mechae263C_homework2.py

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

import matplotlib.pyplot as plt
import numpy as np
from numpy.typing import NDArray
import math

import pyvista as pv
pv.OFF_SCREEN = True
pv.start_xvfb()

from mechae263C_helpers.hw2 import (
    generate_points_on_unit_sphere,
    plot_ellipsoids,
    calc_fk_2D,
    calc_ellipsoid_projection,
)


def calc_jacobian(
    link_lens: NDArray[np.double], config: NDArray[np.double]
) -> NDArray[np.double]:
    """
    Calculate the geometric Jacobian specified in the problem statement of Homework #2

    Parameters
    ----------
    link_lens
        A NumPy array of shape (3,) that specifies the link lengths (starting from the
        base) of the planar 3R manipulator.

    config
        A NumPy array of shape (3,) that specifies the joint angles (starting from the
        first joint) of the planar 3R manipulator.

    Returns
    -------
    A NumPy array of shape (3, 3) (i.e. a 3 by 3 matrix) representing the geometric
    Jacobian specified in the problem statement of Homework #2
    """
    # Some helpful functions:
    #   `math.cos(angle_in_rad)` (you will need to import the `math` module first)
    #   `math.sin(angle_in_rad)` (you will need to import the `math` module first)
    #   `np.ones((3, 3))` (This will return a new 3 by 3 NumPy array filled with ones)
```

```python
    #
    # Also see:
    #   https://numpy.org/doc/stable/user/quickstart.html
    # for a quick introduction to NumPy

    # TODO:
    #   Replace "..." below so that the function returns the correct value (i.e. the
    #   Geometric Jacobian specified in Homework #2)

    # L1 = link_lens[0]
    # L2 = link_lens[1]
    # L3 = link_lens[2]

    Jac = np.empty((3,3))
    Jac[0,0] = -(link_lens[0]*math.sin(config[0]) + link_lens[1]*math.sin(config[0]+config[1]) + link_lens[2]*math.sin(config[0]+config[1]+config[2]))
    Jac[0,1] = -(link_lens[1]*math.sin(config[0]+config[1]) + link_lens[2]*math.sin(config[0]+config[1]+config[2]))
    Jac[0,2] = -(link_lens[2]*math.sin(config[0]+config[1]+config[2]))

    Jac[1,0] = (link_lens[0]*math.cos(config[0]) + link_lens[1]*math.cos(config[0]+config[1]) + link_lens[2]*math.cos(config[0]+config[1]+config[2]))
    Jac[1,1] = (link_lens[1]*math.cos(config[0]+config[1]) + link_lens[2]*math.cos(config[0]+config[1]+config[2]))
    Jac[1,2] = (link_lens[2]*math.cos(config[0]+config[1]+config[2]))

    Jac[2,0] = 1
    Jac[2,1] = 1
    Jac[2,2] = 1

    return Jac


if __name__ == "__main__":
    # TODO:
    # Make sure to check out:
    #   https://numpy.org/doc/stable/user/quickstart.html
    # for a quick introduction to NumPy!
    # NumPy is third-party Python package for multidimensional arrays (and linear
    # algebra) that is **heavily** used in the field of robotics (and many other
    # fields).

    # This changes how NumPy arrays are formatted for use with  the built-in `print`
    # function
    # - `supress=True` prevents NumPy from formatting numbers with exponential notation
    # (unless the numbers are huge or tiny)
    # - `precision=4` makes NumPy format numbers with four digits of precision
    #
    np.set_printoptions(suppress=True, precision=4, floatmode="fixed")

```

```python
    # Generate points on a unit sphere
    # `sphere_points` is a NumPy array of shape (N, 3) that contains N three-dimensional
    # points. Each index along the first axis of `sphere_points` corresponds to a
    # distinct point. The indices along the second axis of `sphere_points` correspond
    # to the x, y, and z coordinates of the points, in that order.
    #
    # You can check the "shape" of a NumPy array via the `shape` property. See:
    #   https://numpy.org/doc/stable/reference/generated/numpy.ndarray.shape.html
    # for more details.
    sphere_points = generate_points_on_unit_sphere()
    print("Number of Sphere Points:", sphere_points.shape[0])

