause(1/numSamples)

end
```




```
%plot exponential coordinates as a function of time
figure,sgtitle('Exponential coordinates as a function of time')
subplot(3,1,1),hold on
plot(t,alphaV(:,1),LineWidth=2),ylabel('$\alpha
v_x$',Interpreter='latex'),xlabel('time (s)')
subplot(3,1,2),hold on
plot(t,alphaV(:,2),LineWidth=2),ylabel('$\alpha
v_y$',Interpreter='latex'),xlabel('time (s)')
subplot(3,1,3),hold on
plot(t,alphaV(:,3),LineWidth=2),ylabel('$\alpha
v_z$',Interpreter='latex'),xlabel('time (s)')
```



7.2.2.2 Quaternions

The quaternion formulation using SLERP follows a similar procedure to that of the rotation matrix. We first define the *exponential mapping*, $\exp$, that relates the exponential coordinates to quaternions, as

$$\exp: \qquad \alpha\hat{\mathbf{v}} \in \mathbb{R}^3 \rightarrow \mathbf{q} \in \mathbb{R}^4.$$

We can therefore use this mapping to easily represent the quaternion as

$$\mathbf{q} \quad = \exp(\alpha\hat{\mathbf{v}}) = \begin{bmatrix} \cos\frac{\alpha}{2} \\ \sin\frac{\alpha}{2}\hat{\mathbf{v}} \end{bmatrix}.$$

The inverse is referred to as the *logarithmic mapping* with the associated property of

$$\log: \qquad \mathbf{q} \in \mathbb{R}^4 \rightarrow \alpha\hat{\mathbf{v}} \in \mathbb{R}^3,$$

or explicitly,

$$\alpha\hat{\mathbf{v}} = \log(\mathbf{q}).$$

Considering an initial orientation of ${}^{W}\mathbf{q}_0$ and final orientation of ${}^{W}\mathbf{q}_T$, the incremental rotation matrix that rotates ${}^{W}\mathbf{q}_0$ to ${}^{W}\mathbf{q}_T$, given by ${}^{0}\mathbf{q}_T$, accomplishes

$${}^{W}\mathbf{q}_T = {}^{W}\mathbf{q}_0 \otimes {}^{0}\mathbf{q}_T.$$

As the notation implies, ${}^{0}\mathbf{q}_{T}$ is the quaternion describing the orientation of the reference frame at the end of the motion, with respect to the initial reference frame. One can then solve for ${}^{0}\mathbf{q}_{T}$ based on

$$\begin{aligned} {}^{0}\mathbf{q}_{T} &= {}^{W}\mathbf{q}_{0}^{-1} \otimes {}^{W}\mathbf{q}_{T}, \\ &= {}^{0}\mathbf{q}_{W} \otimes {}^{W}\mathbf{q}_{T}. \end{aligned}$$

Using the logarithmic mapping, we can extract the exponential coordinates from ${}^{0}\mathbf{q}_{T}$ using

$$\begin{aligned} \alpha {}^{0}\hat{\mathbf{v}} &= \log\left({}^{0}\mathbf{q}_{T}\right), \\ &= \log\left({}^{0}\mathbf{q}_{W} \otimes {}^{W}\mathbf{q}_{T}\right). \end{aligned}$$

The time scaling parameter s can then be applied to the exponential form of the incremental rotation using

$$\begin{aligned} (s\alpha) {}^{0}\hat{\mathbf{v}} &= s \log {}^{0}\mathbf{q}_{T}, \\ &= s \log\left({}^{0}\mathbf{q}_{W} \otimes {}^{W}\mathbf{q}_{T}\right). \end{aligned}$$

A simple way to think of this time scaling is that the effective rotation angle of $s\alpha$ is interpolated based on the variation of $s \in [0, 1]$, while the rotation vector remains unchanged.

Finally, the time-scaled exponential coordinates can be mapped back into quaternion form using the exponential mapping and then pre-multiplied by the quaternion describing the initial orientation, namely ${}^{W}\mathbf{q}_{0}$, to obtain the time-scaled parameterisation of the trajectory:

$$\begin{aligned} {}^{W}\mathbf{q}(s) &= {}^{W}\mathbf{q}_{0} \otimes \exp\left(s \log {}^{0}\mathbf{q}_{T}\right), \\ &= {}^{W}\mathbf{q}_{0} \otimes \exp\left(s \log\left({}^{0}\mathbf{q}_{W} \otimes {}^{W}\mathbf{q}_{T}\right)\right). \end{aligned}$$

Analogous to the rotation matrix result in the previous sub-section, the equation above represents spherical linear interpolation from ${}^{W}\mathbf{q}_{0}$ to ${}^{W}\mathbf{q}_{T}$, which is dependent on the choice of s.

7.3 Time-scaling

This section will consider two approaches to time scaling, namely:

- polynomial interpolation (1st-order, 3rd-order, 5th-order), and
- trapezoidal velocity profiles.

We will show that as the degree of the polynomial interpolation becomes higher, we obtain smoother, more well-behaved trajectories. However, we also need to specify more information about the trajectory, which may not be easily known. Trapezoidal velocity profiles are an alternative approach to polynomial interpolation that considers time optimality over smoothness, which also comes with some strict constraints.

We will only consider point to point motion in this Section. That is, a motion that starts at rest and ends at rest. A latter section on polynomial splines will cover the general motion case, where the initial and final velocities are not necessarily zero.

7.3.1 First-order polynomial interpolation

Consider the task of using a first-order time scaling to describe the trajectory that adheres to $s : [0, T] \to [0, 1]$. The corresponding 1^{st}-order position trajectory can be written as

$$s(t) = a_0 + a_1 t,$$

where $\{a_0, a_1\} \in \mathbb{R}$ are constants, t is the time parameter, and T is the time that corresponds with the end of the trajectory. Assuming $t \in [0, T]$, we can find both parameters using

$$s(0) = a_0 = 0,$$

$$s(T) = a_0 + a_1 T = a_1 T = 1.$$

We can therefore determine a_i as

$$\begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{1}{T} \end{bmatrix},$$

which results in the simple first-order time-scaling of

$$s(t) = \frac{t}{T}.$$

Note that only the terminal time needs to be specified to define $s(t)$. To see how the corresponding velocity trajectory will evolve, we differentiate $s(t)$, which gives us

$$\dot{s}(t) = \frac{1}{T}.$$

A demonstration of first-order time scaling can be seen in the Example in Section 7.2.2.1.

The choice of first-order polynomial above means that we have no way to enforce a desired initial and final time-scaling velocity. Given the task of point to point motion, which implies that the robot is stationery (zero velocity) at the start and end of the motion, the velocity will be required to instantaneously jump from zero to $\frac{1}{T}$ at $t = 0$, and from $\frac{1}{T}$ back to zero at $t = T$ (see Figure 7.2).

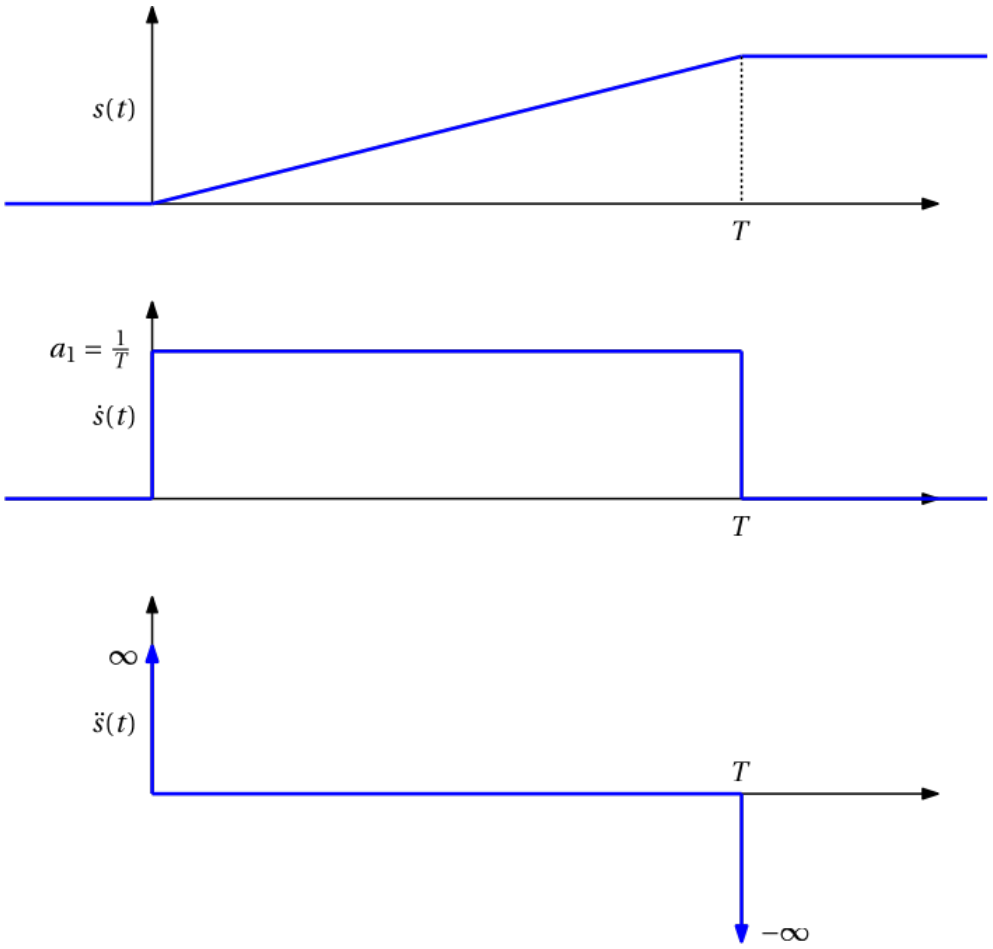


Figure 7.2: First-order polynomial interpolation of a point to point trajectory and the resulting time-derivative profiles.

This implies that the corresponding time-scaling acceleration at $t = 0$ and $t = T$ are infinite impulses (dirac delta functions) as the velocity gradient at these time points are infinitely steep. This type of impulsive behaviour implies instantaneously applying a very large amount of energy in practice and should be avoided ideally, as the actuators that are inherently responsible for motion cannot produce such a signal, nor do they want to! - sharp signals on actuators results in wear and can damage the mechanical components. For this reason, a higher order interpolation method is required that has some favourable derivative properties.

7.3.2 Cubic polynomial interpolation

Cubic (3^{rd}-order) polynomials are commonly used to interpolate between two points, and can be written as

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

where $\{a_0, a_1, a_2, a_3\} \in \mathbb{R}$ are constants. The equation above can be repeatedly differentiated to find the corresponding kinematic profiles, namely, the quadratic *velocity* profile

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

the linear *acceleration* profile

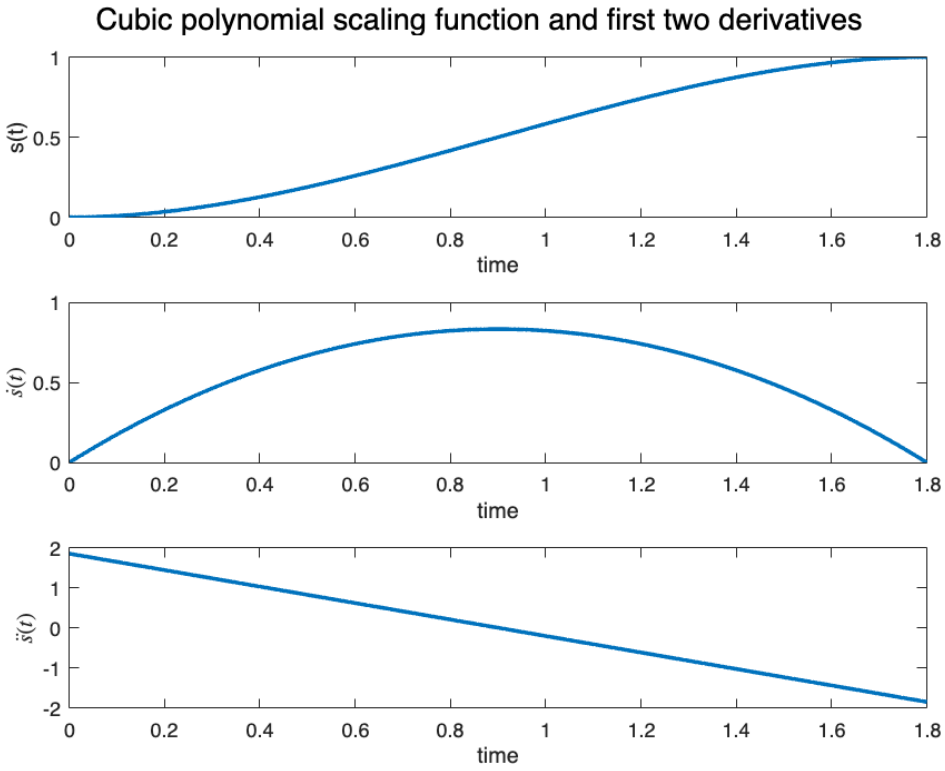
$$\ddot{s}(t) = 2a_2 + 6a_3 t,$$

and the constant *jerk* profile

$$\dddot{s}(t) = 6a_3.$$

With reference to the figure generated from the code block below, the position and velocity profiles notably exhibits a continuous behaviour, as a result of being at least twice twice differentiable, which is a necessary condition for a *smooth* position trajectory. However, the acceleration (and by extension, jerk) profile does not have the same property. The acceleration profile, for example, is required to instantaneously change from 0 to $2a_2$ at $t = 0$.

