Advanced Robotics 2014
University of Birmingham
Due: April 6th 2014
Total Mark: 40%

General notes:
You don't have to use Matlab but I strongly recommend it (alternatively you may use any programming language of your choice). The Robotics Toolbox already provides functionality for many of the problems you will solve here, but the goal of this assignment is not to copy what the Toolbox is already doing, but to help you understand how these algorithms work by implementing them yourself. I suggest you use the Robotics Toolbox for plotting/visualization and for testing the final results of your algorithms. Submit all your source code along with a report. The code should be made easy enough for me to execute and see what the robot is doing. Note I am not concerned with computational efficiency or speed. I prefer to see accurate results, even if your program is slow. The report should explain more details along with providing figures and plots. Submit your assignment on Canvas:

https://birmingham.instructure.com/

Note, if you submit some parts of the assignment early, I will do my best to provide feedback (without giving the answer away). You may then update your submission before the final deadline. I have put suggested deadlines for each problem as a guideline. Don't wait until the last week!

1. (10%) [Suggested deadline: 14/3] Write a Matlab function to compute the direct kinematics transformation and the Jacobian matrix, given a set of Denavit-Hartenberg parameters and the robot configuration vector. The function should take as arguments DH: an N by 5 matrix of the 4 DH parameters per joint plus a column to indicate if the joint is revolute or prismatic (0 for revolute, 1 for prismatic), and q: the N by 1 vector of the robot configuration. The function should return T the homogenous transformation matrix from base to end-effector frame and J the Jacobian with respect to the base frame:

```
function [T,J] = fk(DH,q)
```

2. (10%) [Suggested deadline: 21/3] Write a Matlab function to solve for the iterative inverse kinematics solution to position the end-effector of a planar robot with revolute links. The function should take as arguments:

DH (same as defined above) (but you may limit your algorithm to only planar robots and revolute links),
q: the current configuration vector,
p_des: the 2 by 1 desired end-effector position,
q_des: the N by 1 secondary desired joint position (the solution of which should not affect the solution of p_des),

Fq: a flag of 0 or 1 indicating whether or not to ignore q_des (0 to ignore, 1 to not ignore). If Fq is 1 then the algorithm should try to minimize q_des - q as best as possible, while maintaining p_des - p = 0.

Your function should run iteratively, so you will need to choose an appropriate terminating condition. Please clearly indicate (by comments in your code and text in your report) which method you are choosing.

The function returns q_new: the new joint configuration vector.

function [q_new] = ik(DH,q,p_des,q_des,Mq)

Show that your function works by demonstrating some interesting cases. For example, show that a 20+ link robot can curl into a spiral centered at its end-effector, or show that the robot can trace out some letters with its body. Show at least one case where the robot does not have enough DOF to achieve q_des (but show that the algorithm still attempts to realize q_des as best as possible).

3. (10%) [Suggested deadline:28/3] Consider the 3-DOF planar robot shown on the last page. There are three point masses connected by massless rods and revolute joints. Gravity acts downward as shown on the figure. Derive the equation of motion for this system using the Lagrangian formulation. Show your work in each of the intermediate steps below:

a. Derive the Kinetic Energy of the system
b. Derive the Potential Energy of the system.
c. Derive the Lagrangian, L, for the system
d. Derive each term for the required partial derivatives individually, i.e:

$$\frac{\partial L}{\partial \theta_i}, \frac{\partial L}{\partial \dot{\theta}_i}, \frac{d}{dt}\frac{\partial L}{\partial \theta_i}$$

e. put the above terms together to define the complete equations of motion. Clearly indicate what are the inertia, Coriolis, and centrifugal matrices as well as the gravity vector.

4. (10%) [Suggested deadline: 6/4] Use the Iterative Newton-Euler algorithm to create a simulator of a planar robot with a series of N revolute links. To simplify matters you may assume motion in the plane only, the base is fixed (does not move), and each link has the same inertial properties (same length, mass, moment of inertia, and friction coefficients). Also assume the center of mass of each link is located at the center of the link. Follow the the steps below.

