# Modern Robotics: Mechanics, Planning, and Control
# Code Library

October 26, 2016

## Introduction

The code library accompanying *Modern Robotics: Mechanics, Planning, and Control* is written for MATLAB, Mathematica, and Python, and originates from students' solutions to programming assignments in courses using material from the book. The following students made significant contributions to the code: Mikhail Todes, Huan Weng, Matthew Collins, Mojtaba Mozaffar, Chang Liu, and Wentao Chen.

The code is commented and mostly self-explanatory. The primary purpose of the provided software is to be easy to read and educational, reinforcing the concepts in the book. The code is optimized neither for efficiency nor robustness, nor does it do full error-checking on its inputs. This document provides an overview of the available functions using MATLAB syntax. Functions are organized according to the chapter in which they are introduced in the book. Basic functions, such as functions to calculate the magnitude of a vector, normalize a vector, and perform matrix operations such as multiplication and inverses, are not documented here.

Notation that is used throughout this document is summarized below.

| **Math symbol** | **Computer variable** | **Description** |
|---|---|---|
| $R$ | `R` | $3 \times 3$ rotation matrix in $SO(3)$. |
| $\omega$ | `omg` | 3-vector angular velocity. |
| $\hat{\omega}$ | `omghat` | 3-vector unit rotation axis or unit angular velocity. |
| $\theta$ | `theta` | Angle of rotation about an axis or distance traveled along a screw axis. |
| $\hat{\omega}\theta$ | `expc3` | 3-vector of exponential coordinates for rotation. |
| $[\omega], [\hat{\omega}\theta]$ | `so3mat` | $3 \times 3$ $so(3)$ representation of $\omega$ or $\hat{\omega}\theta$. |
| $p$ | `p` | 3-vector for a position in space. |
| $T$ | `T` | $4 \times 4$ transformation matrix in $SE(3)$ corresponding to $(R, p)$. |
| $[\mathrm{Ad}_T]$ | `AdT` | $6 \times 6$ matrix adjoint representation of $T \in SE(3)$. |
| $v$ | `v` | 3-vector linear velocity. |
| $\mathcal{V}$ | `V` | 6-vector twist $(\omega, v)$. |
| $\mathcal{S}$ | `S` | A normalized 6-vector screw axis $(\omega, v)$, where (a) $\|\omega\| = 1$ or (b) $\|\omega\| = 0$ and $\|v\| = 1$. |
| $\mathcal{S}\theta$ | `expc6` | 6-vector of exponential coordinates for rigid-body motion. |
| $[\mathcal{V}], [\mathcal{S}\theta]$ | `se3mat` | $4 \times 4$ $se(3)$ representation of $\mathcal{V}$ or $\mathcal{S}\theta$. |
| $M$ | `M` | End-effector configuration in $SE(3)$ when manipulator is at its zero position. |
| $\mathcal{B}_i$ | `Blist` | $\mathcal{B}_i$ is the screw axis of the $i$th joint expressed in the end-effector frame when the manipulator is at the zero position. `Blist` is a list of all the joint screw axes for the manipulator, $i = 1 \dots n$. |
| $\mathcal{S}_i$ | `Slist` | $\mathcal{S}_i$ is the screw axis of the $i$th joint expressed in the space frame when the manipulator is at the zero position. `Slist` is a list of all the joint screw axes for the manipulator, $i = 1 \dots n$. |

