Lab 11 Exercises

1 Joystick input

(Download **joystick_test_fprint.m** from UTSOnline)

1.1 Connect a suitable input device to your computer. A generic USB joystick or gamepad is recommended.

1.2 Run the script to check that your input device can be read by MATLAB

1.3 Make of note of which buttons/axes numbers correspond to the physical buttons and axes on the joystick. You will need these to configure your input device for the next lab exercise

Note: Axes values are usually a number ranging from -1 to 1 proportional to the input. Button values are either on/off and are usually either 1 or 0.

2 Human-in-the-loop control: Jogging

(Download **Lab11Question2Skeleton.m** from UTSOnline)

2.1 Inspect the skeleton code to understand how the real time control of the robot is simulate

2.2 Run the script. You will see a simulated Puma robot however it remains stationary as the joint coordinates are not being updated

2.3 In the control loop after the joystick input is read, use the buttons/axes values to construct an end-effector velocity command (in the global frame) for the robot, for example:

$$\boldsymbol{x} = \left[v_x \; v_y \; v_z \; \omega_x \; \omega_y \; \omega_z\right]^T$$

For each Cartesian task-space direction map a joystick axis or a pair of buttons. For example:
`vx = K*axes(1);` or `wx = K*(buttons(2)-buttons(1));` where `K` is a gain used to set max speed.

Note: the axes and buttons used will depend in your input device. You may want to print the vector to the console to check that it is working as intended.

Note: you may want to scale down the robot speed. A gain of 0.3 and 0.8 for linear and angular components respectively worked well for me.

2.4 Use the Jacobian inverse to transform the end-effector velocity into a joint velocity vector:

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^{-1} \cdot \boldsymbol{x}$$

2.5 Apply a step to the robot's joint position based on the joint velocity command. Remember that the size of the step depends on the loop time:

$$\boldsymbol{q}_{k+1} = \boldsymbol{q}_k + \Delta\boldsymbol{q} = \boldsymbol{q}_k + \dot{\boldsymbol{q}}_k \times \Delta t$$

2.6 Run the script and move the simulated robot with your input device.

2.7 Drive the robot to the edge of its workspace. What behaviour is the robot exhibiting? What is the cause of this behaviour?

3 Jogging with Damped-Least-Squares

3.1 Continue from the solution from your previous question

3.2 Rather than using the Jacobian inverse to calculate joint velocity, use DLS to compute an inverse Jacobian with damping. Use parameter $\lambda = 0.1$.

$$\boldsymbol{J}_{DLS}^{-1} = (\boldsymbol{J}^T\boldsymbol{J} + \lambda\boldsymbol{I})^{-1} \cdot \boldsymbol{J}^T$$

$$\dot{\boldsymbol{q}} = \boldsymbol{J}_{DLS}^{-1} \cdot \boldsymbol{x}$$

3.3 Drive the robot to the edge of its workspace and observe its behaviour. How does the motion compare to that in the previous question? Try with damping values $\lambda = 1.0$ and $\lambda = 0.01$ and observe the differences in behaviour.

4 Human-in-the-loop control: Admittance control

(Download **Lab11Question4Skeleton.m** from UTSOnline)

4.1 Consider the scenario of where velocity commands are generated from a sensor mounted on to the robot's end-effector. This sensor measures the human force/torque applied to the end-effector:

$$\boldsymbol{f} = \left[f_x\ f_y\ f_z\ \tau_x\ \tau_y\ \tau_z\right]^T$$

<u>*Note*</u>: because the sensor is mounted on the end-effector, this vector is represented in the end-effector frame.

4.2 In the control loop after the joystick input is read, use the buttons/axes values to construct an end-effector force/torque measurement (in the end-effector frame). Map the buttons/axes so that one axis maps to f_x and another axis maps to τ_y

4.3 Convert the force/torque "measured" at the end-effector by the sensor into a velocity command by using a simple proportional admittance control scheme:

$$\boldsymbol{x} = \boldsymbol{K}_a \cdot \boldsymbol{f} \quad \text{where } \boldsymbol{K}_a = \begin{bmatrix} K_f & 0 & 0 & 0 & 0 & 0 \\ 0 & K_f & 0 & 0 & 0 & 0 \\ 0 & 0 & K_f & 0 & 0 & 0 \\ 0 & 0 & 0 & K_\tau & 0 & 0 \\ 0 & 0 & 0 & 0 & K_\tau & 0 \\ 0 & 0 & 0 & 0 & 0 & K_\tau \end{bmatrix}$$

<u>*Note*</u>: K_f and K_τ are gains used to convert the measured force/torque into linear/angular velocity. Try using gains of $K_f = 0.5$ and $K_\tau = 0.8$

4.4 Using DLS with $\lambda = 0.1$ convert this end-effector velocity into joint velocity. Note that since this velocity command is still represented in the end-effector frame, you can either:
* Use the `robot.jacobe()` function to calculate the Jacobian in the end-effector frame
* Transform the velocity vector into the global frame before computing joint velocities

4.5 Apply a virtual force back and forth in the x direction and notice the robots movement. Next apply a virtual moment about the y axis so that the end-effector's orientation is changed. Now again apply force in the x direction. Notice how the motion from the force applied in the x direction depends on the orientation of the end effector? How does this compare to the previous jogging exercises?