Yifei Chen, Nov 2024

Introduction

My submission code include `code.py` and `run.py`. The `code.py` include the function that required for simulate the tasks. I also write two more functions to give the the spring and damping force. The `run.py` run the `puppet` functions for different settings for UR5 and save them as csv file.
Here are the short summary of the content for each file:

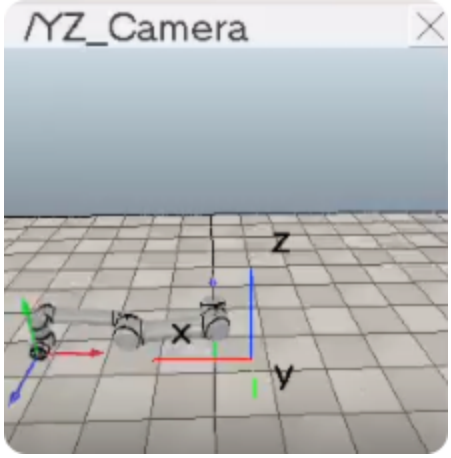
1. code.py
    1. `Puppet` : Main simulate function.
    2. `referencePos` : For the position of the spring force.
    3. `SpringForce` : Compute the springforce according to the stiffness and relative length. return the Ftip.
    4. `DampingForce` : Compute the τ for each joint with damping force.
    5. `filter_nan` : Because sometimes if I choose larger spring stiffness or damping, there will be `NAN` value in csv, which cannot be run by CoppeliaSim. So I have a function here to get rid of them.
2. run.py

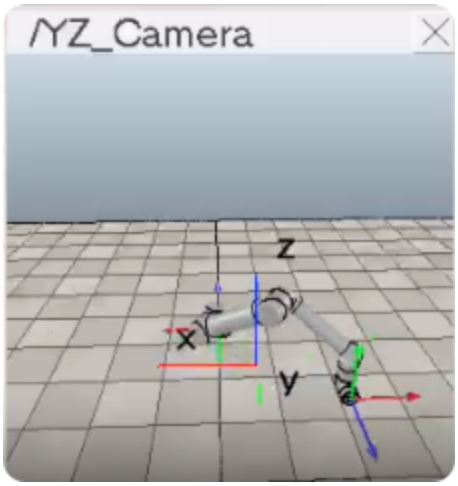
Part 1

In part 1, the spring stiffness and damping are all zero. `t = 5`. The only change is the dt

Part 1 (a)

`dt = 0.005`
The videos show the robot falls in gravity with a slow increase in energy of the system over time. We can see the height is higher than the previous.

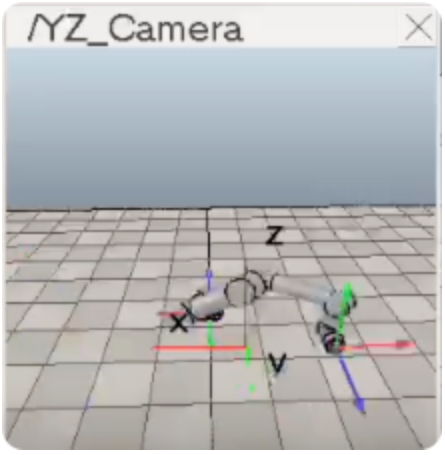




Part 1 (b)

`dt = 0.015`

For large timesteps dt, this growth in energy is substantial.



Part 2

In part 2, the `t = 5` and `dt = 0.01`, the spring stiffness is still zero. The only change is the damping coefficient.

Do you see any strange behavior in the simulation if you choose the damping constant to be a large positive value? Can you explain it?

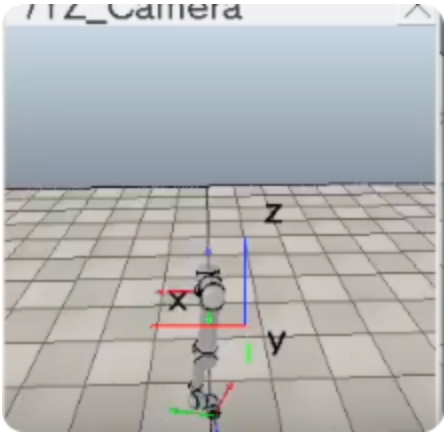
When a large positive damping constant is chosen, the robot's movement can exhibit unstable behavior, with its speed increasing exponentially over time, resulting in extremely high velocities. This phenomenon occurs because the damping torque (intended to counteract motion) actually begins to amplify the robot's joint velocities in an unstable way.