| | | |
|---|---|---|
| $J_b, J_s$ | `Jb`, `Js` | The $6 \times n$ manipulator Jacobian for a robot with $n$ joints, expressed in the end-effector frame ($J_b$) or the space frame ($J_s$). |
| $\epsilon_\omega$ | `eomg` | A small positive tolerance on the end-effector orientation error when calculating numerical inverse kinematics. |
| $\epsilon_\nu$ | `ev` | A small positive tolerance on the end-effector linear position error when calculating numerical inverse kinematics. |
| $\theta_0$ | `thetalist0` | A list of joint variables that serve as an initial guess for the inverse kinematics solution. |
| $\theta_i$ | `thetalist` | $\theta_i$ is the joint variable for joint $i$, and `thetalist` is $\theta = (\theta_1, \ldots, \theta_n)$. |
| | `thetamat` | An $N \times n$ matrix where each row represents $\theta$ one timestep after the row preceding it in the matrix. |
| $\dot{\theta}_i$ | `dthetalist` | $\dot{\theta}_i$ is the rate of change of joint variable $i$, and `dthetalist` is $\dot{\theta} = (\dot{\theta}_1, \ldots, \dot{\theta}_n)$. |
| | `dthetamat` | An $N \times n$ matrix where each row represents $\dot{\theta}$ one timestep after the row preceding it in the matrix. |
| $\ddot{\theta}_i$ | `ddthetalist` | $\ddot{\theta}_i$ is the acceleration of joint $i$, and `ddthetalist` is $\ddot{\theta} = (\ddot{\theta}_1, \ldots, \ddot{\theta}_n)$. |
| | `dthetamat` | An $N \times n$ matrix where each row represents $\ddot{\theta}$ one timestep after the row preceding it in the matrix. |
| $\mathfrak{g}$ | `g` | 3-vector for gravitational acceleration. |
| $\tilde{\mathfrak{g}}$ | `gtilde` | A possibly incorrect model for $\mathfrak{g}$ used by a controller. |
| $M_{i-1,i}$ | `Mlist` | $M_{i-1,i} \in SE(3)$ is the configuration of manipulator link $i$ relative to link $i-1$ when the manipulator is at its zero position. The link frames are defined at the link centers of mass. `Mlist` is a list of all $M_{i-1,i}$ for $i = 1 \ldots n+1$. The frame $\{n+1\}$ is the end-effector frame, and it is fixed relative to the frame $\{n\}$ of the last link. It simply offers the opportunity to define an end-effector frame other than at the center of mass of the last link. |
| | `Mtildelist` | A possibly incorrect model for `Mlist` used by a controller. |
| $\mathcal{F}_{\text{tip}}$ | `Ftip` | 6-vector wrench applied by the manipulator end-effector, expressed in the end-effector frame $\{n+1\}$. |
| | `Ftipmat` | An $N \times 6$ matrix where each row represents $\mathcal{F}_{\text{tip}}$ one timestep after the row preceding it in the matrix. |
| $\mathcal{G}_i$ | `Glist` | $\mathcal{G}_i$ is the $6 \times 6$ spatial inertia matrix for link $i$ of the manipulator, and `Glist` is a list of all $\mathcal{G}_i$ for $i = 1 \ldots n$. |
| | `Gtildelist` | A possibly incorrect model for `Glist` used by a controller. |
| $\tau_i$ | `taulist` | $\tau_i$ is the generalized force applied at joint $i$, and `taulist` is the list of all joint torques $\tau = (\tau_1, \ldots, \tau_n)$. |
| | `taumat` | An $N \times n$ matrix where each row represents $\tau$ one timesetep after the row preceding it in the matrix. |
| $[\text{ad}_\mathcal{V}]$ | `adV` | $6 \times 6$ matrix adjoint representation of $\mathcal{V} \in se(3)$, used to calculate the Lie bracket of two twists, $[\mathcal{V}_1, \mathcal{V}_2] = [\text{ad}_{\mathcal{V}_1}]\mathcal{V}_2$. |
| $T_f$ | `Tf` | The total time of a motion in seconds from rest to rest when calculating trajectories. |
| $\Delta t$ | `dt` | A timestep (e.g., between consecutive rows in a matrix representing a trajectory or force history). |
| $t$ | `t` | The current time. |
| $\theta_{\text{start}}$ | `thetastart` | An $n$-vector of initial joint variables with which to start a trajectory. |

| | | |
|---|---|---|
| $\theta_{\text{end}}$ | `thetaend` | An $n$-vector of final joint variables with which to end a trajectory. |
| $X_{\text{start}}$ | `Xstart` | An initial end-effector configuration $X_{\text{start}} \in SE(3)$ with which to start a trajectory. |
| $X_{\text{end}}$ | `Xend` | A final end-effector configuration $X_{\text{end}} \in SE(3)$ with which to end a trajectory. |
| $e_{\text{int}}$ | `eint` | An $n$-vector of the time-integral of joint errors. |
| $\theta_d$ | `thetalistd` | An $n$-vector of reference joint variables $\theta_d$. |
| | `thetamatd` | An $N \times n$ matrix where each row represents $\theta_d$ one timestep after the row preceding it in the matrix. |
| $\dot{\theta}_d$ | `dthetalistd` | An $n$-vector of reference joint velocities $\dot{\theta}_d$. |
| | `dthetamatd` | An $N \times n$ matrix where each row represents $\dot{\theta}_d$ one timestep after the row preceding it in the matrix. |
| $\ddot{\theta}_d$ | `ddthetalistd` | An $n$-vector of reference joint accelerations $\ddot{\theta}_d$. |
| | `dthetamatd` | An $N \times n$ matrix where each row represents $\ddot{\theta}_d$ one timestep after the row preceding it in the matrix. |
| $K_p$ | `Kp` | A scalar feedback proportional gain. |
| $K_i$ | `Ki` | A scalar feedback integral gain. |
| $K_d$ | `Kd` | A scalar feedback derivative gain. |
| | `intRes` | The number of integration steps during each timestep $\Delta t$. The value must be a positive integer. Larger values result in slower simulations but less accumulation of integration error. |

