ME 449 Robotic Manipulation
Fall 2015
Problem Set 5
Due Friday December 11 by 9 AM (turn in on Canvas).

1. You will add the following functions to your robotics library. Some are implementations of the dynamic equations

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

- `InverseDynamics`: This function computes the inverse dynamics of an n-joint serial chain. The output is a vector of joint forces/torques τ. The input is:
  - A vector θ of joint variables.
  - A vector $\dot{\theta}$ of joint velocities.
  - A vector $\ddot{\theta}$ of joint accelerations.
  - A three-vector g indicating the direction and magnitude of gravitational acceleration. (The acceleration of the base link of the robot is considered to be the negative of this.)
  - A spatial force six-vector that indicates the wrench $\mathcal{F}_{\text{tip}}$ applied by the robot's end-effector expressed in its frame $\{n+1\}$, which is fixed relative to frame $\{n\}$, where frame $\{0\}$ is the base frame and frames $\{1\}$ to $\{n\}$ are at the centers of mass of the n robot links.
  - A description of the robot, including:
    - $\mathcal{M}_{i-1,i} \in SE(3)$ for $i = 1, \ldots, n+1$, indicating the configuration of the center-of-mass frame of link i relative to the center-of-mass frame of link $i-1$ when joint i is at its zero (home) position. (Note there is no center of mass for the $(n+1)$th frame; this frame is only used to express end-effector forces in a frame other than the last link's center of mass.)
    - The 6×6 spatial inertia $\mathcal{G}_i$ for each link $i = 1, \ldots, n$, expressed in the links' center-of-mass frames.
    - The n screw axes $\mathcal{S}_i$ expressed in the fixed base frame.

  This function will make use of the Newton-Euler inverse dynamics algorithm. You might find it convenient to define a helper function or two, like the Lie bracket or the calculation of the spatial force applied to a rigid body given its spatial inertia matrix, spatial velocity, and rate of change of spatial velocity.

- `InertiaMatrix`: This function computes the numerical inertia matrix $M(\theta)$ of an n-joint serial chain at a given configuration θ. There is more than one way to do this, but we will use a simple method that makes use of the function `InverseDynamics`. You will call `InverseDynamics` n times, each time getting one column of the inertia matrix, and assemble those n columns into $M(\theta)$. To get the ith column of M, call `InverseDynamics` with $\dot{\theta} = 0$, $g = 0$, $\mathcal{F}_{\text{tip}} = 0$, and $\ddot{\theta}$ equal to zero except for the ith component, which is set equal to 1. Now when `Inverse Dynamics` calculates $\tau = M(\theta)\ddot{\theta} + c(\theta, \dot{\theta}) + g(\theta)$, clearly c will be equal to zero (no Coriolis or centripetal terms since $\dot{\theta} = 0$), $g(\theta)$ will be equal to zero, and our choice of $\ddot{\theta}$ will pick out the ith column of M. So the input to the function `InertiaMatrix` is
  - A vector θ of joint variables.
  - A description of the robot, including:
    - $\mathcal{M}_{i-1,i} \in SE(3)$ for $i = 1, \ldots, n+1$.
    - The 6×6 spatial inertia $\mathcal{G}_i$ for each link $i = 1, \ldots, n$, expressed in the links' center-of-mass frames.
    - The n screw axes $\mathcal{S}_i$ expressed in the fixed base frame.

- `CoriolisForces`: This function computes the vector $c(\theta, \dot{\theta})$ of Coriolis and centripetal terms for a given θ and $\dot{\theta}$. It also uses `InverseDynamics`, calling it once with $g = 0$, $\mathcal{F}_{\text{tip}} = 0$, and $\ddot{\theta} = 0$. The inputs to `CoriolisForces` are θ, $\dot{\theta}$, and a description of the robot.

- `GravityForces`: This function computes the vector $g(\theta)$ using `InverseDynamics`, calling it once with $\dot{\theta} = \ddot{\theta} = 0$ and $\mathcal{F}_{\text{tip}} = 0$. The inputs to `GravityForces` are θ, g, and a description of the robot.

- `EndEffectorForces`: This function computes $J^T(\theta)\mathcal{F}_{\text{tip}}$. It is your choice how to compute it. As input it needs (at a minimum) $\mathcal{F}_{\text{tip}}$, θ, and a description of your robot's kinematics from which the Jacobian can be derived. Alternatively, you can use `InverseDynamics` again.

