Lab 1
Solutions and Grading Guidelines

Introduction to Pybullet

- 1. (50 Points) Successfully install all packages and run the first notebook.

Sensors and actuators

- 2.1 (5 Points) What is the control rate that we used in this example in Hz?
- 1000 Hz. At each iteration in the for-loop, the simulation time increases 1 ms and the control command is updated once. Hence, the control rate is 1/0.001=1000 Hz.

Sensors and actuators

- 2.2 (10 Points) Describe the behavior of the robot when 0.1 Nm is applied to the first joint and 0 Nm applied to the others. What is the physical explanation for this behavior (in words)?
- The first joint rotates back and forth (5 points) with a decreasing magnitude (1 point), while the other joints remain (almost) still (1 point). This is similar to applying a constant torque to a pendulum (2 points). As time goes to infinity, the robot converges to the configuration in which the external torque is compensated by the gravitational force. (1 point, you will see this more clearly if you increase the simulation time)

Sensors and actuators

- 2.3 (5 Points) What is the maximum constant torque that can be applied on the first joint before it hits the joint limit? (start from the same initial pose for the robot)
- Any number from 0.18 - 0.20 Nm is acceptable

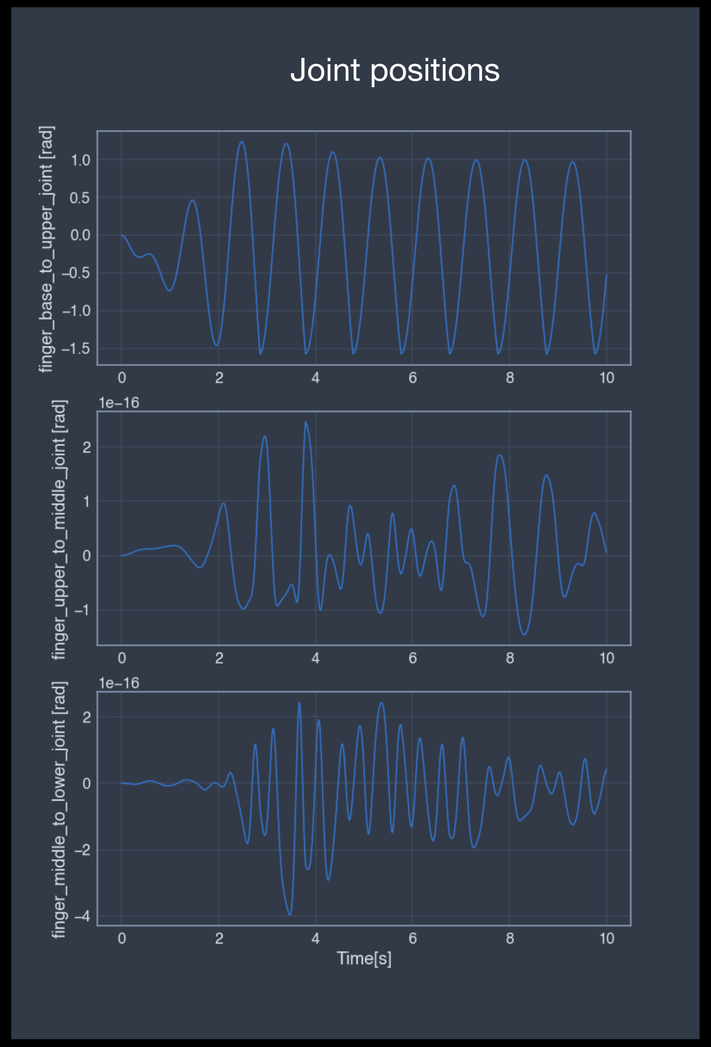