## Chapter 3: Rigid-Body Motions

```
invR = RotInv(R)
```

Input:
`R`: Rotation matrix.
Output:
`invR`: The inverse of `R`.

For efficiency, the inverse is calculated as the transpose rather than a matrix inverse.

```
so3mat = VecToso3(omg)
```

Input:
`omg`: A 3-vector.
Output:
`so3mat`: The corresponding $3 \times 3$ skew-symmetric matrix in $so(3)$.

```
omg = so3ToVec(so3mat)
```

Input:
`so3mat`: A $3 \times 3$ skew-symmetric matrix (an element of $so(3)$).

Output:

`omg`: The corresponding 3-vector.

```
[omghat,theta] = AxisAng3(expc3)
```

Input:

`expc3`: A 3-vector of exponential coordinates for rotation $\hat{\omega}\theta$.

Output:

`omghat`: The corresponding unit rotation axis $\hat{\omega}$.

`theta`: The corresponding rotation angle $\theta$.

```
R = MatrixExp3(expc3)
```

Input:

`expc3`: A 3-vector of exponential coordinates $\hat{\omega}\theta$.

Output:

`R`: The $R' \in SO(3)$ that is achieved by rotating about $\hat{\omega}$ by $\theta$ from an initial orientation $R = I$.

```
expc3 = MatrixLog3(R)
```

Input:

`R`: Rotation matrix.

Output:

`expc3`: The corresponding 3-vector of exponential coordinates $\hat{\omega}\theta$.

Note that the `MatrixLog3` function returns exponential coordinates, not the true matrix log, which would be the $so(3)$ representation of the exponential coordinates.

```
T = RpToTrans(R,p)
```

Input:

`R`: Rotation matrix.

`p`: A position $p \in \mathbb{R}^3$.

Output:

`T`: The corresponding homogeneous transformation matrix $T \in SE(3)$.

```
[R,p] = TransToRp(T)
```

Input:

`T`: Transformation matrix.

Output:

`R`: The corresponding rotation matrix.

`p`: The corresponding position.

```
invT = TransInv(T)
```

Input:
`T`: Transformation matrix.
Output:
`invT`: Inverse of `T`.

Uses the structure of transformation matrices to avoid taking a matrix inverse, for efficiency.

```
se3mat = VecTose3(V)
```

Input:
`V`: A 6-vector (representing a twist, for example).
Output:
`se3mat`: The corresponding $4 \times 4$ $se(3)$ matrix.

```
V = se3ToVec(se3mat)
```

Input:
`se3mat`: A $4 \times 4$ $se(3)$ matrix.
Output:
`V`: The corresponding 6-vector.

```
AdT = Adjoint(T)
```

Input:
`T`: Transformation matrix.
Output:
`AdT`: The corresponding $6 \times 6$ adjoint representation $[\mathrm{Ad}_T]$.

```
S = ScrewToAxis(q,s,h)
```

Input:
`q`: A point $q \in \mathbb{R}^3$ lying on the screw axis.
`s`: A unit vector $\hat{s} \in \mathbb{R}^3$ in the direction of the screw axis.
`h`: The pitch $h \in \mathbb{R}$ (linear velocity divided by angular velocity) of the screw axis.
Output:
`S`: The corresponding normalized screw axis $\mathcal{S} = (\omega, v)$.

```
[S,theta] = AxisAng6(expc6)
```

Input:
`expc6`: A 6-vector of exponential coordinates for rigid-body motion, $\mathcal{S}\theta$.
Output:
`S`: The corresponding normalized screw axis $\mathcal{S}$.
`theta`: The distance traveled along/about $\mathcal{S}$.

```
T = MatrixExp6(expc6)
```

Input:
`expc6`: A 6-vector of exponential coordinates for rigid-body motion, $\mathcal{S}\theta$.
Output:
`T`: The $T' \in SE(3)$ that is achieved by traveling along/about the screw axis $\mathcal{S}$ a distance $\theta$ from an initial configuration $T = I$.

```
expc6 = MatrixLog6(T)
```

Input:
`T`: Transformation matrix.
Output:
`expc6`: The corresponding 6-vector of exponential coordinates $\mathcal{S}\theta$.