```
T =1.8; %adjust completion time
A = [0 0 3/T^2 -2/T^3];

t = linspace(0,T,1e3);

s = A(1)+A(2)*t+A(3)*t.^2+A(4)*t.^3;
ds = A(2)+2*A(3)*t+3*A(4)*t.^2;
dds = 2*A(3)+6*A(4)*t;

figure,sgtitle('Cubic polynomial scaling function and first two
derivatives')
subplot(3,1,1),plot(t,s,LineWidth=2),ylabel('s(t)'),xlabel('time')
subplot(3,1,2),plot(t,ds,LineWidth=2),ylabel('$\dot{s}(t)
$',Interpreter='latex'),xlabel('time')
subplot(3,1,3),plot(t,dds,LineWidth=2),ylabel('$\ddot{s}(t)
$',Interpreter='latex'),xlabel('time')
```



If we were interested in ensuring there was continuity at the acceleration and jerk level, we could use a higher order polynomial, such a quartic (4^{th}-order) or quintic (5^{th}-order), which will be discussed in Section 7.3.3.

Definition: Smooth trajectory

A trajectory in which the first two temporal derivatives are continuous is classed as being a **smooth trajectory**.

Smoothness is appealing if we want our robot to operate in an energy efficient and controlled manner. In fact, the cubic polynomial trajectory originates as a result of solving an energy-optimal cost function. So while we may not have a truly smooth trajectory at all kinematic levels, we are at least commanding finite acceleration commands; something that an actuator can more closely approximate.

Given that there are four undefined constants of $\{a_0, a_1, a_2, a_3\}$, we need to impose four constraints on the various profiles in order to solve said parameters. This can be done by assigning initial and final position and velocity requirements. Based on the requirement of the time-scaling function, we require $s(0) = 0$ and $s(T) = 1$. We would then have to choose corresponding initial and final velocity constraints of $\dot{s}(0)$ and $\dot{s}(T)$. In the case of point to point motion (with the robot required to be stationary at the start and end of the motion), this is simply $\dot{s}(0) = \dot{s}(T) = 0$. Based on this information, we can make the following deductions:

$$s(0) = a_0 = 0.$$

$$\dot{s}(0) = a_1 + 2a_2 0 + 3a_3 0^2 = a_1 = 0.$$

$$\begin{aligned} s(T) &= a_0 + a_1 T + a_2 T^2 + a_3 T^3, \\ &= a_1 T + a_2 T^2 + a_3 T^3, \\ &= a_2 T^2 + a_3 T^3, \\ &= 1. \end{aligned}$$

$$\dot{s}(T) = a_1 + 2a_2 T + 3a_3 T^2 = 2a_2 T + 3a_3 T^2 = 0.$$

The last two equations above constitute two unknown parameters — $\{a_2, a_3\}$. We can also write this in matrix form as

$$\begin{bmatrix} T^2 & T^3 \\ 2T & 3T^2 \end{bmatrix} \begin{bmatrix} a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix},$$

and then invert our matrix to solve for our constant vector

$$\begin{aligned} \begin{bmatrix} a_2 \\ a_3 \end{bmatrix} &= \begin{bmatrix} T^2 & T^3 \\ 2T & 3T^2 \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \\ &= \begin{bmatrix} \frac{3}{T^2} \\ -\frac{2}{T^3} \end{bmatrix}. \end{aligned}$$

Note that in point to point motion we only need to select terminal time T to determine $\{a_2, a_3\}$.

```
syms T real

M = [T^2 T^3;
     2*T 3*T^2];
A = M\[1 0]'

-A(1)/(3*A(2))
```

Once our constants and initial conditions are defined, we can populate the kinematic equations governing the time-scaling function, namely

$$s(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 = \frac{3}{T^2} t^2 - \frac{2}{T^3} t^3,$$

$$\dot{s}(t) = a_1 + 2a_2 t + 3a_3 t^2 = \frac{6}{T^2} t - \frac{6}{T^3} t^2,$$

$$\ddot{s}(t) = 2a_2 + 6a_3 t = \frac{6}{T^2} - \frac{12}{T^3} t,$$

$$\dddot{s}(t) = 6a_3 = -\frac{12}{T^3}.$$

Notably, we have no control over how the acceleration profile evolves. If we wanted to have a say about the initial and final acceleration conditions, we would require a quintic polynomial — see Section 7.3.3. Given that our velocity profile from the equation above is quadratic, we can find the time point when the maximum value occurs (at the turning point) as

$$t_{max} = -\frac{a_2}{3a_3} = \frac{T}{2},$$

which occurs at the halfway point of the motion due to symmetry. The corresponding maximum velocity then follows as

$$\dot{s}_{max} = \dot{s}(t_{max}) = \frac{6}{T^2} t_{max} - \frac{6}{T^3} t_{max}^2 = \frac{3}{2T}.$$