Sensors and actuators

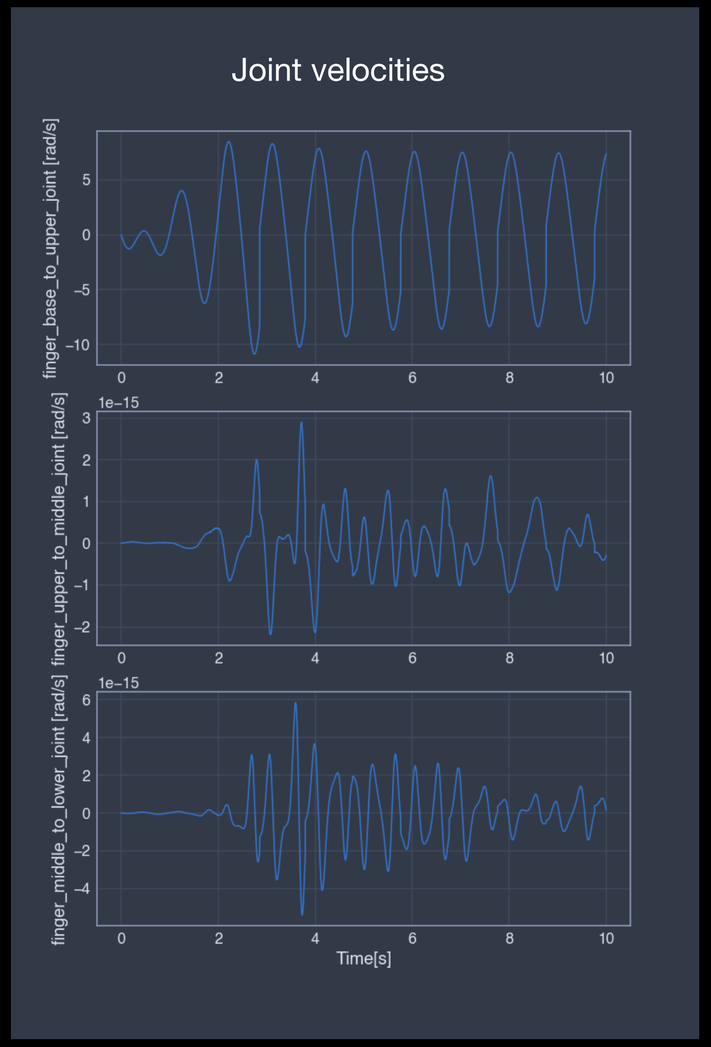
- 2.4 (10 Points) Apply a periodic torque of $0.05\ \sin(2\pi t)$ Nm on the first joint. What do you observe? Join a plot of the position and velocity for each joint. (start from the same initial pose for the robot)
- Inside the for-loop, modify the joint torque

```
t = robot.dt * i
joint_torques = np.array([0.05*np.sin(2*np.pi*t), 0., 0.])
```

- The first joint starts to rotate back and forth with an increasing magnitude (5 points) until it reaches the joint limit. Then, the magnitude decreases and the motion converges to a periodical one with a constant magnitude (1 point). 4 points for the correct plots.

Sensors and actuators





PD Controllers

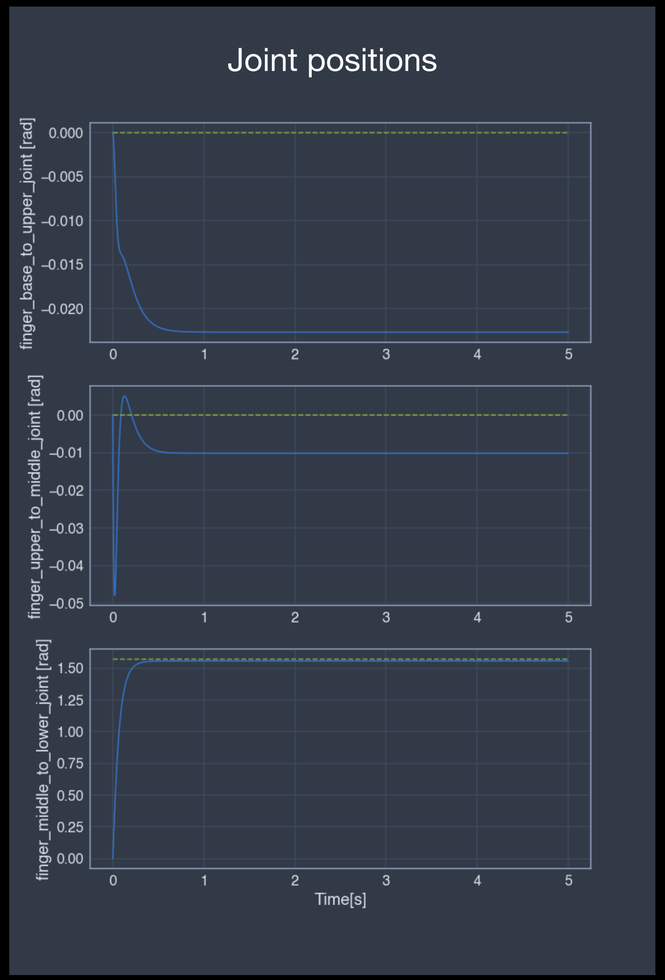
- 3.1 (10 Points) Describe qualitatively what you observe when you increase/ decrease P and D.
- Higher P gain allows the joints to reach the desired position faster (2 points) with smaller steady-state error (1 point). If the P gain is too high, it can cause overshoot or oscillation (2 points). Higher D gain damps/decelerates the joint more (2 points). When the D gain is not too high, increasing it reduces the overshoot and oscillation caused by the proportional term (2 points). However, when the D gain is too high, the system also starts to oscillate (1 point).