Note that the `MatrixLog6` function returns exponential coordinates, not the true matrix log, which would be the $se(3)$ representation of the exponential coordinates.

## Chapter 4: Forward Kinematics

```
T = FKinBody(M,Blist,thetalist)
```

Input:
`M`: The home configuration of the end-effector.
`Blist`: The joint screw axes in the end-effector frame when the manipulator is at the home position.
`thetalist`: A list of joint coordinate values.
Output:
`T`: The $T \in SE(3)$ representing the end-effector frame when the joints are at the specified coordinates.

```
T = FKinSpace(M,Slist,thetalist)
```

Input:

M: The home configuration of the end-effector.

Slist: The joint screw axes in the space frame when the manipulator is at the home position.

thetalist: A list of joint coordinate values.

Output:

T: The $T \in SE(3)$ representing the end-effector frame when the joints are at the specified coordinates.

## Chapter 5: Velocity Kinematics and Statics

```
Jb = JacobianBody(Blist,thetalist)
```

Input:

Blist: The joint screw axes in the end-effector frame when the manipulator is at the home position.

thetalist: A list of joint coordinate values.

Output:

Jb: The corresponding body Jacobian $J_b(\theta) \in \mathbb{R}^{6\times n}$.

```
Js = JacobianSpace(Slist,thetalist)
```

Input:

Slist: The joint screw axes in the space frame when the manipulator is at the home position.

thetalist: A list of joint coordinate values.

Output:

Js: The corresponding space Jacobian $J_s(\theta) \in \mathbb{R}^{6\times n}$.

## Chapter 6: Inverse Kinematics

```
[thetalist,success] = IKinBody(Blist,M,T,thetalist0,eomg,ev)
```

Input:

Blist: The joint screw axes in the end-effector frame when the manipulator is at the home position.

M: The home configuration of the end-effector.

T: The desired end-effector configuration $T_{sd}$.

thetalist0: An initial guess $\theta_0 \in \mathbb{R}^n$ that is "close" to satisfying $T(\theta_0) = T_{sd}$.

eomg: A small positive tolerance on the end-effector orientation error. The returned joint variables must give an end-effector orientation error less than $\epsilon_\omega$.

ev: A small positive tolerance on the end-effector linear position error. The returned joint variables must give an end-effector position error less than $\epsilon_\nu$.

Output:

thetalist: Joint variables that achieve T within the specified tolerances.

success: A logical value where TRUE (1) means that the function found a solution and FALSE (0) means that it ran through the set number of maximum iterations without finding a solution within the tolerances $\epsilon_\omega$ and $\epsilon_\nu$.

The algorithm uses an iterative Newton-Raphson root-finding method starting from the initial guess thetalist0. The algorithm terminates when the stopping criteria are met or after a maximum number of iterations, whichever comes first. The maximum number of iterations has been hardcoded in as a variable in the function, which can be changed if desired. If the stopping criteria are not met, the function returns the last calculation of thetalist as well as a FALSE value for the success variable.

```
[thetalist,success] = IKinSpace(Slist,M,T,thetalist0,eomg,ev)
```

Equivalent to IKinBody, except the joint screw axes are specified in the space frame.

## Chapter 8: Dynamics of Open Chains

This chapter is concerned with calculating and simulating the dynamics of a serial-chain manipulator with dynamics of the form

$$\tau = M(\theta)\ddot{\theta} + c(\theta, \dot{\theta}) + g(\theta) + J^T(\theta)\mathcal{F}_{\text{tip}}.$$

```
adV = ad(V)
```

Input:
- V: A 6-vector (e.g., a twist).

Output:
- adV: The corresponding $6 \times 6$ matrix $[\text{ad}_\mathcal{V}]$.

