EEE4119F – practical 4

Student names: Student numbers:

Notes/tips before you start

Revise!

You can easily waste a lot of time getting the wrong answers on this practical. If you get stuck, DON'T enter stuff until things work/immediately ask a friend. First revise the Lagrangian and 3D sections! This is an opportunity to make sure you actually understand everything!

Matlab or Octave

While unsure whether we'd use Matlab or Octave for this practical, I modified it so that you can complete it with equal effort in either language. That mainly meant removing Simulink, so you have to Simulate the system using code (as opposed to doing it graphically). However, if you can, I suggest you simulate both systems using Simulink as well so you get a better idea of what it does for you.

Defining symbols

There are two approaches you could use to work with symbols and their derivatives:

1. Use MATLABs built in time derivative stuff, which assumes your variables are functions of time. Eg:

```
>>> syms x(t) y(t)
>>> q = [x; y]; dq = [dx; dy]; ddq = [ddx; ddy];
>>> eqn = x^2 + y^2;
>>> diff(eqn)
2*x(t)*diff(x(t), t) + 2*y(t)*diff(y(t), t)
```

2. Define the derivatives yourself, and be careful to make the chain rule to apply. Eg:

```
>>> syms x dx ddx y dy ddy
>>> q = [x; y]; dq = [dx; dy]; ddq = [ddx; ddy];
>>> eqn = x^2 + y^2;
>>> jacobian(eqn, q) * dq  + jacobian(eqn, dq) * ddq
2*dx*x + 2*dy*y
```

You may use whichever approach you prefer. However, despite what the example above would make you believe, I have found that the second approach is far easier to work with (for reasons that escape me at the moment...), once you get used to it. A small reason is that it's nicer when calculating Jacobians and Hessians, which occur far more in mechanics than actually calculating time

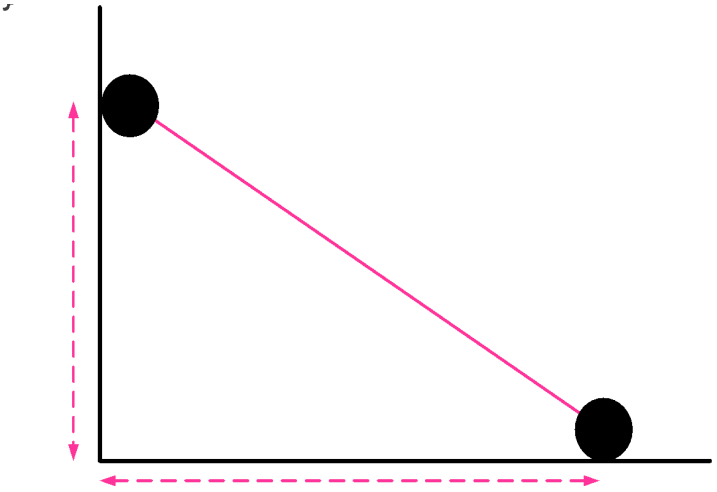
derivatives. I suggest trying question 1 both ways and see what you prefer, and then continuing that in question 2. Remember that you can post on the Q&A for help

Skew symmetric matrix

Question 2 makes use of the skew symmetric matrix property when calculating angular velocities in the zero frame. If you'd like to know how it works, you can read this paper from the lecture 11 folder on Vula to see how it works. Otherwise, this video is also very good. Understanding it is not vital, though!

Question 1

Two point-masses, each with mass (m), are connected by a rigid, massless rod of length l. Mass 1 can move without friction on the vertical y-axis and mass 2 can move without friction on the horizontal x-axis.



a. Use a MATLAB script to generate the EOM for this system.
Hint: you can modify one of the scripts in the slides if you need help

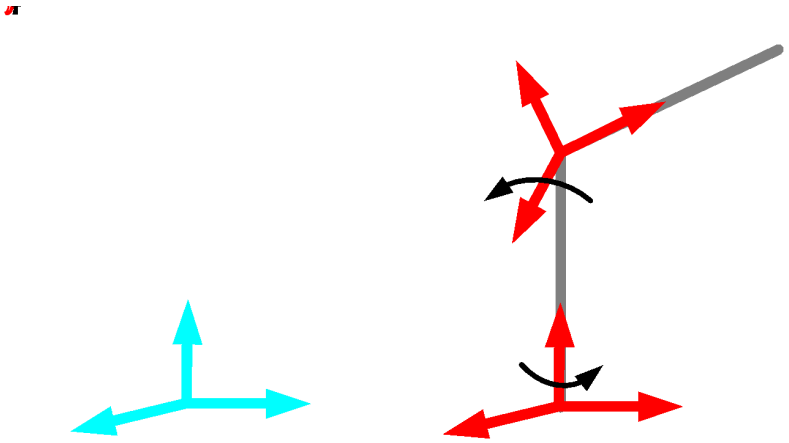
b. Simulate these equations by defining a derivative function `f(t, q)` like you did in practical 3. Your simulation must start at $\alpha = 15°$ (don't forget that MATLAB's trig functions are in radians) and $\dot{\alpha} = 0$, and terminate when $\alpha = 90°$. Assume $m = 1\ kg$, $L = 0.5\ m$. You can make use of the provided `simulate_twoMassSliding.m` script, which handles the simulation termination for you. If you can, simulate it using Simulink as well (with all the constraints as I've described above, like the termination condition)

c. Use the provided animation script `anim_twoMassSliding.m` to observe your solution trajectory. The script requires a variable named `alpha` to be in your MATLAB workspace. If you also used Simulink in (b) above, the variable should be exported from your Simulink model using the `To Workspace` block, with the Save Format settings `Structure with Time`. If needed, you can edit the start of the animation script.

Question 2

Hint: one useful part of Lagrange/Manipulator Equations/etc is that deriving equations of motion becomes more 'algorithmic'. In other words, one could set up standard functions which do everything for you! So, given q, dq, ddq, Ek, Ep and B, you can write a function which finds the rest of the EOM! So, try to stick to the notation and not take shortcuts -- once you get used to it, tackling any new system becomes a lot easier

A two-link robot arm is shown below where each link is actuated by its own torque from a motor:



a. Derive the EOM of the robot by filling in the blanks in the `robotDynamics_twoLink.m` script.

b. Finish off and simulate the Simulink script with input torques = 0.

c. Using a simple PD control for the i-th actuator and angle:

$$\tau_i = k_p^i(\alpha_c - \alpha_m) + k_D^i(\dot{\alpha}_c - \dot{\alpha}_m)$$

$$\alpha_c = desired\ angle, \quad \alpha_m = measured\ angle$$

$$k_p^1 = 10, \quad k_D^1 = 1, \quad k_p^2 = 100, \quad k_D^2 = 1.$$

Assume $m_1 = 10\ kg$, $m_2 = 1\ kg$, $L_1 = 0.5\ m$, $L_2 = 0.25\ m$ and that each link can be modelled as a cylinder with radius $r_c = 0.1m$. Simulate the system and show the tracking response to a step command starting from zero to $\psi_c, \varphi_c = 45°$

d. Copy `anim_twoMassSliding.m` script and modify it so that it produces a visualisation for this system.