    # ---------------------------------------------------------------------------------
    # Define Configurations and Link Lengths
    # ---------------------------------------------------------------------------------
    # TODO:
    #   Replace all occurrences "..." with code to make 1D NumPy arrays with correct
    #   link lengths and configurations (i.e. the joint positions for each case
    #   specified in Homework #2)
    #   Note: All arrays should start with the quantity closest to the base of the
    #         robot (i.e. the order for link lengths should be [L1, L2, L3]).
    # Make numpy array and convert from deg to rad
    config0 = np.deg2rad([0, -0.05, 0])
    config1 = np.deg2rad([-22.5, -22.5, -45])
    config2 = np.deg2rad([-45, -67.5, -67.5])

    link_lens = np.asarray([2, 1, 0.75])

    configs = [config0, config1, config2]

    # ---------------------------------------------------------------------------------
    # Make 2D Plots
    # ---------------------------------------------------------------------------------
    # The below code makes a new `plt.Figure` object with id 1 and stores it in the
    # variable `fig_velocity_ellipses`.
    # See: https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.figure.html
    fig_velocity_ellipses = plt.figure(1)  # This makes a figure with id 1

    # TODO:
    #   Replace "..." with code to make a figure with id 2
    fig_force_ellipses = plt.figure(2)  # This makes a figure with id 2

    # This below code adds a `plt.Axes` object on the `fig_velocity_ellipses` figure
    # and stores it in the variable `ax_vel`
    # See:
https://matplotlib.org/stable/api/figure_api.html#matplotlib.figure.Figure.add_subplot
    ax_vel = fig_velocity_ellipses.add_subplot(1, 1, 1)

    # TODO:
    #   Replace "..." with code to make `plt.Axes` object on the figure for the force
```

```
    #   ellipses (`fig_force_ellipses`).
    ax_force = fig_force_ellipses.add_subplot(1, 1, 1)
    #ax_force = plt.Axes(fig_force_ellipses, (-3.75,-3.75,3.75,3.75))

    # This loop iterates over the list of configurations (`configs`) and uses the
    # `enumerate` function so that each iteration also has access to the index of the
    # iteration (i.e. 0, 1, 2, ...).
    # See: https://docs.python.org/3.10/library/functions.html#enumerate
    for i, config in enumerate(configs):
        print(f"Configuration {i}:")
        # ------------------------------------------------------------------------------
        # Calculate Jacobian, Jacobian Transpose Inverse, and Singular Values
        # ------------------------------------------------------------------------------
        # TODO:
        #   Replace "..." with the code to call your `calc_jacobian` function and pass
        #   it the arguments `link_lens` and `config` (in the order specified by the
        #   function).
        #print(config)
        J = calc_jacobian(link_lens, config)
        print("Associated Jacobian")
        print(J)

        # TODO:
        # Replace "..." with the code to calculate Inverse Transpose of Jacobian
        #   See:
        #   https://numpy.org/doc/stable/reference/generated/numpy.linalg.inv.html
        #   and
        #   https://numpy.org/doc/stable/reference/generated/numpy.ndarray.T.html
        J_inv = np.linalg.inv(J)
        J_Tinv = J_inv.T

        # TODO:
        #   Replace "..." with the code to calculate singular values of Jacobian and
        #   Jacobian Inverse Transpose
        #   See:
        #   https://numpy.org/doc/stable/reference/generated/numpy.linalg.svd.html
        #   Hint: If you just want singular values returned pass `False` for the
        #         `compute_uv` argument to `np.linalg.svd`
        singular_values_J = np.linalg.svd(J, compute_uv=False)
        singular_values_JTinv = np.linalg.svd(J_Tinv, compute_uv=False)

        print("\tSingular Values of J:\t\t", singular_values_J)
        print("\tSingular Values of J_Tinv:\t", singular_values_JTinv, "\n")

        # ------------------------------------------------------------------------------
        # Calculate Jacobian, Jacobian Transpose Inverse, and Singular Values
        # ------------------------------------------------------------------------------
        # TODO:
        #   Replace occurrences of "..." with the code to multiply points on unit sphere
        #   (`sphere_points`) by Jacobian or Jacobian Inverse Transpose to get the
```

```
        #   appropriate manipulability ellipsoid.
        #   Hint: If `matNx3` is a (N by 3) array and `mat3x3` is a (3 x 3) array, then
        #         you can efficiently multiply every row of `matNx3` by `mat3x3` using:
        #         `result = (mat3x3 @ matNx3.T).T`
        #         where the `@` indicates matrix multiplication.
        velocity_ellipsoid = (J @ sphere_points.T).T
        force_ellipsoid = (J_Tinv @ sphere_points.T).T