5. Write a function:

```
function [tau] = NewtonEuler(N,L,m,I,fv,fc,g,q,qd,qdd)
```

With the following inputs:
• N: number of links (scalar)
• L: length of each link (scalar)
• m: mass of each link (scalar)
• I: moment of inertia of each link (scalar). You may choose your convention (with respect to the center of mass of the link, or with respect to the axis of rotation), but clearly indicate your choice.
• fv: coefficient of viscous friction for each link (scalar)
• fc: coefficient of coulomb friction for each link (scalar)
• g: the gravity vector w.r.t the base (2x1 vector). [0; -9.81] is typically a good choice.
• q: the configuration of the robot (Nx1 vector)
• qd: velocity of the robot (Nx1 vector)
• qdd: acceleration of the robot (Nx1 vector).

And the output tau is a Nx1 vector of torques at the joints. Note that actuator torque is not included in the above function. Use your NewtonEuler function to first generate the inertia matrix, **M**. Then use it again to generate the following term (without the inertia matrix):

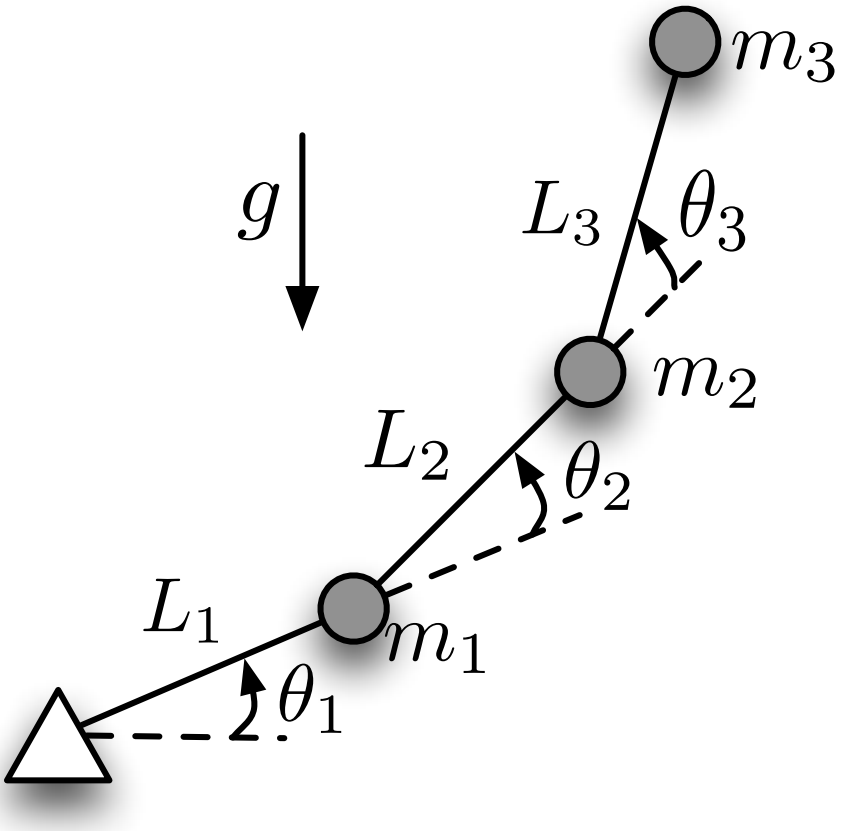
$$\tau' = \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) + f_v \dot{\mathbf{q}} + f_c \mathrm{sgn}(\dot{\mathbf{q}})$$

Then compute the acceleration of the system for the next simulation step, using the equation:

$$\ddot{\mathbf{q}} = \mathbf{M}^{-1}(\mathbf{q})(\tau_{\mathrm{act}} - \tau')$$

where tau_act is the actuator torque. Finally, use Euler integration to compute the position and velocity of the next simulation step. Submit your code as well as some example outputs. Use the robotics toolbox to visualize a few steps of the simulation on at least a 1 link robot (i.e. a pendulum) and a 3 link robot. (You may create your own simple visualization if you choose not to use the robotics toolbox.) You may include some actuation torque (e.g. a sine wave), or just show the effects of gravity given some interesting initial condition. Note for this problem I am not concerned about computational efficiency, speed of computation, etc., just the accuracy of the computation.

Also, use the NewtonEuler function to verify the analytical derivation of the robot in question 2. In at least 2 configurations, show that the M, C, and G computed by NewtonEuler matches with the analytic result you derived in q2.


m_3
g
L_3
θ_3
m_2
L_2
θ_2
L_1
m_1
θ_1