Here's why this happens: In a typical damping model, the damping force is proportional to the velocity (through the damping coefficient B). For large values of B, any small velocity can result in a large damping force. If the damping force is too large relative to the velocity, it can cause the velocity direction to flip from positive to negative (or vice versa) in successive timesteps. Each time this flip occurs, the magnitude of the velocity can grow further instead of being dampened, leading to an exponential increase in speed.

How would this behavior be affected if you chose shorter simulation timesteps?
If we reduce the simulation dt, the tendency for the system to reach infinite velocities would decrease, or it might not occur at all. The damping force $-B \cdot \dot{\theta}_{list}$ (where $\dot{\theta}_{list}$ represents the joint velocities) remains the same regardless of dt, but smaller timesteps mean smaller updates to the velocity in each iteration.

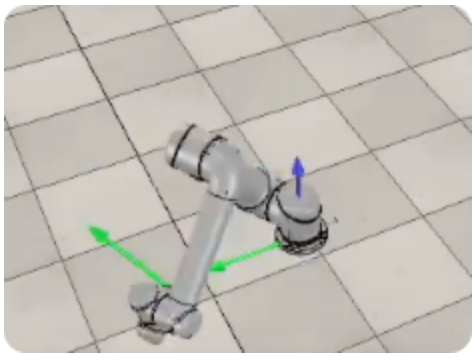
Part 2 (a) Positive damping

`damping = 3`



Part 2 (b) Negative damping

`damping = -1`

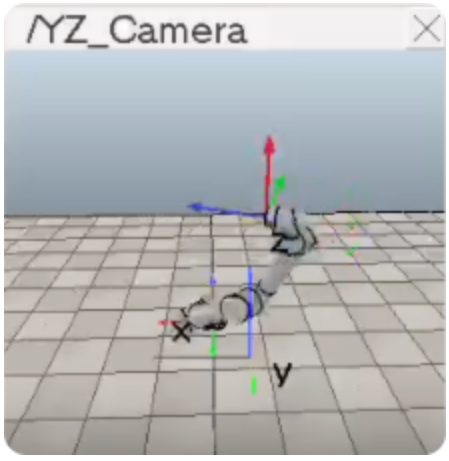


Part 3

In part 3, `t = 10`, `dt = 0.01`. The `referencePos` function return a constant spring position `[0, 1, 1]`
And the gravitation and spring rest length is set to be zero.

Part 3 (a)

`stiffness = 8`, `damping = 0`



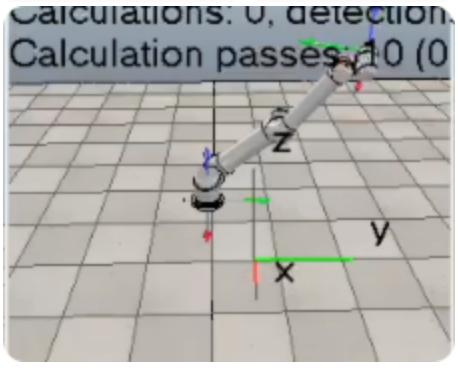
Does the motion make sense to you?
The arm's motion aligns with a force applied from the robot's end effector toward the spring target position at `[0, 1, 1]` in the space frame. The robot overshoots the target consistently, likely due to the inertia of each link in the open chain. The only unexpected aspect is the consistent rotation of the last joint (connected to the end effector), which may be caused by the rotational inertia in the final joints.

What do you expect to happen to the total energy over time?
Ideally, total energy should remain constant, as there are no external energy inputs and no damping forces to dissipate energy. The sum of kinetic and spring potential energy should be conserved. However, the observed increase in energy over time might be due to numerical errors in the method we're using.

Do you see any strange behavior if you choose the spring constant to be large?
With a high spring constant (e.g., 30), the robot arm's motion no longer oscillates predictably around the spring target. Initially, the arm moves toward the target, but then it becomes more erratic, with the end effector accelerating in multiple directions rapidly. This behavior may be due to increased energy in the system. Higher spring forces lead to larger joint accelerations, which could amplify errors in the Euler integration step, especially with greater spring stiffness.

Part 3 (b)

`stiffness = 8`, `damping = 3`



Part 4

In part 4, `referencePos` function returns a varied position for the spring, sinusoidally oscillate along a line from `[1, 1, 1]` to `[1, -1, 1]`.
And the spring stiffness and damping is same as Part 3 (b) `stiffness = 8`, `damping = 3`.
`t = 10`, `dt = 0.01`
And the gravitation and spring rest length is set to be zero.