        # The below code calls `calc_ellipsoid_projection` to calculate the boundary
        # of the 2D ellipses formed by projecting the 3D ellipsoids onto the xy plane.
        # In turn, the value returned by `calc_ellipsoid_projection` is NumPy array
        # of shape (N, 2)
        velocity_ellipse_points = calc_ellipsoid_projection(
            ellipsoid_points3D=velocity_ellipsoid
        )
        force_ellipse_points = calc_ellipsoid_projection(
            ellipsoid_points3D=force_ellipsoid
        )

        # The below code calculates the x and y positions of each joint of the planar 3R
        # manipulator given its link lengths and a configuration (aka joint positions)
        frame_x_positions, frame_y_positions = calc_fk_2D(
            link_lens=link_lens, config=config
        )

        # TODO:
        #   Replace occurrences of "..." below with code to plot the projected ellipses
        #   (i.e. `velocity_ellipse_points` and `force_ellipse_points`).
        #   Make sure to plot the ellipse points so that the ellipse center is at
        #   the position of the end effector.
        #
        # Hint:
        #   You can access the end-effector frame position coordinates via
        #   `frame_x_positions[-1]` and `frame_y_positions[-1]`. Negative indices in
        #   Python index an array starting from the end.
        #
        #   Also see:
        #   https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.plot.html

        ax_vel.plot(
            velocity_ellipse_points[:, 0] + frame_x_positions[-1],
            velocity_ellipse_points[:, 1] + frame_y_positions[-1],
            label=f"Configuration {i}"
        )
        ax_force.plot(
            force_ellipse_points[:, 0] + frame_x_positions[-1],
            force_ellipse_points[:, 1] + frame_y_positions[-1],
            label=f"Configuration {i}:"
        )

```

```
        # The code below plots the manipulator links
        ax_vel.plot(frame_x_positions, frame_y_positions, color="k")
        ax_force.plot(frame_x_positions, frame_y_positions, color="k")

        # ------------------------------------------------------------------------------
        # Make and save 3D Plots
        # ------------------------------------------------------------------------------
        # TODO:
        #   Replace "..." with a string containing the path with file name included
        #   where you want to save the 3D plots
        #
        # Note:
        #   Take a second to verify that your 3D plots are consistent with your
        #   intuition about singular configurations
        plot_name = "/workspaces/MAE263C_HW2/EllipsePlots/EllipseProjections_" +
f"Configuration {i}"

        plot_ellipsoids(
            axis_title=f"Configuration {i}",  # Add plot title
            velocity_ellipsoid_points3D=velocity_ellipsoid,
            force_ellipsoid_points3D=force_ellipsoid,
            file_path=plot_name,  # Add file path/name to save plot in
        )

    # ----------------------------------------------------------------------------------
    # Format Ellipse Plots
    # ----------------------------------------------------------------------------------
    # TODO:
    #   Replace occurrences of "..." with code to set the x and y limits of the plots.
    #   See:
    #   https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.set_xlim.html
    #   and
    #   https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.set_ylim.html
    ax_vel.set_xlim(-5, 8)  # Set y limits of velocity ellipse plot to range [-5, 8]
    ax_vel.set_ylim(-5, 5)  # Set y limits of velocity ellipse plot to range [-5, 5]
    ax_force.set_xlim(-5, 8)  # Set x limits of force ellipse plot to range [-5, 8]
    ax_force.set_ylim(-5, 5)  # Set y limits of force ellipse plot to range [-5, 5]

    # TODO:
    #   Replace occurrences of "..." with code to set the x and y labels 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
    ax_vel.set_xlabel('X Position [m]')  # Set x label of velocity ellipse plot
    ax_vel.set_ylabel('Y Position [m]')  # Set y label of velocity ellipse plot
    ax_force.set_xlabel('X Position [m]')  # Set x label of force ellipse plot
    ax_force.set_ylabel('Y Position [m]')  # Set y label of force ellipse plot

    # TODO:
    #   Replace occurrences of "..." with code to set title of plot.
```

```
    #   https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.set_title.html
    # Set title of velocity ellipse plot
    ax_vel.set_title('Velocity Ellipse Plot')
    ax_vel.legend()
    # Set title of force ellipse plot
    ax_force.set_title('Force Ellipse Plot')
    ax_force.legend()

    # TODO:
    #   Replace occurrences of "..." with code to save your figures
    #   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).
    # Save velocity ellipse plot
    fig_velocity_ellipses.savefig("Velocity Ellipse Plot", dpi=300)

    # Save force ellipse plot
    fig_force_ellipses.savefig("Force Ellipse Plot", dpi=300)

    # Show the ellipse plots
    #plt.show()
```