Used to calculate the Lie bracket $[\mathcal{V}_1, \mathcal{V}_2] = [\text{ad}_{\mathcal{V}_1}]\mathcal{V}_2$.

```
taulist = InverseDynamics(thetalist,dthetalist,ddthetalist,
             g,Ftip,Mlist,Glist,Slist)
```

Input:
- thetalist: $n$-vector of joint variables $\theta$.
- dthetalist: $n$-vector of joint velocities $\dot{\theta}$.
- ddthetalist: $n$-vector of joint accelerations $\ddot{\theta}$.
- g: Gravity vector $\mathfrak{g}$.
- Ftip: Wrench $\mathcal{F}_{\text{tip}}$ applied by the end-effector expressed in frame $\{n+1\}$.
- Mlist: List of link frames $\{i\}$ relative to $\{i-1\}$ at the home position.
- Glist: Spatial inertia matrices $\mathcal{G}_i$ of the links.
- Slist: Screw axes $\mathcal{S}_i$ of the joints in a space frame.

Output:
- taulist: The $n$-vector $\tau$ of required joint forces/torques.

This function uses forward-backward Newton-Euler iterations.

```
M = MassMatrix(thetalist,Mlist,Glist,Slist)
```

Input:
- `thetalist`: $n$-vector of joint variables $\theta$.
- `Mlist`: List of link frames $\{i\}$ relative to $\{i-1\}$ at the home position.
- `Glist`: Spatial inertia matrices $\mathcal{G}_i$ of the links.
- `Slist`: Screw axes $\mathcal{S}_i$ of the joints in a space frame.

Output:
- `M`: The numerical inertia matrix $M(\theta)$ of an $n$-joint serial chain at the given configuration $\theta$.

This function calls `InverseDynamics` $n$ times, each time passing a $\ddot{\theta}$ vector with a single element equal to one and all other inputs set to zero. Each call of `InverseDynamics` generates a single column of the robot's mass matrix, and these columns are assembled to create the full mass matrix.

```
c = VelQuadraticForces(thetalist,dthetalist,Mlist,Glist,Slist)
```

Input:
- `thetalist`: $n$-vector of joint variables $\theta$.
- `dthetalist`: $n$-vector of joint velocities $\dot{\theta}$.
- `Mlist`: List of link frames $\{i\}$ relative to $\{i-1\}$ at the home position.
- `Glist`: Spatial inertia matrices $\mathcal{G}_i$ of the links.
- `Slist`: Screw axes $\mathcal{S}_i$ of the joints in a space frame.

Output:
- `c`: The vector $c(\theta, \dot{\theta})$ of Coriolis and centripetal terms for a given $\theta$ and $\dot{\theta}$.

This function calls `InverseDynamics` with $\mathfrak{g} = 0$, $\mathcal{F}_{\text{tip}} = 0$, and $\ddot{\theta} = 0$.

```
grav = GravityForces(thetalist,g,Mlist,Glist,Slist)
```

Input:
- `thetalist`: $n$-vector of joint variables $\theta$.
- `g`: Gravity vector $\mathfrak{g}$.
- `Mlist`: List of link frames $\{i\}$ relative to $\{i-1\}$ at the home position.
- `Glist`: Spatial inertia matrices $\mathcal{G}_i$ of the links.
- `Slist`: Screw axes $\mathcal{S}_i$ of the joints in a space frame.

Output:
- `grav`: The joint forces/torques required to overcome gravity at $\theta$.

This function calls `InverseDynamics` with $\dot{\theta} = \ddot{\theta} = 0$ and $\mathcal{F}_{\text{tip}} = 0$.

```
JTFtip = EndEffectorForces(thetalist,Ftip,Mlist,Glist,Slist)
```

Input:
- `thetalist`: $n$-vector of joint variables $\theta$.
- `Ftip`: Wrench $\mathcal{F}_{\text{tip}}$ applied by the end-effector expressed in frame $\{n+1\}$.
- `Mlist`: List of link frames $\{i\}$ relative to $\{i-1\}$ at the home position.
- `Glist`: Spatial inertia matrices $\mathcal{G}_i$ of the links.
- `Slist`: Screw axes $\mathcal{S}_i$ of the joints in a space frame.

Output:
- `JTFtip`: The joint forces and torques $J^T(\theta)\mathcal{F}_{\text{tip}}$ required to create the end-effector force $\mathcal{F}_{\text{tip}}$.