```
syms T real
t_max = T/2
ds_max = 6/T^2*t_max-6/T^3*t_max^2
```

If we know the time-scaling velocity limit, $\dot{s}_{lim}$, we can impose a constraint on the obtainable maximum velocity, and then ensure that

$$\dot{s}_{max} = \frac{3}{2T} \leq \dot{s}_{lim}.$$

Recalling that the velocity of a straight-line trajectory is given as

$$\dot{X} \ = (X_T - X_0)\dot{s},$$

we can quantify the constraint on the maximum velocity of the trajectory, $0 < \dot{X}_{max} \leq \dot{X}_{lim}$ based on

$$\dot{X}_{max} \ = |X_T - X_0|\dot{s}_{max} \leq |X_T - X_0|\dot{s}_{lim} \leq \dot{X}_{lim},$$

The time-scaling velocity limit can then be determined as

$$\dot{s}_{lim} = \frac{\dot{X}_{lim}}{|X_T - X_0|},$$

which can be used to enforce the selection of T using

$$\dot{s}_{max} = \frac{3}{2T} \leq \frac{\dot{X}_{lim}}{|X_T - X_0|},$$

or explicitly

$$T \geq \frac{3|X_T - X_0|}{2\dot{X}_{lim}}.$$

We can similarly impose a constraint on our maximum obtainable acceleration: $0 < \ddot{X}_{max} \leq \ddot{X}_{lim}$. This is done by first identifying that the time-scaling acceleration profile will have a maximum value at either the start point or end point, as the acceleration profile is a straight line. In the case of point to point motion, the initial and final accelerations will be equivalent, resulting in the same constraint of

$$\frac{6}{T^2} \leq \ddot{s}_{lim},$$

where $\ddot{s}_{lim}$ is the time-scaling acceleration limit. Given that

$$\ddot{X} = (X_T - X_0)\ddot{s},$$

the maximum acceleration of the motion will be

$$\ddot{X}_{max} = |X_T - X_0|\ddot{s}_{max} \leq |X_T - X_0|\ddot{s}_{lim} \leq \ddot{X}_{lim}.$$

We can therefore relate the terminal time requirement to the maximum acceleration of the trajectory as

$$\frac{6|X_T - X_0|}{T^2} \leq \ddot{X}_{lim},$$

or directly as

$$T \geq \sqrt{\frac{6|X_T - X_0|}{\ddot{X}_{lim}}}.$$

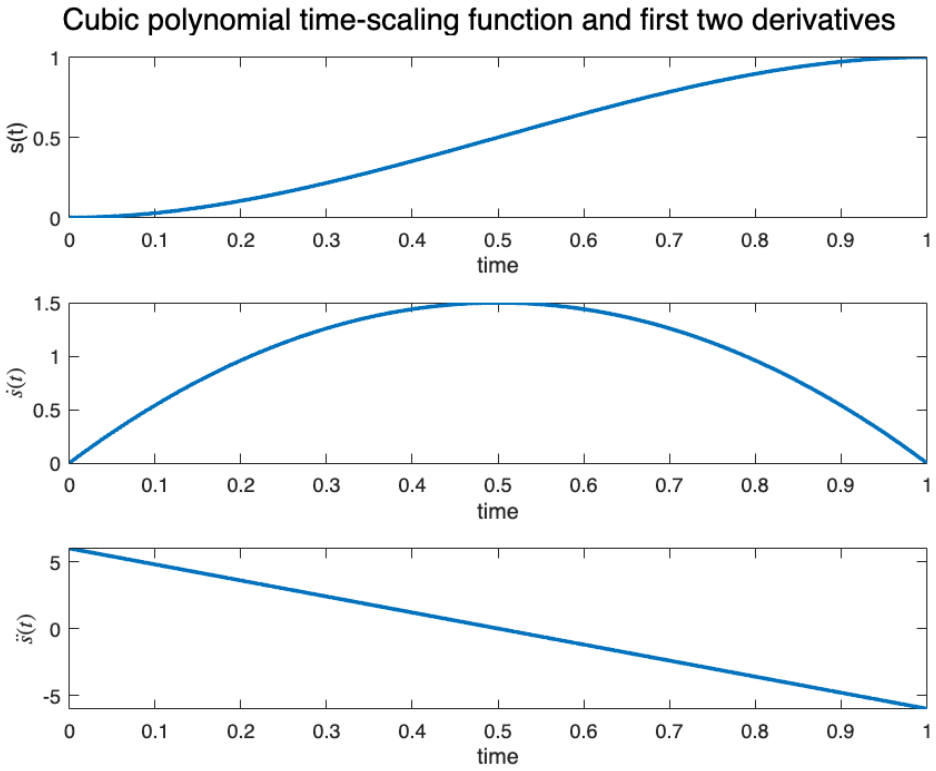
Therefore, given the above-mentioned requirements on completion time T, one can choose the duration of the cubic polynomial in order to meet the underlying velocity and/or acceleration constraints.

Example: Cubic polynomial trajectory

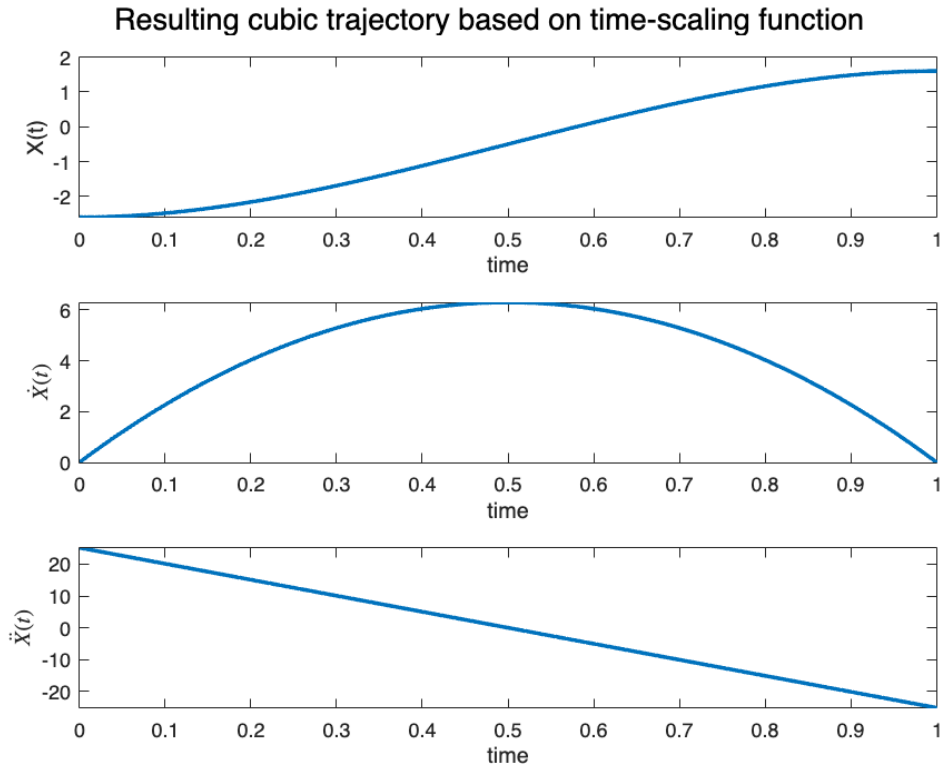
The example below generates a cubic time-scaling function, $s(t)$, and resulting trajectory, $X(t)$. The time-scaling function is only dependent on the choice of completion time, T, whereas the scale of $X(t)$ will depend on the initial and final positions that are specified.

```
X0 =-2.6; %initial position
XT =1.6; %final position
T = 1;    %completion time

M = [1 0 0 0;
     0 1 0 0;
     1 T T^2 T^3;
     0 1 2*T 3*T^2];

A = M\[0 0 1 0]';

t = linspace(0,T,1e3);

s = A(1)+A(2)*t+A(3)*t.^2+A(4)*t.^3;
```

```
ds = A(2)+2*A(3)*t+3*A(4)*t.^2;
dds = 2*A(3)+6*A(4)*t;
ddds = 6*A(4)*ones(size(t));

figure,sgtitle('Cubic polynomial time-scaling function and first two
derivatives')
subplot(3,1,1),plot(t,s,LineWidth=2),ylabel('s(t)'),xlabel('time')
subplot(3,1,2),plot(t,ds,LineWidth=2),ylabel('$\dot{s}(t)
$',Interpreter='latex'),xlabel('time')
subplot(3,1,3),plot(t,dds,LineWidth=2),ylabel('$\ddot{s}(t)
$',Interpreter='latex'),xlabel('time')
```




```
X = X0+s*(XT-X0);
dX = ds*(XT-X0);
ddX = dds*(XT-X0);

figure,sgtitle('Resulting cubic trajectory based on time-scaling function')
subplot(3,1,1),plot(t,X,LineWidth=2),ylabel('X(t)'),xlabel('time')
subplot(3,1,2),plot(t,dX,LineWidth=2),ylabel('$\dot{X}(t)
$',Interpreter='latex'),xlabel('time')
subplot(3,1,3),plot(t,ddX,LineWidth=2),ylabel('$\ddot{X}(t)
$',Interpreter='latex'),xlabel('time')
```



7.3.3 Quintic polynomial interpolation

Because third-order time scaling does not constrain the endpoint path accelerations $\ddot{s}(0)$ and $\ddot{s}(T)$ to be zero, the robot is expected to achieve a discontinuous jump in acceleration at both $t = 0$ and $t = T$ as seen above. This implies an infinite jerk, the rate of change of acceleration, which may cause vibration of the robot and also potentially damage the actuators.

One way to circumvent requesting an infinite jerk is to employ constraints on the endpoint accelerations, namely $\ddot{s}(0) = \ddot{s}(T) = 0$. The adding of these two constraints necessitates two additional design freedoms in the polynomial, which increases the degree of the required polynomial to 5. In other words, a quintic polynomial of the form

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

is required. The velocity, acceleration, and jerk profiles follow as

$$\dot{s}(t) = a_1 + 2a_2 t + 3a_3 t^2 + 4a_4 t^3 + 5a_5 t^4,$$

$$\ddot{s}(t) = 2a_2 + 6a_3 t + 12a_4 t^2 + 20a_5 t^3,$$

$$\dddot{s}(t) = 6a_3 + 24a_4 t + 60a_5 t^2.$$

We can make use of the six endpoint constraints (two position, two velocity, two acceleration) to solve for $\{a_0, a_1, a_2, a_3, a_4, a_5\}$ uniquely, which yields a smoother motion with a higher maximum velocity than a cubic time scaling:

$$s(0) = a_0 = 0,$$

$$s(T) = a_0 + a_1 T + a_2 T^2 + a_3 T^3 + a_4 T^4 + a_5 T^5 = 1,$$

$$\dot{s}(0) = a_1 = 0,$$

$$\dot{s}(T) = a_1 + 2a_2 T + 3a_3 T^2 + 4a_4 T^3 + 5a_5 T^4 = 0,$$

$$\ddot{s}(0) = 2a_2 = 0,$$

$$\ddot{s}(T) = 2a_2 + 6a_3 T + 12a_4 T^2 + 20a_5 T^3 = 0.$$

The constraints above can be represented in matrix form as

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & T & T^2 & T^3 & T^4 & T^5 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2T & 3T^2 & 4T^3 & 5T^4 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 6T & 12T^2 & 20T^3 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix},$$

or, noting that $a_0 = a_1 = a_2 = 0$, more compactly as

$$\begin{bmatrix} T^3 & T^4 & T^5 \\ 3T^2 & 4T^3 & 5T^4 \\ 6T & 12T^2 & 20T^3 \end{bmatrix} \begin{bmatrix} a_3 \\ a_4 \\ a_5 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}.$$

The coefficients can then be solved as

$$\begin{aligned} \begin{bmatrix} a_3 \\ a_4 \\ a_5 \end{bmatrix} &= \begin{bmatrix} T^3 & T^4 & T^5 \\ 3T^2 & 4T^3 & 5T^4 \\ 6T & 12T^2 & 20T^3 \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \\ &= \begin{bmatrix} \frac{10}{T^3} \\ -\frac{15}{T^4} \\ \frac{6}{T^5} \end{bmatrix}. \end{aligned}$$

