**mechae263C_homework3.py**

```python
"""
IMPORTANT NOTE:
    The instructions for completing this template are inline with the code. You can
    find them by searching for: "TODO:"
"""

import math

import numpy as np
from matplotlib import pyplot as plt
from numpy.typing import NDArray
from pydrake.systems.analysis import Simulator
from pydrake.systems.framework import DiagramBuilder, Diagram, Context
from pydrake.systems.primitives import MatrixGain, LogVectorOutput

from mechae263C_helpers.drake import LinearCombination, plot_diagram
from mechae263C_helpers.hw3.arm import Arm, Gravity


def run_simulation(
    q_desired: NDArray[np.double],
) -> tuple[NDArray[np.double], NDArray[np.double], Diagram]:
    """
    Runs a simulation with a desired joint position

    Parameters
    ----------
    q_desired:
        A numpy array of shape (2,) containing the desired joint positions

    Returns
    -------
    A tuple with three elements:
        1. A numpy array with shape (T,) of simulation time steps
        2. A numpy array with shape (2, T) of joint positions corresponding to each
           simulation time step
        3. A Drake diagram
    """
    # Calculate the initial joint position
    q_initial = q_desired - 0.1

    # ------------------------------------------------------------------------------
    # Add "systems" to a `DiagramBuilder` object.
    #   - "systems" are the blocks in a block diagram
    #   - Some examples for how to add named systems to a `DiagramBuilder` are given
    #     below
    #
    # TODO:
```

```
    #   Replace any `...` with the correct block
    # ------------------------------------------------------------------------------
    builder = DiagramBuilder()

    K_p_gain = builder.AddNamedSystem(
        "K_p", MatrixGain(np.asarray(K_p, dtype=np.double))
    )
    K_d_gain = builder.AddNamedSystem(
        "K_d", MatrixGain(np.asarray(K_d, dtype=np.double))
    )
    position_error = builder.AddNamedSystem(
        "position_error", LinearCombination(input_coeffs=(1, -1), input_shapes=(2,))
    )
    input_torque = builder.AddNamedSystem(
        "input_torque", LinearCombination(input_coeffs=(-1, 1, 1), input_shapes=(2,))
    )
    arm = builder.AddNamedSystem("arm", Arm(F_v=F_v))
    gravity = builder.AddNamedSystem("gravity", Gravity(dyn_params=arm.dyn_params))

    # ------------------------------------------------------------------------------
    # Connect the systems in the `DiagramBuilder` (i.e. add arrows of block diagram)
    # ------------------------------------------------------------------------------
    # `builder.ExportInput(input_port)` makes the provided "input_port" into an input
    # of the entire diagram
    # The functions system.get_input_port() returns the input port of the given system
    #   - If there is more than one input port, you must specify the index of the
    #     desired input
    # The functions system.get_output_port() returns the output port of the given system
    #   - If there is more than one output port, you must specify the index of the
    #     desired output
    builder.ExportInput(position_error.get_input_port(0), name="q_d")

    joint_velocity_output = arm.get_output_port(0)
    joint_position_output = arm.get_output_port(1)

    # TODO:
    #   Replace any `...` below with the correct system and values. Please keep the
    #   system names the same
    builder.Connect(position_error.get_output_port(), K_p_gain.get_input_port())

    builder.Connect(K_p_gain.get_output_port(), input_torque.get_input_port(1))
    builder.Connect(input_torque.get_output_port(), arm.get_input_port())

    builder.Connect(joint_velocity_output, K_d_gain.get_input_port())
    builder.Connect(K_d_gain.get_output_port(), input_torque.get_input_port(0))

    builder.Connect(joint_position_output, position_error.get_input_port(1))
    builder.Connect(joint_position_output, gravity.get_input_port())
    builder.Connect(gravity.get_output_port(), input_torque.get_input_port(2))

```