This function calls `InverseDynamics` with $\mathfrak{g} = 0$ and $\dot{\theta} = \ddot{\theta} = 0$.

```
ddthetalist = ForwardDynamics(thetalist,dthetalist,taulist,
              g,Ftip,Mlist,Glist,Slist)
```

Input:
- `thetalist`: $n$-vector of joint variables $\theta$.
- `dthetalist`: $n$-vector of joint velocities $\dot{\theta}$.
- `taulist`: The $n$-vector $\tau$ of required joint forces/torques.
- `g`: Gravity vector $\mathfrak{g}$.
- `Ftip`: Wrench $\mathcal{F}_{\text{tip}}$ applied by the end-effector expressed in frame $\{n+1\}$.
- `Mlist`: List of link frames $\{i\}$ relative to $\{i-1\}$ at the home position.
- `Glist`: Spatial inertia matrices $\mathcal{G}_i$ of the links.
- `Slist`: Screw axes $\mathcal{S}_i$ of the joints in a space frame.

Output:
- `ddthetalist`: The resulting joint accelerations $\ddot{\theta}$.

This function computes $\ddot{\theta}$ by solving

$$M(\theta)\ddot{\theta} = \tau - c(\theta, \dot{\theta}) - g(\theta) - J^T(\theta)\mathcal{F}_{\text{tip}}.$$

```
[thetalistNext,dthetalistNext] = EulerStep(thetalist,dthetalist,ddthetalist,dt)
```

Input:
- `thetalist`: $n$-vector of joint variables $\theta$.
- `dthetalist`: $n$-vector of joint velocities $\dot{\theta}$.
- `ddthetalist`: $n$-vector of joint accelerations $\ddot{\theta}$.
- `dt`: The timestep $\Delta t$.

Output:
- `thetalistNext`: Vector of joint variables $\theta$ after $\Delta t$ from first-order Euler integration.
- `dthetalistNext`: Vector of joint velocities $\dot{\theta}$ after $\Delta t$ from first-order Euler integration.

```
taumat = InverseDynamicsTrajectory(thetamat,dthetamat,ddthetamat,
            g,Ftipmat,Mlist,Glist,Slist)
```

Input:

`thetamat`: An $N \times n$ matrix of robot joint variables. Each row is an $n$-vector of joint variables, and the $N$ rows correspond to $N$ time instants. (The time instants can be thought of as starting at time 0 and ending at time $T_f$, in increments $\Delta t = T_f/(N-1)$.)

`dthetamat`: An $N \times n$ matrix of robot joint velocities.

`ddthetamat`: An $N \times n$ matrix of robot joint accelerations.

`g`: Gravity vector $\mathfrak{g}$.

`Ftipmat`: An $N \times 6$ matrix, where each row is a vector of the form $\mathcal{F}_{\text{tip}}(k\Delta t)$. (If there are no tip forces the user should input a zero and a zero matrix will be used).

`Mlist`: List of link frames $\{i\}$ relative to $\{i-1\}$ at the home position.

`Glist`: Spatial inertia matrices $\mathcal{G}_i$ of the links.

`Slist`: Screw axes $\mathcal{S}_i$ of the joints in a space frame.

Output:

`taumat`: The $N \times n$ matrix of joint forces/torques for the specified trajectory, where each of the $N$ rows is the vector of joint forces/torques at each time step.

This function uses `InverseDynamics` to calculate the joint forces/torques required to move the serial chain along the given trajectory.

```
[thetamat,dthetamat] = ForwardDynamicsTrajectory(thetalist,dthetalist,taumat,
            g,Ftipmat,Mlist,Glist,Slist,dt,intRes)
```

Input:

`thetalist`: $n$-vector of initial joint variables.

`dthetalist`: $n$-vector of initial joint velocities.

`taumat`: An $N \times n$ matrix of joint forces/torques, where each row is the joint effort at any instant. The time corresponding to row $k$ is $k\Delta t$, $k \in \{0, \ldots, N\}$, where $\Delta t$ is defined below.

`g`: Gravity vector $\mathfrak{g}$.

`Ftipmat`: An $N \times 6$ matrix, where each row is a vector of the form $\mathcal{F}_{\text{tip}}(k\Delta t)$. (If there are no tip forces the user should input a zero and a zero matrix will be used).

`Mlist`: List of link frames $\{i\}$ relative to $\{i-1\}$ at the home position.

`Glist`: Spatial inertia matrices $\mathcal{G}_i$ of the links.