```
syms T real
A = [T^3 T^4 T^5;
     3*T^2 4*T^3 5*T^4;
     6*T 12*T^2 20*T^3];
```

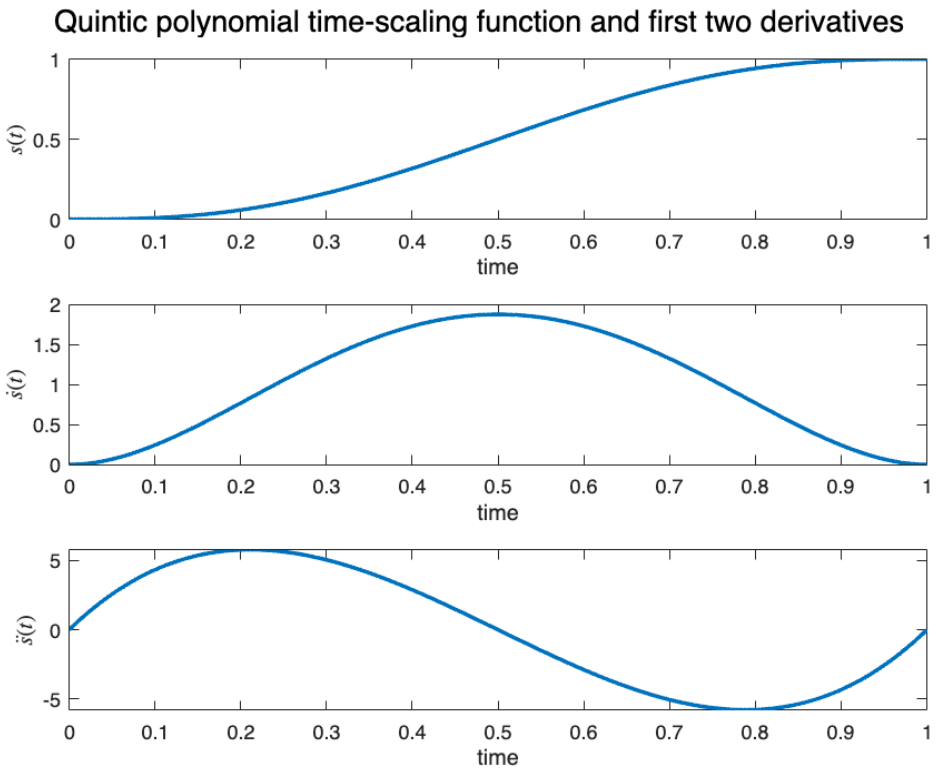
```
B = [1 0 0]';
a_345 = A\B
```

Example: Quintic polynomial interpolation

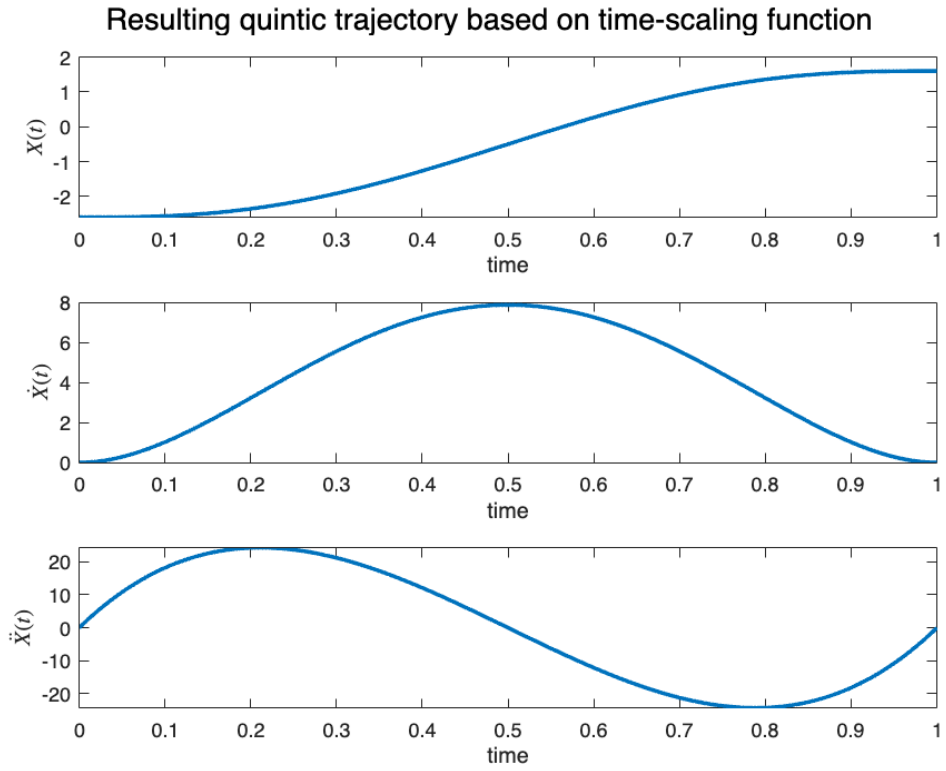
The example below generates a quintic time-scaling function, $s(t)$, and resulting trajectory, $X(t)$. The time-scaling function is only dependent on the choice of completion time, T, whereas the scale of $X(t)$ will depend on the initial and final positions that are specified.

Unlike the acceleration profile for the cubic polynomial, the acceleration profile from the quintic polynomial is now smooth. The maximum obtainable velocity is also increased, when compared to the cubic polynomial.

```
X0 =-2.6; %initial position
XT =1.6; %final position
T = 1;  %completion time

M = [T^3 T^4 T^5;
     3*T^2 4*T^3 5*T^4;
     6*T 12*T^2 20*T^3];

a_345 = M\[1 0 0]';
A = [0 0 0 a_345'];

t = linspace(0,T,1e3);

s = A(1)+A(2)*t+A(3)*t.^2+A(4)*t.^3+A(5)*t.^4+A(6)*t.^5;
ds = A(2)+2*A(3)*t+3*A(4)*t.^2+4*A(5)*t.^3+5*A(6)*t.^4;
dds = 2*A(3)+6*A(4)*t+12*A(5)*t.^2+20*A(6)*t.^3;

figure,sgtitle('Quintic polynomial time-scaling function and first two
derivatives')
subplot(3,1,1),plot(t,s,lineWidth=2),ylabel('$s(t)
$',Interpreter='latex'),xlabel('time')
subplot(3,1,2),plot(t,ds,lineWidth=2),ylabel('$\dot{s}(t)
$',Interpreter='latex'),xlabel('time')
subplot(3,1,3),plot(t,dds,lineWidth=2),ylabel('$\ddot{s}(t)
$',Interpreter='latex'),xlabel('time')
```




```
X = X0+s*(XT-X0);
dX = ds*(XT-X0);
ddX = dds*(XT-X0);