- `ForwardDynamics`: This function computes $\ddot{\theta}$ given θ, $\dot{\theta}$, τ, g, $\mathcal{F}_{\text{tip}}$, and a description of the robot. It does this by solving
$$M(\theta)\ddot{\theta} = \tau - c(\theta, \dot{\theta}) - g(\theta) - J^T(\theta)\mathcal{F}_{\text{tip}}$$
for $\ddot{\theta}$. You can use your programming language's method for solving equations like $Ax = b$ for known A and b.

- `EulerStep`: This function takes the current state $(\theta(t), \dot{\theta}(t))$, the current acceleration $\ddot{\theta}(t)$, and the timestep Δt, and returns the state after Δt according to a simple first-order Euler integration step:
$$\theta(t + \Delta t) = \theta(t) + \Delta t\dot{\theta}(t)$$
$$\dot{\theta}(t + \Delta t) = \dot{\theta}(t) + \Delta t\ddot{\theta}(t).$$

- `InverseDynamicsTrajectory`: This function takes a robot trajectory specified as a set of $N+1$ points $(\theta(k\Delta t), \dot{\theta}(k\Delta t), \ddot{\theta}(k\Delta t)$, where $k \in \{0, 1, \ldots, N\}$ and Δt is the timestep between points. The total time of the trajectory is $T = N\Delta t$. The function also takes as input a set of $N + 1$ values of the form $\mathcal{F}_{\text{tip}}(k\Delta t)$, allowing the specification of a time-varying set of endpoint forces. (By default these can be zero.) Other inputs are the gravity vector g and the description of the robot. The output of the function is a matrix with n columns and $N + 1$ rows, where the kth row of the matrix is $\tau(k\Delta t)$ as computed by `InverseDynamics`.

- `ForwardDynamicsTrajectory`: This function takes an initial robot state $(\theta(0), \dot{\theta}(0))$ and a joint force/torque history $\tau(k\Delta t)$, where $k \in \{0, 1, \ldots, N\}$ and Δt is the timestep between points. It uses `EulerStep` to compute and return the robot state as a function of time, $(\theta(k\Delta t), \dot{\theta}(k\Delta t))$. It takes whatever else is needed as input.

2. For the UR5 robot, described by its URDF file (and note the changed base frame), write the description of the robot given by the $\mathcal{M}_{i-1,i} \in SE(3)$ for $i = 1, \ldots, n + 1$; the $\mathcal{G}_i$; and the $\mathcal{S}_i$.

3. For the UR5 robot in gravity, choose a timestep $\Delta t = 0.001$ s and create a 1001-point discretized quintic trajectory from rest at $\theta(0) = 0$ to $\theta(T = 1 \text{ s})$ when all joint angles are $\pi/2$. Plot $\theta(t)$, $\dot{\theta}(t)$, and $\ddot{\theta}(t)$ for one of the joints (it will be the same for all joints), and use `InverseDynamicsTrajectory` to generate the necessary robot torques as a function of time. Plot all six torques on the same vertical scale. Assume gravity is 9.81 m/s^2, downward in the base frame's $\hat{z}$-axis direction.

4. For the UR5 robot in gravity as in the previous problem, and given the initial state of the previous problem, assume that the joint torques at each joint are constant at 2 Nm. Simulate the motion of the robot for one second (at 1 ms resolution) using `ForwardDynamicsTrajectory`. Plot the six joint positions as a function of time. Make a movie of the motion, post it on Youtube, and provide the link.

5. Extra credit (worth up to an extra 25% of this assignment's grade): Robot control. Write two more functions for your library:

- `FF_FB_Control`: Takes the robot model; desired position, velocity, and acceleration at the current time; actual position and velocity at the current time; and feedback gains Kp, Ki, and Kd; and calculates the commanded joint forces/torques τ.

- `SimulateControl`: Given

- the actual model of the robot
- the robot's estimate of the model
- the gain matrices
- the initial state of the robot
- a desired trajectory of the robot (including the timestep)

simulate the robot controller `FF_FB_Control` attempting to follow the trajectory and output the joint torques and the actual joint angles as a function of time. This function can also plot the actual versus desired joint angles as a function of time.

Use the UR5 and the quintic trajectory above as your test motion. The controller should use the URDF model, and the simulator should use the same, except with a 1 kg point mass added at the end-effector frame. Indicate the new effective inertia for the last link, simulate your controller, provide plots of the joint torques and joint angles as a function of time, and put a movie of the motion on Youtube.