`Slist`: Screw axes $\mathcal{S}_i$ of the joints in a space frame.

`dt`: The timestep $\Delta t$ between consecutive joint forces/torques.

`intRes`: This input must be an integer greater than or equal to 1. `intRes` is the number of Euler integration steps during each timestep $\Delta t$. Larger values result in slower simulations but less accumulation of integration error.

Output:

`thetamat`: The $N \times n$ matrix of robot joint variables resulting from the specified joint forces/torques.

`dthetamat`: The $N \times n$ matrix of robot joint velocities resulting from the specified joint forces/torques.

This function simulates the motion of a serial chain given an open-loop history of joint forces/torques. It calls a numerical integration procedure that uses `ForwardDynamics`.

## Chapter 9: Trajectory Generation

```
s = CubicTimeScaling(Tf,t)
```

Input:

`Tf`: Total time of the motion $T_f$ in seconds from rest to rest.

`t`: The current time $t$ satisfying $0 \leq t \leq T_f$.

Output:

`s`: The path parameter $s(t)$ corresponding to a third-order polynomial motion that begins (at $s = 0$) and ends (at $s = 1$) at zero velocity.

```
s = QuinticTimeScaling(Tf,t)
```

Input:

`Tf`: Total time of the motion $T_f$ in seconds from rest to rest.

`t`: The current time $t$ satisfying $0 \leq t \leq T_f$.

Output:

`s`: The path parameter $s(t)$ corresponding to a fifth-order polynomial motion that begins (at $s = 0$) and ends (at $s = 1$) at zero velocity and zero acceleration.

```
traj = JointTrajectory(thetastart,thetaend,Tf,N,method)
```

Input:

`thetastart`: The initial joint variables $\theta_{\text{start}} \in \mathbb{R}^n$.

`thetaend`: The final joint variables $\theta_{\text{end}}$.

`Tf`: Total time of the motion $T_f$ in seconds from rest to rest.

`N`: The number of points $N \geq 2$ in the discrete representation of the trajectory.

`method`: The time-scaling method, where 3 indicates cubic (third-order polynomial) time scaling and 5 indicates quintic (fifth-order polynomial) time scaling.

Output:

`traj`: A trajectory as an $N \times n$ matrix, where each row is an $n$-vector of joint variables at an instant in time. The first row is $\theta_{\text{start}}$ and the $N$th row is $\theta_{\text{end}}$. The elapsed time between each row is $T_f/(N-1)$.

The returned trajectory is a straight-line motion in joint space.

```
traj = ScrewTrajectory(Xstart,Xend,Tf,N,method)
```

Input:

`Xstart`: The initial end-effector configuration $X_{\text{start}} \in SE(3)$.

`Xend`: The final end-effector configuration $X_{\text{end}}$.

`Tf`: Total time of the motion $T_f$ in seconds from rest to rest.

`N`: The number of points $N \geq 2$ in the discrete representation of the trajectory.

`method`: The time-scaling method, where 3 indicates cubic (third-order polynomial) time scaling and 5 indicates quintic (fifth-order polynomial) time scaling.

Output:

`traj`: The discretized trajectory as a list of $N$ matrices in $SE(3)$ separated in time by $T_f/(N-1)$. The first in the list is $X_{\text{start}}$ and the $N$th is $X_{\text{end}}$.

This function calculates a trajectory corresponding to a screw motion about a constant screw axis.

```
traj = CartesianTrajectory(Xstart,Xend,Tf,N,method)
```

Input:

`Xstart`: The initial end-effector configuration $X_{\text{start}} \in SE(3)$.

`Xend`: The final end-effector configuration $X_{\text{end}}$.

`Tf`: Total time of the motion $T_f$ in seconds from rest to rest.

`N`: The number of points $N \geq 2$ in the discrete representation of the trajectory.

`method`: The time-scaling method, where 3 indicates cubic (third-order polynomial) time scaling and 5 indicates quintic (fifth-order polynomial) time scaling.

Output:

`traj`: The discretized trajectory as a list of $N$ matrices in $SE(3)$ separated in time by $T_f/(N-1)$. The first in the list is $X_{\text{start}}$ and the $N$th is $X_{\text{end}}$.

Similar to `ScrewTrajectory`, except the origin of the end-effector frame follows a straight line, decoupled from the rotational motion.