figure,sgtitle('Resulting quintic trajectory based on time-scaling
function')
subplot(3,1,1),plot(t,X,lineWidth=2),ylabel('$X(t)
$',Interpreter='latex'),xlabel('time')
subplot(3,1,2),plot(t,dX,lineWidth=2),ylabel('$\dot{X}(t)
$',Interpreter='latex'),xlabel('time')
subplot(3,1,3),plot(t,ddX,lineWidth=2),ylabel('$\ddot{X}(t)
$',Interpreter='latex'),xlabel('time')
```



7.3.4 Trapezoidal velocity profiles

While the polynomial trajectories are appealing from an energy efficient point of view at sufficiently high degree, they are not time optimal. This is exemplified by the fact that any imposed velocity constraint can at best only momentarily be met at equality momentarily (at the halfway turning point for point to point motion). This means that for almost the entire trajectory, the robot is not being driven at its velocity limit. Similarly, the acceleration will only be maximised at two time points (depending on the order of the polynomial), implying that the acceleration is below its theoretical maximum for every other point on the trajectory. This obviously means that we are not dealing with the time-optimal trajectory given the robot's dynamic constraints.

A **trapezoidal velocity profile** is an alternative approach that blends different polynomial types together during a point to point manoeuvre — where the initial and final velocities are zero. The name originates from the fact that the velocity profile has a trapezoidal shape.

With reference to Figure 7.3, the trajectory is split up into three phases:

- **Phase 1** (*spool up*): Constant acceleration of $\ddot{s} = a$ for duration t_a.
- **Phase 2** (*cruise*): Constant velocity of $\dot{s} = v$ for duration $t_a = T - 2T_a$.
- **Phase 3** (*spool down*): Constant deceleration of $\ddot{s} = -a$ for duration t_a.

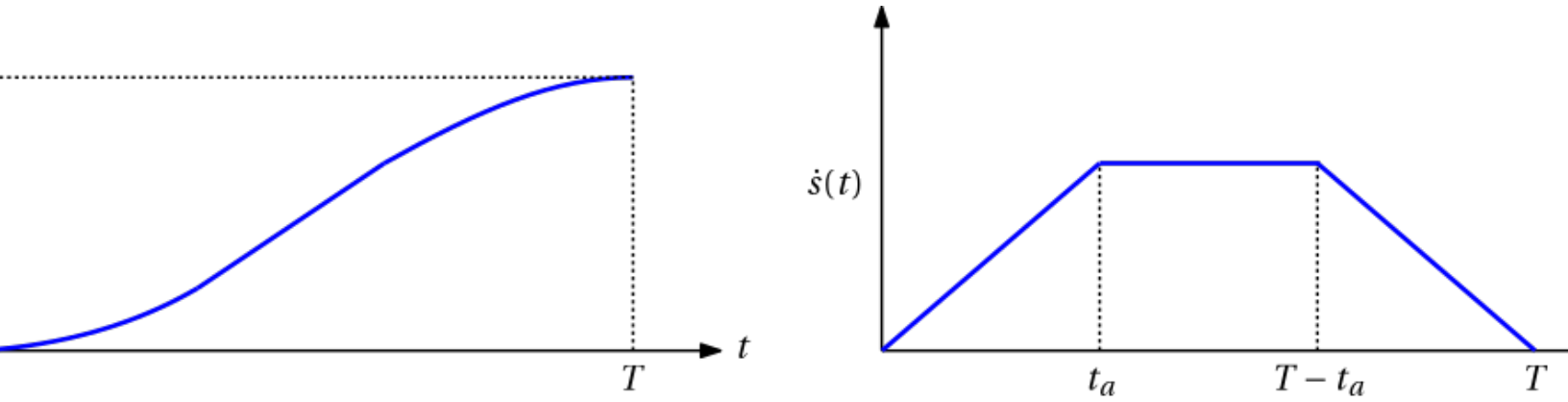


Figure 7.3: Trapezoidal position and velocity profiles

Note that the velocity profile is required to be symmetric about the halfway point of $t = T/2$. The resulting position trajectory is made up of a linear segment (**Phase 2**) that is connected on either side by quadratic segments (**Phase 1** and **Phase 3**). As indicated in Figure 7.2, both the initial and final velocities are required to be zero, with the time durations of **Phase 1** and **Phase 3** also equal.

While the trapezoidal velocity profile is not as smooth as the cubic or quintic counterparts, the benefit lies in the fact that the time scaling can take into account the velocity and acceleration limits using

$$a \leq \ddot{s}_{lim}, \quad v \leq \dot{s}_{lim},$$

where $\dot{s}_{lim} = \dfrac{\dot{X}_{lim}}{|X_T - X_0|}$, and $\ddot{s}_{lim} = \dfrac{\ddot{X}_{lim}}{|X_T - X_0|}$, as shown in Section 7.3.2. It follows that the trapezoidal motion using the largest v and a satisfying

$$v \leq \frac{\dot{X}_{lim}}{|X_T - X_0|},$$

$$a \leq \frac{\ddot{X}_{lim}}{|X_T - X_0|},$$

yields the fast straight-line motion possible.

With reference to Figure 7.2, we see that the time-scaling velocity and acceleration are related to the **Phase 1** end time based on $v = at_a$. We can therefore remove the dependence of $t_a = \dfrac{v}{a}$ when characterising the kinematic profiles. The three phases will therefore have the following timespans:

1. **Phase 1**: $0 \leq t \leq \dfrac{v}{a}$,
2. **Phase 2**: $\dfrac{v}{a} < t \leq T - \dfrac{v}{a}$,

3. **Phase 3**: $T - \frac{v}{a} < t \leq T$.

The acceleration profile over the three phases, as defined above, is given as

$$\ddot{s}(t) = \begin{cases} a, & 0 \leq t \leq \frac{v}{a}, \\ 0, & \frac{v}{a} < t \leq T - \frac{v}{a}. \\ -a, & T - \frac{v}{a} < t \leq T. \end{cases}$$

The corresponding velocity profile is obtained by integrating the acceleration profiles for each phase, whilst taking into account the time parameterisation for each phase, as well as the initial condition for velocity at the start of each phase. This yields

$$\dot{s}(t) = \begin{cases} at, & 0 \leq t \leq \frac{v}{a}, \\ v, & \frac{v}{a} < t \leq T - \frac{v}{a}. \\ a(T-t), & T - \frac{v}{a} < t \leq T. \end{cases}$$

Finally, the time-scaling position profiles are obtained via integration of the velocity profile above, again taking into account the initial position at the start of each phase.

$$s(t) = \begin{cases} \frac{1}{2}at^2, & 0 \leq t \leq \frac{v}{a}, \\ vt - \frac{v^2}{2a}, & \frac{v}{a} < t \leq T - \frac{v}{a}. \\ \frac{2avT - 2v^2 - a^2(t-T)^2}{2a}, & T - \frac{v}{a} < t \leq T. \end{cases}$$

In order for the trapezoidal velocity profile to reach the velocity limit of v, we require $t_a = \frac{v}{a} \leq \frac{T}{2}$, based on the symmetry of the profile. Noting that the final time-scaling position is required to be $s(T) = 1$, we can deduce that

$$s(T) = \frac{2avT - 2v^2}{2a} = vT - \frac{v^2}{a} = 1.$$

We then solve for T as

$$T = \frac{1}{v} + \frac{v}{a}$$

and rewrite the constraint of $t_a = \frac{v}{a} \leq \frac{T}{2}$ as

$$\begin{aligned}\frac{v}{a} &\leq \frac{T}{2},\\ &\leq \frac{1}{2v}+\frac{v}{2a},\\ &\leq \frac{a+v^2}{2av}.\end{aligned}$$

which, after some simplification, yields the requirement of

$$\frac{v^2}{a} \leq 1.$$

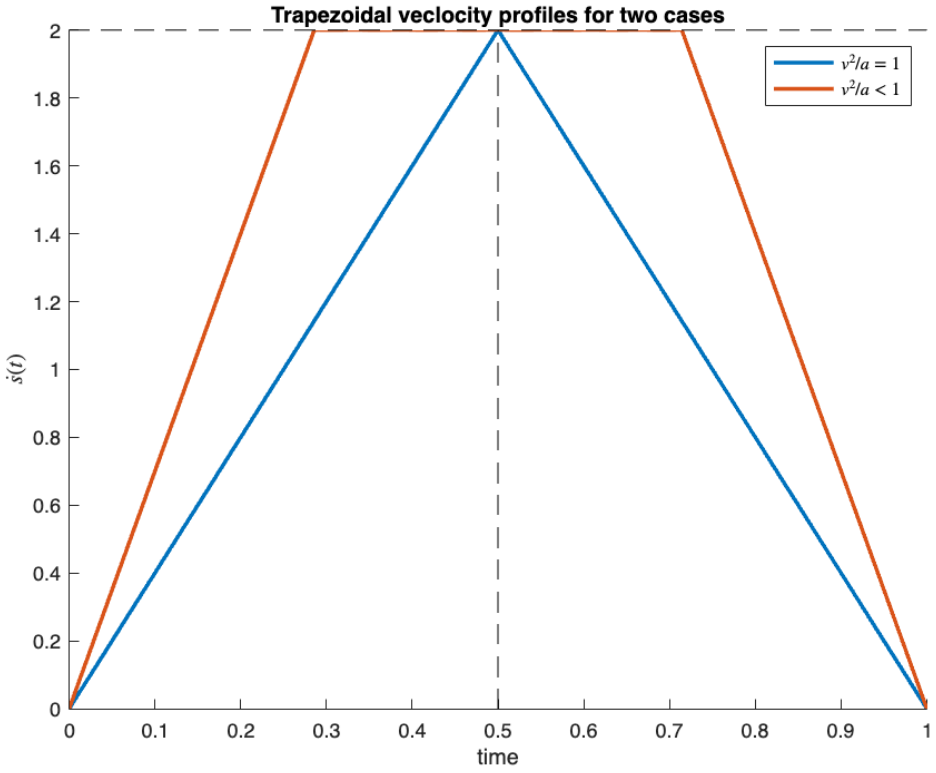
The requirement above is necessary for the trapezoidal velocity profile to (a) reach the velocity limit, and (b) be feasible (can be completed in time). It follows that:

- if $\frac{v^2}{a} < 1$, the velocity limit will be reached for all time points $\frac{v}{a} < t \leq T - \frac{v}{a}$, and the velocity profile will have a trapezoidal shape.
- if $\frac{v^2}{a} = 1$, the velocity limit will be reached for a single time instance, at $t = \frac{T}{2}$, and the velocity profile will have a two-phase triangular shape.
- if $\frac{v^2}{a} > 1$, the velocity limit will never be reached and the position trajectory will not be feasible.

The first two cases can be shown using the code block below.