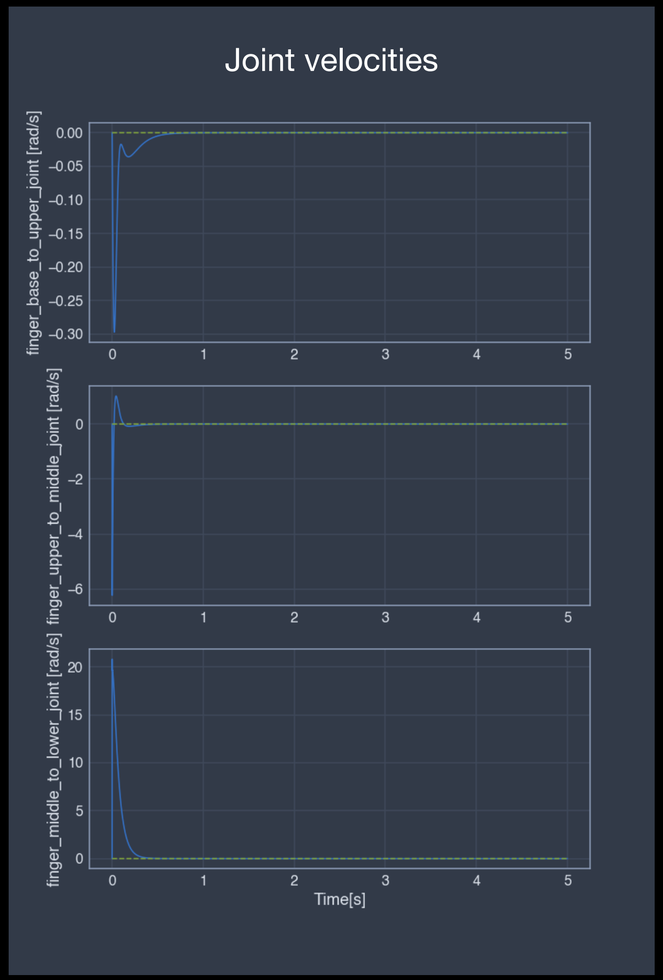
PD Controllers

- 3.2 (5 Points) Tune the P and D gains to have a good tracking of the positions $[0,0,\frac{\pi}{2}]$without any oscillations. The P and D gains need not be the same for different joints. What gains did you find? Plot the position and velocities of each joints as a function of time with these gains. (starting from the original initial robot configuration).
- In order to tune the PD gains, you can first increase the P gains gradually to have smaller steady-state error (error between the actual value and the desired value after the system stabilizes) As you continue increasing the P gains, overshoot/oscillation may occur, then you can increase the D gains gradually to attenuate these effects. Repeat this process until the performance is satisfactory.

PD Controllers

```
# One possible solution
P = np.array([4, 4, 3])
D = np.array([.5, .4, .2])
```





- Transient response: overshoot is acceptable; (damped) oscillation is not.
- Steady-state response: steady-state error should be < 0.05 rad.