## Chapter 11: Robot Control

The two functions in this chapter focus on the use of the feedback linearizing controller

$$\tau = \widehat{M}(\theta)\left(\ddot{\theta}_d + K_p\theta_e + K_i\int \theta_e(t)dt + K_d\dot{\theta}_e\right) + \widehat{h}(\theta,\dot{\theta})$$

to control the motion of a serial chain in free space. The term $\widehat{h}(\theta,\dot{\theta})$ comprises the model of Coriolis forces and gravitational forces.

```
taulist = ComputedTorque(thetalist,dthetalist,eint,g,
             Mlist,Glist,Slist,thetalistd,dthetalistd,ddthetalistd,Kp,Ki,Kd)
```

Input:

`theta0`: $n$-vector of initial joint variables.
`dtheta0`: $n$-vector of initial joint velocities.
`eint`: An $n$-vector of the time-integral of joint errors.
`g`: Gravity vector $\mathfrak{g}$.
`Mlist`: List of link frames $\{i\}$ relative to $\{i-1\}$ at the home position.
`Glist`: Spatial inertia matrices $\mathcal{G}_i$ of the links.
`Slist`: Screw axes $\mathcal{S}_i$ of the joints in a space frame.
`thetalistd`: $n$-vector of reference joint variables $\theta_d$.
`dthetalistd`: $n$-vector of reference joint velocities $\dot{\theta}_d$.
`ddthetalistd`: $n$-vector of reference joint accelerations $\ddot{\theta}_d$.
`Kp`: The feedback proportional gain (identical for each joint).
`Ki`: The feedback integral gain (identical for each joint).
`Kd`: The feedback derivative gain (identical for each joint).

Output:
`taulist`: The vector of joint forces/torques computed by the computed torque controller at the current instant.

```
[taumat,thetamat] = SimulateControl(thetalist,dthetalist,g,Ftipmat,Mlist,Glist,
              Slist,thetamatd,dthetamatd,ddthetamatd,gtilde,Mtildelist,
              Gtildelist,Kp,Ki,Kd,dt,intRes)
```

Input:
`theta0`: $n$-vector of initial joint variables.
`dtheta0`: $n$-vector of initial joint velocities.
`g`: Actual gravity vector $\mathfrak{g}$.
`Ftipmat`: An $N \times 6$ matrix, where each row is a vector of the form $\mathcal{F}_{\text{tip}}(k\Delta t)$. (If there are no tip forces the user should input a zero and a zero matrix will be used).
`Mlist`: Actual list of link frames $\{i\}$ relative to $\{i-1\}$ at the home position.
`Glist`: Actual spatial inertia matrices $\mathcal{G}_i$ of the links.
`Slist`: Screw axes $\mathcal{S}_i$ of the joints in a space frame.
`thetamatd`: An $N \times n$ matrix of desired joint variables $\theta_d$ from the reference trajectory. The first row is the initial desired joint configuration, and the $N$th row is the final desired joint configuration. The time between each row is `dt`, below.
`dthetamatd`: An $N \times n$ matrix of desired joint velocities $\dot{\theta}_d$.
`ddthetamatd`: An $N \times n$ matrix of desired joint accelerations $\ddot{\theta}_d$.
`gtilde`: The (possibly incorrect) model of the gravity vector.
`Mtildelist`: The (possibly incorrect) model of the link frame locations.
`Gtildelist`: The (possibly incorrect) model of the link spatial inertias.
`Kp`: The feedback proportional gain (identical for each joint).
`Ki`: The feedback integral gain (identical for each joint).
`Kd`: The feedback derivative gain (identical for each joint).
`dt`: The timestep $\Delta t$ between points on the reference trajectory.
`intRes`: This input must be an integer greater than or equal to 1. `intRes` is the number of Euler integration steps during each timestep $\Delta t$. Larger values result in slower simulations but less accumulation of integration error.

Output:

`taumat`: An $N \times n$ matrix of the controller's commanded joint forces/torques, where each row of $n$ forces/torques corresponds to a single time instant.

`thetamat`: An $N \times n$ matrix of actual joint variables, to be compared to `thetamatd`.

Plot: Plot of actual and desired joint variables.

This function uses `ComputedTorque`, `ForwardDynamics`, and numerical integration to simulate the performance of a computed torque control law operating on a serial chain. Disturbances come in the form of initial position and velocity errors; incorrect models of gravity, the locations of the link center of mass frames, and the link spatial inertias; and errors introduced by the numerical integration.