```
clear
v = 2;
T = 1;
a = [4 7];

ds = zeros(3e3,2);
%velocity
for i = 1:length(a)
    ta = v/a(i);

    t1 = linspace(0,ta,1e3);
    t2 = linspace(ta,T-ta,1e3);
    t3 = linspace(T-ta,T,1e3);

    t(i,:) = [t1 t2 t3];

    ds_p1 = a(i)*t1;
    ds_p2 = v*ones( 1,length(t2) );
    ds_p3 = a(i)*(T-t3);

    ds(:,i) = [ds_p1'; ds_p2'; ds_p3'];
end
```

```
figure,hold on
plot(t(1,:),ds(:,1),lineWidth=2)
plot(t(2,:),ds(:,2),lineWidth=2)
xlabel('time'),ylabel('$\dot{s}(t)$',Interpreter='latex')
plot(t,v*ones(1,length(t)),'--k'),axis tight
line([0.5 0.5],[0,v],"lineStyle","--","color","k")
legend('$v^2/a=1$','$v^2/a<1$','Interpreter','latex')
title('Trapezoidal veclocity profiles for two cases')
```




```
```

Since only two of v, a, and T can be chosen independently, we have three options:

Option 1: Choosing velocity and acceleration

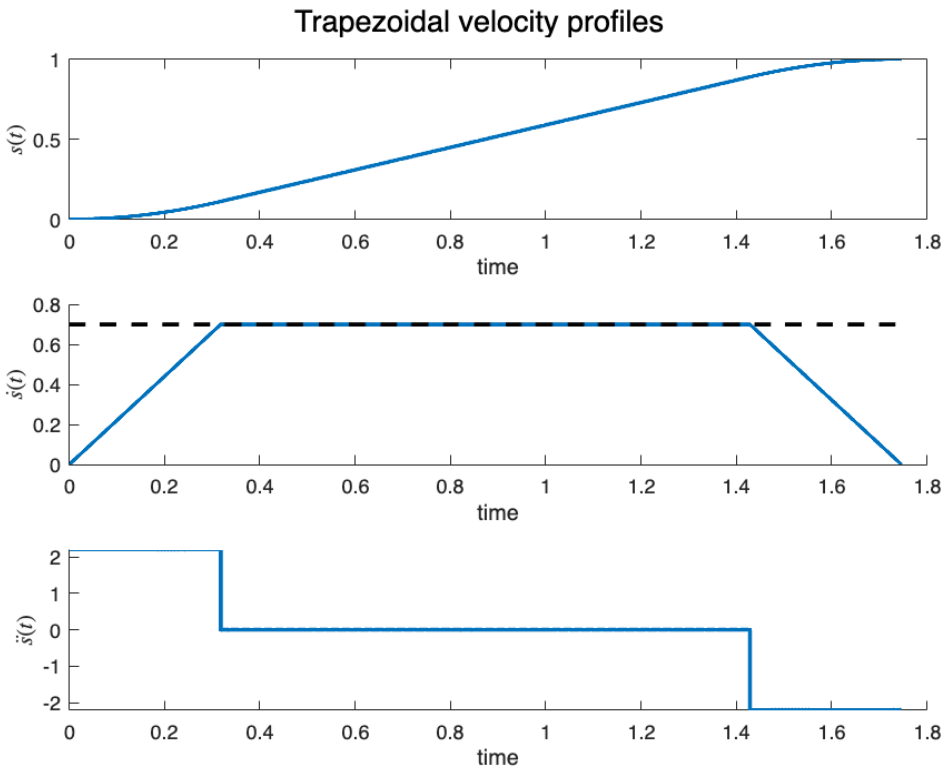
Choose v and a such that $v^2/a \leq 1$, ensuring a feasible profile, and solve for $s(T) = 1$ to obtain

$$T = \frac{a + v^2}{va}.$$

If v and a correspond to the highest possible velocities and accelerations, this is the minimum possible time for the motion.

```
a =2.2;
v =0.7;

if( (v^2/a>1) )
```

```
    fprintf('Select v and T such that v^2/a<=1')
    return
end

T = (a+v^2)/(v*a);

ta = v/a;

t1 = linspace(0,ta,1e3);
t2 = linspace(ta,T-ta,1e3);
t3 = linspace(T-ta,T,1e3);
t = [t1 t2 t3];

%acceleration
dds_p1 = a*ones( 1,length(t1) );
dds_p2 = zeros( 1,length(t2) );
dds_p3 = -a*ones( 1,length(t3) );
dds = [dds_p1 dds_p2 dds_p3];
%velocity
ds_p1 = a*t1;
ds_p2 = v*ones( 1,length(t2) );
ds_p3 = a*(T-t3);
ds = [ds_p1 ds_p2 ds_p3];
%position
s_p1 = 0.5*a*t1.^2;
s_p2 = v*t2-v^2/(2*a);
s_p3 = (2*a*v*T - 2*v^2-a^2*(t3-T).^2)/(2*a);
s = [s_p1 s_p2 s_p3];

figure
subplot(3,1,1),sgtitle('Trapezoidal velocity profiles when specifying
acceleration and velocity limits')
plot(t,s,lineWidth=2),xlabel('time'),ylabel('$s(t)$',Interpreter='latex')
subplot(3,1,2),hold on
plot(t,ds,lineWidth=2),xlabel('time'),ylabel('$\dot{s}(t)
$',Interpreter='latex')
plot(t,v*ones(1,length(t)),'--k',lineWidth=2)
subplot(3,1,3),hold on
plot(t,dds,lineWidth=2),xlabel('time'),ylabel('$\ddot{s}(t)
$',Interpreter='latex')
```



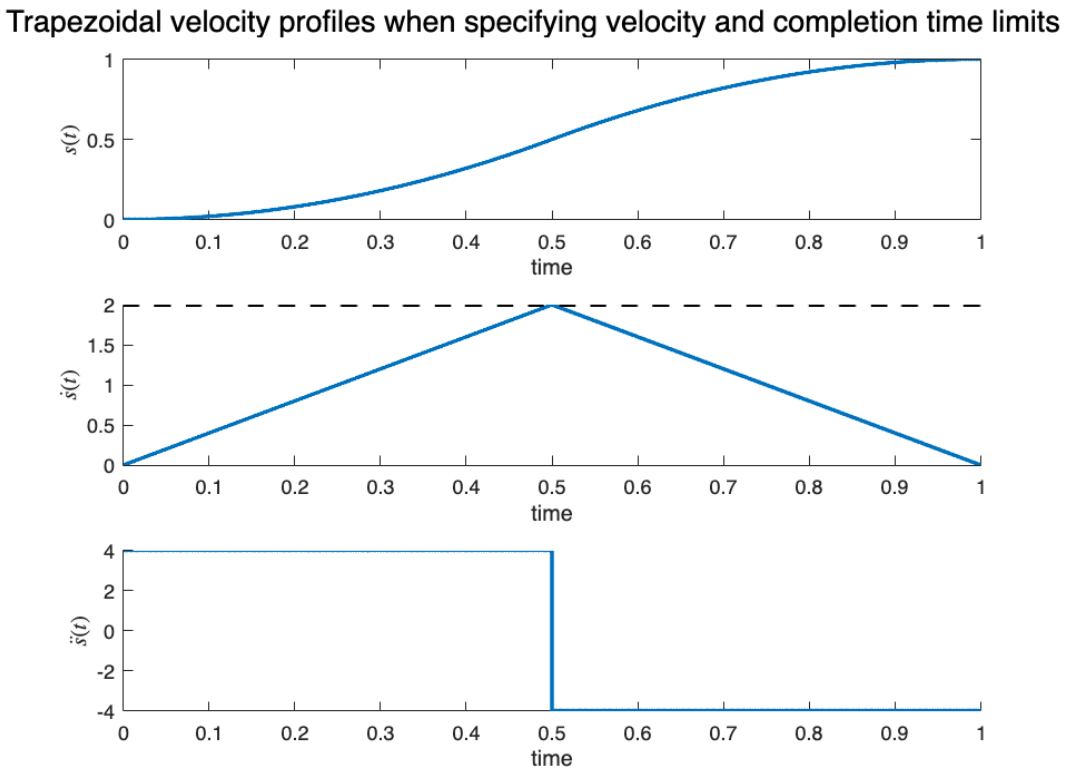
Option 2: Choosing velocity and completion time

Choose v and T such that $1 < vT \leq 2$, ensuring a feasible profile and that the top speed v is sufficient to reach $s(T) = 1$. The acceleration is then given as

$$a = \frac{v^2}{vT - 1}.$$

```
v =2;
T =1;

if( (v*T<1) || (v*T>2) )
    disp("Select v and T such that 1<vT<2")
    return
end

a = v^2/(v*T-1);

ta = v/a;

t1 = linspace(0,ta,1e3);
t2 = linspace(ta,T-ta,1e3);
t3 = linspace(T-ta,T,1e3);
t = [t1 t2 t3];

%acceleration
```

```
dds_p1 = a*ones( 1,length(t1) );
dds_p2 = zeros( 1,length(t2) );
dds_p3 = -a*ones( 1,length(t3) );
dds = [dds_p1 dds_p2 dds_p3];
%velocity
ds_p1 = a*t1;
ds_p2 = v*ones( 1,length(t2) );
ds_p3 = a*(T-t3);
ds = [ds_p1 ds_p2 ds_p3];
%position
s_p1 = 0.5*a*t1.^2;
s_p2 = v*t2-v^2/(2*a);
s_p3 = (2*a*v*T - 2*v^2-a^2*(t3-T).^2)/(2*a);
s = [s_p1 s_p2 s_p3];