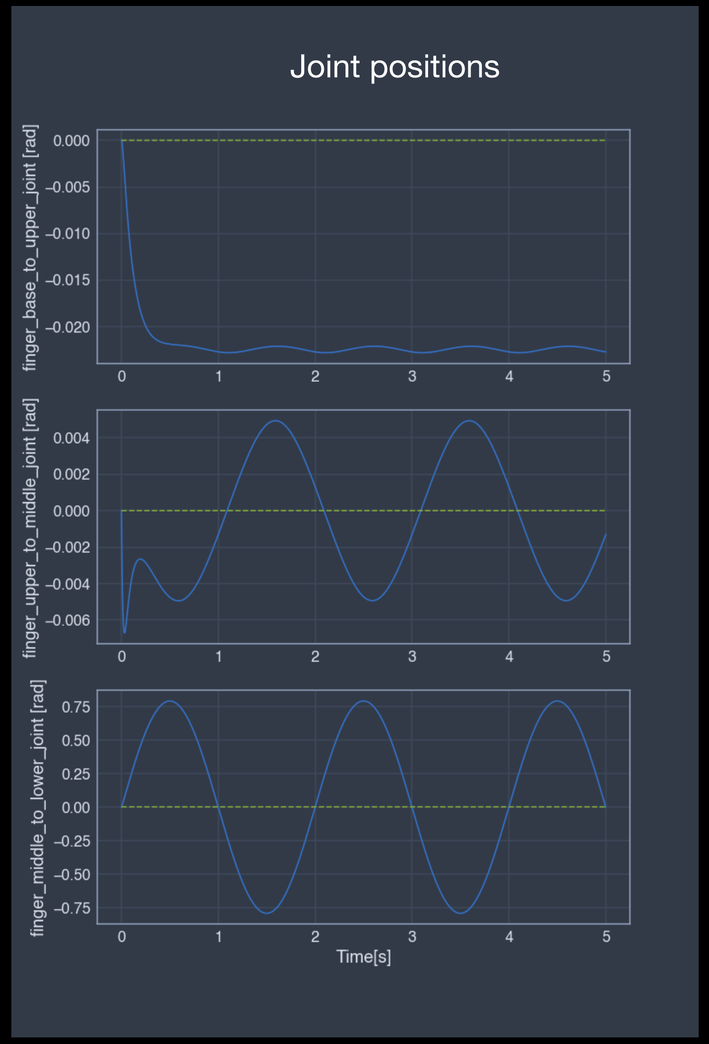
PD Controllers

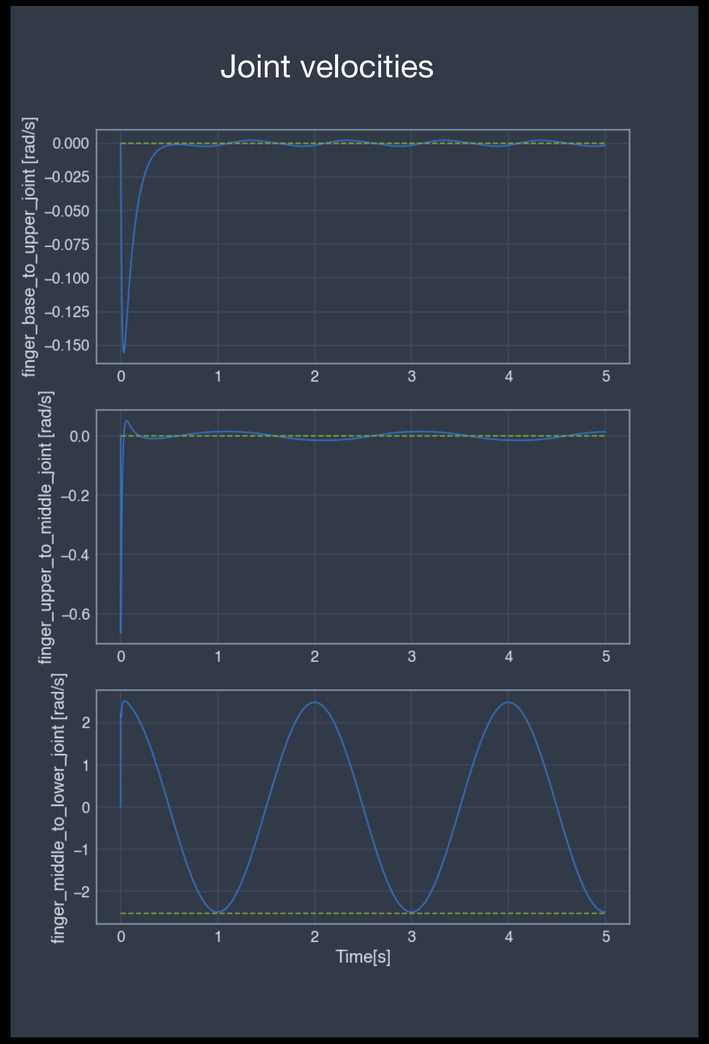
- 3.3 (5 Points) Use the PD controller to do the following task: keep the position of the first two joints fixed and follows the following position trajectory for the last joint$0.8\sin(\pi t)$. Plot the results (positions and velocities as a function of time for all joints). Simulate for at least 10 seconds.
- Inside the for-loop, modify the desired position/velocity as follows

```
t = robot.dt * i
q_des = np.array([0, 0, 0.8*np.sin(np.pi*t)])
dq_des = np.array([0, 0, 0.8*np.pi*np.cos(np.pi*t)])
```

- Note: Points will be subtracted if the tracking error is too large in the plot.

PD Controllers





PD Controllers

- 3.4 (Bonus, 10 Points) Change the joint trajectories to get the robot to draw a circle in the air with its fingertip.
- Inside the for-loop, modify the desired position/velocity as follows

```
'''
For the fingertip trajectory to be a circle:
1. Let two different joints (joint 0/1 or joint 0/2) follow a sinusoidal trajectory with a phase shift = pi/2 (or use cos and sin respectively)
2. Tune the magnitude so the fingertip draws a circle (instead of an ellipse)
'''
t = robot.dt * i
q_des = np.array([0.4*np.cos(np.pi*t), 0.4*np.sin(np.pi*t), 0])
dq_des = np.array([-0.4*np.pi*np.sin(np.pi*t), 0.4*np.pi*np.cos(np.pi*t), 0])

# or use joint 0 and joint 2, here the magnitude of joint 2 needs to be roughly the twice
q_des = np.array([0.4*np.cos(np.pi*t), 0, 0.8*np.sin(np.pi*t)])
dq_des = np.array([-0.4*np.pi*np.sin(np.pi*t), 0, 0.8*np.pi*np.cos(np.pi*t)])
```

- Alternative solution (max. 8 points): fix the first two joints and rotate the rest joint.