```
    # ------------------------------------------------------------------------------
    # Log joint positions
    # ------------------------------------------------------------------------------
    # These systems are special in Drake. They periodically save the output port value
    # a during a simulation so that it can be accessed later. The value is saved every
    # `publish_period` seconds in simulation time.
    joint_position_logger = LogVectorOutput(
        arm.get_output_port(1), builder, publish_period=1e-3
    )

    # ------------------------------------------------------------------------------
    # Setup/Run the simulation
    # ------------------------------------------------------------------------------
    # This line builds a `Diagram` object and uses it to make a `Simulator` object for
    # the diagram
    diagram: Diagram = builder.Build()
    diagram.set_name("PD w/ Gravity Compensation")
    simulator: Simulator = Simulator(diagram)

    # Get the context (this contains all the information needed to run the simulation)
    context: Context = simulator.get_mutable_context()

    # Set initial conditions
    initial_conditions = context.get_mutable_continuous_state_vector()
    initial_conditions.SetAtIndex(2, q_initial[0])
    initial_conditions.SetAtIndex(3, q_initial[1])

    # TODO:
    #   Replace the `...` below with the correct fixed value to simulate a desired joint
    #   position vector of `q_desired`
    diagram.get_input_port().FixValue(context, q_desired)

    # Advance the simulation by `simulation_duration_s` seconds using the
    # `simulator.AdvanceTo()` function
    simulator.AdvanceTo(simulation_duration_s)

    # ------------------------------------------------------------------------------
    # Extract simulation outputs
    # ------------------------------------------------------------------------------
    # The lines below extract the joint position log from the simulator context
    joint_position_log = joint_position_logger.FindLog(simulator.get_context())
    t = joint_position_log.sample_times()
    q = joint_position_log.data()

    # Return a `tuple` of simulation times, simulated joint positions, and the Drake
    # diagram
    return t, q, diagram


if __name__ == "__main__":
```

```
    # ------------------------------------------------------------------------------
    # TODO:
    #   Replace `...` with the correct values for each parameter
    # ------------------------------------------------------------------------------
    # The below functions might be helpful:
    #   np.diag: https://numpy.org/doc/stable/reference/generated/numpy.diag.html
    #   np.eye: https://numpy.org/doc/stable/reference/generated/numpy.eye.html
    #
    # Hint:
    #   The `@` operator can be used to multiply to numpy arrays A and B via: `A @ B`
    K_r = np.asarray([[100, 0], [0, 100]])
    Fm = np.asarray([[0.01, 0], [0, 0.01]])
    F_v = K_r @ Fm @ K_r

    # K_p = np.asarray([[250, 0], [0, 250]])
    # K_d = np.asarray([[150, 0], [0, 150]])
    K_p = np.asarray([[8000, 0], [0, 8000]])
    K_d = np.asarray([[3070, 0], [0, 3070]])

    q_d_case1 = np.asarray([np.pi / 4, np.pi * (-0.5)])
    q_d_case2 = np.asarray([-(np.pi), np.pi * (-3/4)])

    simulation_duration_s = 2.5

    # ------------------------------------------------------------------------------
    # Run the simulations for each case
    # ------------------------------------------------------------------------------
    t_case1, q_case1, diagram = run_simulation(q_desired=q_d_case1)
    t_case2, q_case2, diagram2 = run_simulation(q_desired=q_d_case2)
    print("Finish data")

    # ------------------------------------------------------------------------------
    # Make Plots
    # ------------------------------------------------------------------------------
    # TODO:
    #   Replace `...` with the file name of to save the diagram plot to
    #   (e.g. diagram.png)
    # fig, axp = plot_diagram(diagram)
    # fig.savefig("sim_diagram.png", dpi=300)
    # print("Finish diagram")

    # Plot for Case 1
    fig = plt.figure(figsize=(12, 5))
    ax0 = fig.add_subplot(1, 2, 1)
    ax1 = fig.add_subplot(1, 2, 2)
    ax0.axhline(q_d_case1[0], ls="--", color="k")
    ax0.plot(t_case1, q_case1[0, :])
    ax1.axhline(q_d_case1[1], ls="--", color="k")
    ax1.plot(t_case1, q_case1[1, :])
    # TODO:
```

```
    #   Replace occurrences of "..." with code to set the x labels, y labels, and title
    #   of the plot.
    #   https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.set_xlabel.html
    #   and
    #   https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.set_ylabel.html
    ax0.set_xlabel('Time [s]')
    ax0.set_ylabel('Position [rad]')
    ax1.set_xlabel('Time [s]')
    ax1.set_ylabel('Position [rad]')
    ax0.set_title('First Position: Joint 1')
    ax1.set_title('First Position: Joint 2')

    # TODO:
    #   Replace occurrences of "..." with code to save your figure
    #   See:
    #   https://matplotlib.org/stable/api/figure_api.html#matplotlib.figure.Figure.savefig
    #
    # Hint:
    #   To increase resolution of your saved plots you can pass the `dpi` argument to
    #   `Figure.savefig` with a high value (ex. 300).
    fig.savefig('case1.png', dpi=300)
    print("Finish case 1")

    # Plot for Case 2
    # TODO:
    #   Replace `...` with the code to plot the joint positions vs time for the second
    #   desired joint position case (i.e. q_d_case2 vs time)
    fig2 = plt.figure(figsize=(12, 5))
    ax02 = fig2.add_subplot(1, 2, 1)
    ax12 = fig2.add_subplot(1, 2, 2)
    ax02.axhline(q_d_case2[0], ls="--", color="k")
    ax02.plot(t_case1, q_case2[0, :])
    ax12.axhline(q_d_case2[1], ls="--", color="k")
    ax12.plot(t_case2, q_case2[1, :])

    ax02.set_xlabel('Time [s]')
    ax02.set_ylabel('Position [rad]')
    ax12.set_xlabel('Time [s]')
    ax12.set_ylabel('Position [rad]')
    ax02.set_title('Second Position: Joint 1')
    ax12.set_title('Second Position: Joint 2')

    fig2.savefig('case2.png', dpi=300)
    print("Finish case 2")

    #plt.show()

```