figure
subplot(3,1,1),sgtitle('Trapezoidal velocity profiles when specifying
velocity and completion time limits')
plot(t,s,lineWidth=2),xlabel('time'),ylabel('$s(t)$',Interpreter='latex')
subplot(3,1,2),hold on
plot(t,ds,lineWidth=2),xlabel('time'),ylabel('$\dot{s}(t)
$',Interpreter='latex')
plot(t,v*ones(1,length(t)),'--k',lineWidth=2)
subplot(3,1,3),hold on
plot(t,dds,lineWidth=2),xlabel('time'),ylabel('$\ddot{s}(t)
$',Interpreter='latex')
```



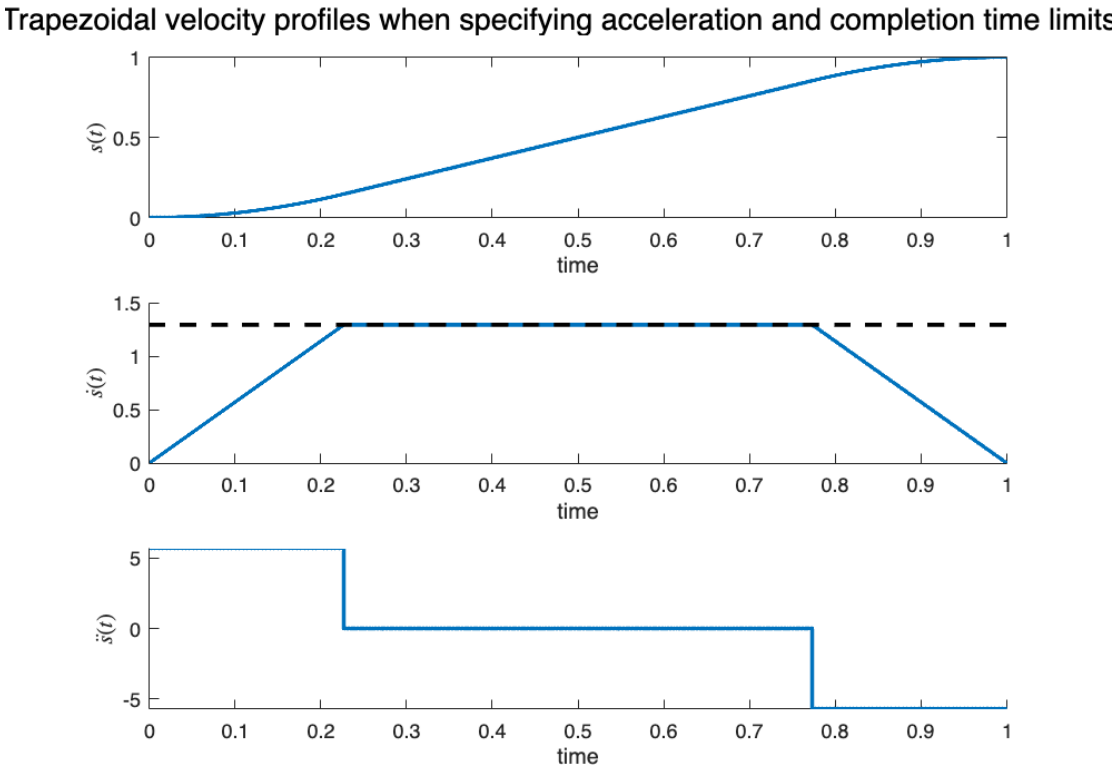
Option 3: Choosing acceleration and completion time

Choose a and T such that $aT^2 \geq 4$, ensuring that the motion is feasible, and solve for $s(T) = 1$ for v:

$$v = \frac{1}{2}\left(aT - \sqrt{a}\sqrt{aT^2 - 4}\right).$$

```
a =5.7;
T =1;

if( a*T^2<4 )
    disp("Select a and T such that aT^2<4")
    return
end

v = 0.5*( a*T-sqrt(a)*sqrt(a*T^2-4) );

ta = v/a;

t1 = linspace(0,ta,1e3);
t2 = linspace(ta,T-ta,1e3);
t3 = linspace(T-ta,T,1e3);
t = [t1 t2 t3];

%acceleration
dds_p1 = a*ones( 1,length(t1) );
dds_p2 = zeros( 1,length(t2) );
dds_p3 = -a*ones( 1,length(t3) );
dds = [dds_p1 dds_p2 dds_p3];
%velocity
ds_p1 = a*t1;
ds_p2 = v*ones( 1,length(t2) );
ds_p3 = a*(T-t3);
ds = [ds_p1 ds_p2 ds_p3];
%position
s_p1 = 0.5*a*t1.^2;
s_p2 = v*t2-v^2/(2*a);
s_p3 = (2*a*v*T - 2*v^2-a^2*(t3-T).^2)/(2*a);
s = [s_p1 s_p2 s_p3];

figure
subplot(3,1,1),sgtitle('Trapezoidal velocity profiles when specifying
acceleration and completion time limits')
plot(t,s,lineWidth=2),xlabel('time'),ylabel('$s(t)$',Interpreter='latex')
subplot(3,1,2),hold on
plot(t,ds,lineWidth=2),xlabel('time'),ylabel('$\dot{s}(t)
$',Interpreter='latex')
plot(t,v*ones(1,length(t)),'--k',lineWidth=2)
subplot(3,1,3),hold on
```

```
plot(t,dds,lineWidth=2),xlabel('time'),ylabel('$\ddot{s}(t)
$',Interpreter='latex')
```



7.4 Polynomial splines

Polynomial splines, such as a cubic spline, is a method to find a smooth trajectory that intersects a number of desired waypoints — often referred to as via points. Conceptually, if we were given a start point, and end point, and some finite amount of intermediate waypoints, we can then use polynomial interpolation between each set of adjacent points. These piece-wise polynomials are then blended together to form a single trajectory that is made of a set of polynomials. The result is a continuous trajectory that moves through each specified point, as shown in Figure 7.4.

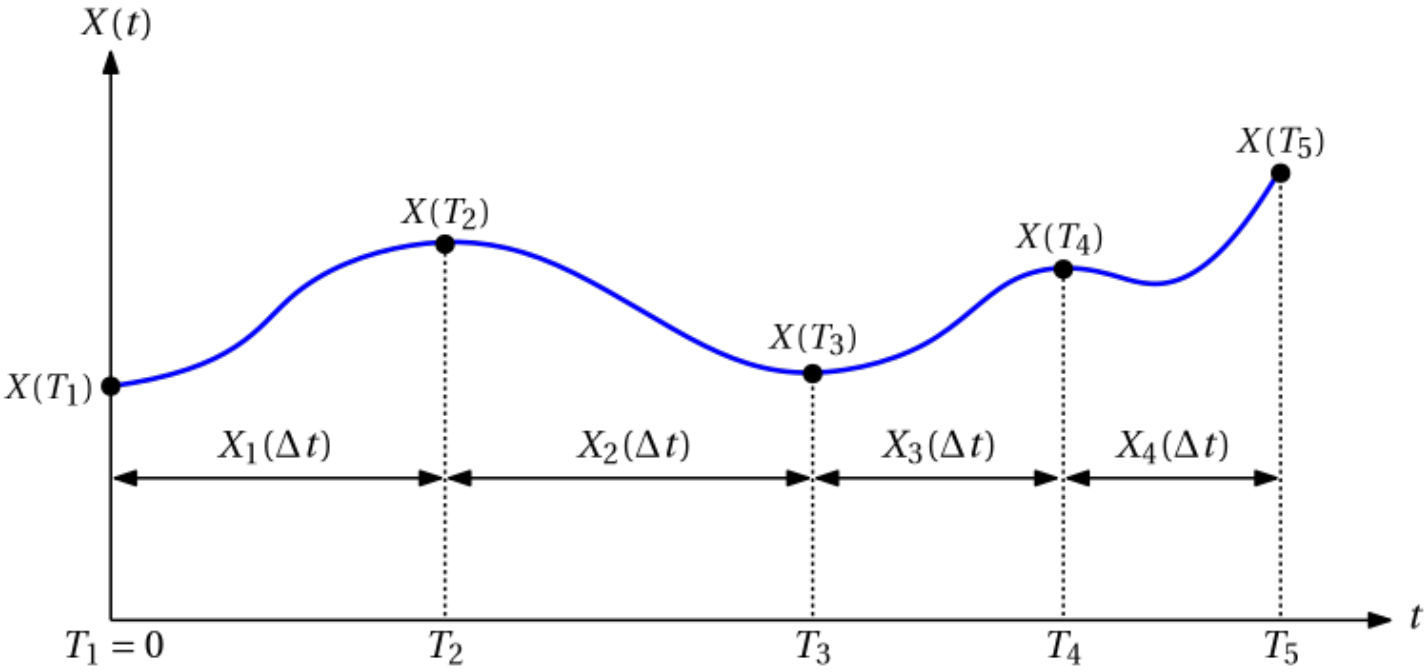


Figure 7.4: An example of a cubic spline trajectory with five waypoints.

This type of trajectory generation method is useful in applications that require surveying or task-specific motion. For example, if a quadrotor is required to inspect different specific location on a map, we are not too concerned with how it behaves between these specified locations (our waypoints), aside from the quadrotor to be energy efficient. So we can define a polynomial spline that intersects all the waypoints with a piece-wise set of polynomials.

We start by considering a generalised trajectory with N waypoints, and a start time and end time of $T_1 = 0$ and $T_N = T$, respectively. $T_1 = 0$ corresponds to the start time at waypoint 1, whereas $T_2 > 0$ is the time requirement at waypoint 2. For each waypoint time T_i, there will be a corresponding specified position, $X(T_i)$.

The polynomial is individually assessed between each moving pair of waypoints. Specifically, we will define a polynomial between waypoint 1 and 2, followed by a *different* polynomial that joins waypoint 2 and 3, and so on. The N waypoints results in $N-1$ polynomials to be determined, with a segment duration equal to $\Delta T_i = T_{i+1} - T_i$, where $i \in \{1, 2, ..., N-1\}$.

In the case of cubic polynomials, four parameters need to be determined in order to fully specify a segment trajectory, as shown in Section 7.3.2. This means that a path with $N-1$ polynomials will require $4(N-1)$ parameters to be determined in order to find all segment polynomials. Instead of considering a time-scaling of $s(t)$, we will directly design for trajectory $X(t)$. The cubic trajectory during segment j follows as

$$X_i(\Delta t) = a_{i_0} + a_{i_1}\Delta t + a_{i_2}\Delta t^2 + a_{i_3}\Delta t^3,$$

where $0 \leq \Delta t \leq \Delta T_i$. Note that there is no loss of generality by expressing the independent variable as incremental time, Δt, instead of t, as in Section 7.3.2. The only difference is that the polynomial coefficients will change accordingly.

To ensure sufficient smoothness, the position at the end of trajectory segment i should be equivalent to the position at the beginning of trajectory segment $i+1$, which requires that

$$X_i(\Delta T_i) = X(T_{i+1}) = X_{i+1}(0).$$

Similarly, continuity of the velocity at each waypoint necessitates

$$\dot{X}_i(\Delta T_i) = \dot{X}(T_{i+1}) = \dot{X}_{i+1}(0).$$

Note that unlike the time-scaling cubic in Section 7.3.2, the polynomials used as part of the spline approach do not necessarily have start and end velocities equal to zero (we can actually choose them to be whatever we desire). Additionally, the start and end positions can be arbitrarily defined. The user is also required to specify the time requirement at each waypoint, namely, $\{T_1, T_2, ..., T_N\}$.

The cubic polynomial equation above and associated constraints can be reposed in matrix form as

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & \Delta T_i & \Delta T_i^2 & \Delta T_i^3 \\ 0 & 1 & 2\Delta T_i & 3\Delta T_i^2 \end{bmatrix} \begin{bmatrix} a_{i_0} \\ a_{i_1} \\ a_{i_2} \\ a_{i_3} \end{bmatrix} = \begin{bmatrix} X(T_i) \\ \dot{X}(T_i) \\ X(T_{i+1}) \\ \dot{X}(T_{i+1}) \end{bmatrix},$$

where $\{a_{i_0}, a_{i_1}, a_{i_2}, a_{i_3}\}$ are determined by

$$_0 = X(T_i), \qquad a_{i_1} = \dot{X}(T_i), \qquad a_{i_2} = \frac{3X(T_{i+1}) - 3X(T_i) - 2\dot{X}(T_i)\Delta T_i - \dot{X}(T_{i+1})\Delta T_i}{\Delta T_i^2}, \qquad a_{i_3} = \frac{2X(T_i) + \left(\dot{X}(T_i) + \dot{X}(T_{i+1})\right)\Delta T_i - 2X(T_{i+1})}{\Delta T_i^3}$$

```
syms X0 dX0 XT dXT dT real
A = [1 0 0 0;
     0 1 0 0;
     1 dT dT^2 dT^3;
     0 1 2*dT 3*dT^2];
B = [X0 dX0 XT dXT]';
a = A\B
```

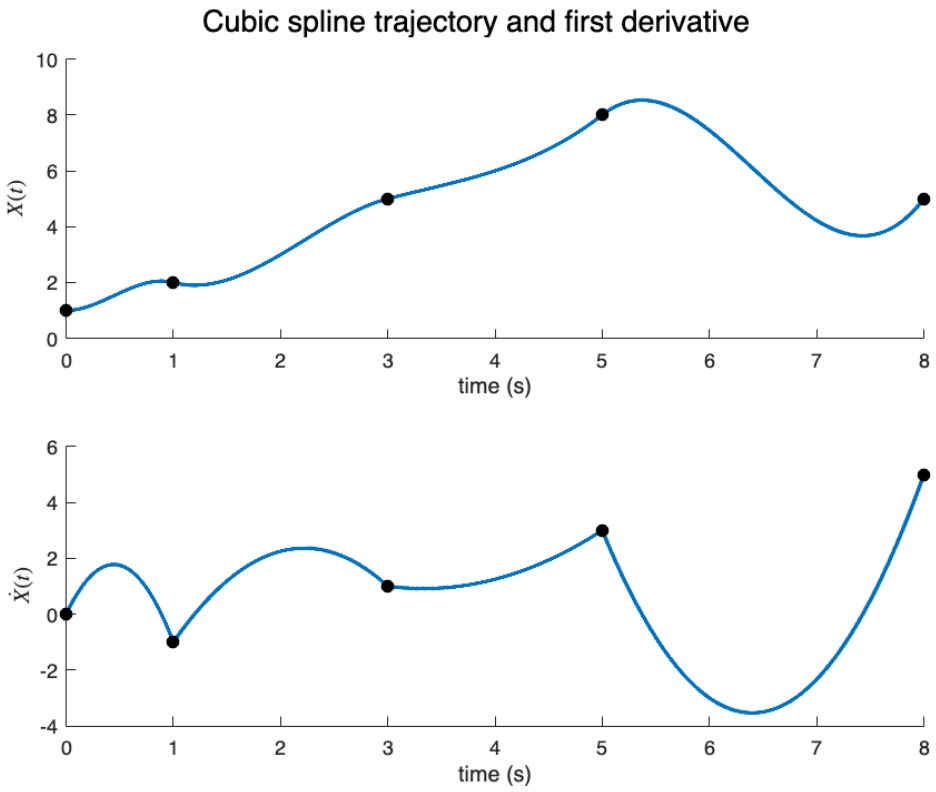
The "efficiency" of the resulting blended trajectory will be dependent on how the waypoint times and velocities are selected. Specifying velocity for intermediate points is not always obvious or intuitive. One can instead only specify the position and time for the via points, and then use heuristic methods to separately find an appropriate velocity that rationally links sequential via point positions (e.g. using simple linear interpolation).

Example: Cubic spline interpolation

The code block below generates a cubic spline for a trajectory with five waypoints. We should notice that:

1. the position behaviour is continuous and smooth throughout the trajectory.
2. the velocity behaviour is continuous but not smooth at the waypoints. This could be resolved if we made use of a quintic spline interpolation.

```
X_wp = [1 2 5 8 5];
dX_wp = [0 -1 1 3 5];
T_wp = [0 1 3 5 8];

N = length(X_wp);

X = zeros(1,(N-1)*1e3);
dX = zeros(1,(N-1)*1e3);
t = zeros(1,(N-1)*1e3);
for i=1:N-1
    dT = T_wp(i+1)-T_wp(i);
    a0 = X_wp(i);
    a1 = dX_wp(i);
    a2 = ( 3*X_wp(i+1)-3*X_wp(i)-2*dX_wp(i)*dT-dX_wp(i+1)*dT )/dT^2;
    a3 = ( 2*X_wp(i)+( dX_wp(i)+dX_wp(i+1) )*dT-2*X_wp(i+1) )/dT^3;

    dt = linspace(0,dT,1e3);
    X((i-1)*1e3+1:1e3*i) = a0+a1*dt+a2*dt.^2+a3*dt.^3;
    dX((i-1)*1e3+1:1e3*i) = a1+2*a2*dt+3*a3*dt.^2;
    t((i-1)*1e3+1:1e3*i) = T_wp(i)+dt;
end

figure,sgtitle('Cubic spline trajectory and first derivative')
subplot(2,1,1),hold on
plot(t,X,LineWidth=2),plot(T_wp,X_wp,'ko',MarkerFaceColor='k')
xlabel('time (s)'),ylabel('$X(t)$',Interpreter='latex')
subplot(2,1,2),hold on
plot(t,dX,LineWidth=2),plot(T_wp,dX_wp,'ko',MarkerFaceColor='k')
xlabel('time (s)'),ylabel('$\dot{X}(t)$',Interpreter='latex')
```



7.5 Task space vs configuration space trajectories

Given that we can describe our robot motion in multiple reference frames, there is an inherent choice related to in which space we specify our trajectory. In almost every conceivable case, we will define our start and end points of the trajectory in the task space. For example, requiring the end-effector of our robot arm to adjust its planar position. However, that does not stop us from formulating the trajectory generation problem in the configuration space. This is beneficial as the actuator limits, such as the slew rate limits, are described in the actuator space, so we can directly impose limits on the trajectory based on our actuator capabilities. This also makes sense from a control perspective if we are making use of sensor information about our actuators (e.g. using an encoder). Our controller would then act to keep our actual positions, for example joint angles, on the defined trajectory. On the other hand, this would result in the corresponding path in the task space becoming essentially arbitrary and unspecified - we would know the start and end point, but the transition between these two points could take any path (within kinematic reason).

We will explore this concept in more detail within a Virtual Lab. An illustration of this is provided in Figure 7.5.

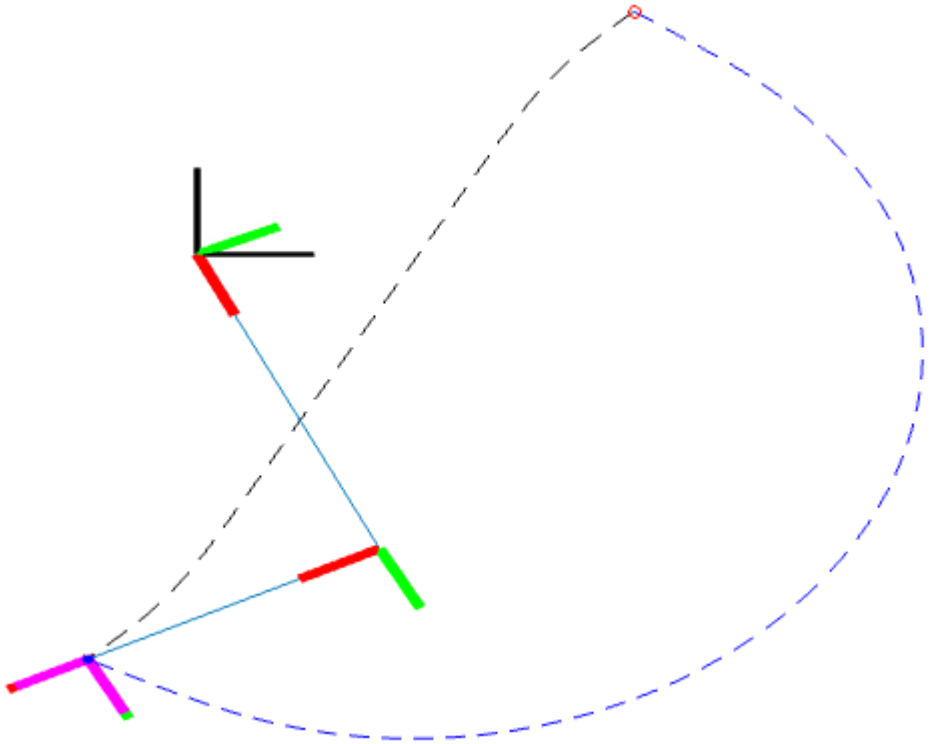


Figure 7.5: Two different end-effector position trajectories that have common start and end points. The black trajectory is the result of a task space trapezoidal velocity profile, whereas the blue trajectory is a result of a joint space trapezoidal velocity profile.