mechae263C_helpers/src/mechae263C_helpers/hw2/__init__.py

```python
import itertools
import math

import numpy as np
from numpy.typing import NDArray
from pyvista.plotting import Plotter
import pyvista as pv
from scipy.spatial import ConvexHull


def generate_points_on_unit_sphere(theta_grid_size: int = 30, phi_grid_size: int = 30):
    """
    Generates points on a unit sphere

    Parameters
    ----------
    theta_grid_size
        An integer representing the number of grid steps to use for discretizing the
        theta spherical coordinate

    phi_grid_size
        An integer representing the number of grid steps to use for discretizing the
        phi spherical coordinate

    Returns
    -------
    A NumPy array of shape (N, 3) containing N 3D points on the surface of a unit sphere
    """
    return pv.Sphere(
        radius=1, theta_resolution=theta_grid_size, phi_resolution=phi_grid_size
    ).points


def plot_ellipsoids(
    axis_title: str,
    force_ellipsoid_points3D: NDArray[np.double],
    velocity_ellipsoid_points3D: NDArray[np.double],
    file_path: str | None = None,
    title_font_size: int = 30,
    axes_font_size: int = 20,
    camera_zoom: float = 0.8,
    window_size: tuple[int, int] = (2000, 2000),
):
    """
    Renders, displays, and (optionally) saves a 3D plot of 3D force and velocity
    ellipsoids

    Parameters
```

```
    ----------
    axis_title
        A string representing the axis title to use for the 3D plot

    velocity_ellipsoid_points3D
        A NumPy array of shape (N, 3) representing the points on a 3D velocity ellipsoid

    force_ellipsoid_points3D
        A NumPy array of shape (M, 3) representing the points on a 3D force ellipsoid

    file_path
        Either a string representing the file path (including file name) in which to
        save a screenshot of the 3D plot or `None`. If `None` is provided, then no
        screenshot is saved.

    title_font_size
        An integer representing the font size of the axis title

    axes_font_size
        An integer representing the font size of the axes labels

    camera_zoom
        A single float representing the initial zoom of 3D plot

    window_size
        A tuple of two integers containing the size of the plot window in pixels (width,
        height)
    """

    plotter: Plotter = pv.Plotter()
    colors = [[255, 0, 0], [0, 0, 255]]

    ellipsoids = [velocity_ellipsoid_points3D, force_ellipsoid_points3D]

    ptps = np.zeros((len(ellipsoids), 3))
    minima = np.zeros((len(ellipsoids), 3))
    maxima = np.zeros((len(ellipsoids), 3))

    for p, ellipsoid in enumerate(ellipsoids):
        for j in range(3):
            ptps[p, j] = ellipsoid[:, j].ptp()
            minima[p, j] = ellipsoid[:, j].min()
            maxima[p, j] = ellipsoid[:, j].max()

    max_ptp = np.max(ptps)
    for p, ellipsoid in enumerate(ellipsoids):
        scale_mat = np.diag(
            [max_ptp / ptps[p, 0], max_ptp / ptps[p, 1], max_ptp / ptps[p, 2]]
        )

```

```
        ellipsoid_data = (
            pv.PolyData((scale_mat @ ellipsoid.T).T)
            .delaunay_3d()
            .extract_surface(nonlinear_subdivision=1)
        )
        plotter.add_mesh(
            ellipsoid_data,
            color=colors[p % len(ellipsoids)],
            opacity=1,
            smooth_shading=True,
        )

    plotter.add_title(axis_title, font_size=title_font_size)
    plotter.show_grid(
        # xtitle=r"$\dot{x}$ or $F_x$",
        # ytitle=r"$\dot{y}$ or $F_y$",
        # ztitle=r"$\dot{\phi}$ or $M_z$",
        xtitle="X component: x-dot [m/s], f_x [N]",
        ytitle="Y component: y-dot [m/s], f_y [N]",
        ztitle="phi-dot [rad/s], M_z [Nm]",
        font_size=axes_font_size,
        grid="back",
        location="outer",
        ticks="both",
        axes_ranges=list(
            itertools.chain(*zip(np.min(minima, axis=0), np.max(maxima, axis=0)))
        ),
    )
    plotter.show_axes()

    plotter.window_size = window_size
    plotter.view_isometric()
    plotter.camera.zoom(camera_zoom)

    if file_path is None:
        plotter.show(auto_close=False, interactive=False)
    else:
        plotter.show(auto_close=False, interactive=False, screenshot=file_path + "_view_iso")

    plotter.view_xy()
    if file_path is None:
        plotter.show(auto_close=False, interactive=False)
    else:
        plotter.show(auto_close=False, interactive=False, screenshot=file_path + "_view_xy")


def calc_ellipsoid_projection(
    ellipsoid_points3D: NDArray[np.double], axes: tuple[int, int] = (0, 1)
):
    """
```

```
    Computes the points on the boundary of the projection of a 3D ellipsoid in the
    plane.

    Parameters
    ----------
    ellipsoid_points3D
        A NumPy array of shape (N, 3) representing the N points on a 3D ellipsoid to
        project

    axes
        A tuple of two integers specifying the axes which to project the 3D ellipsoid
        points onto. Defaults to (0, 1).

    Returns
    -------
    A NumPy array of shape (M, 2) containing the 2D ellipse points on the boundary of
    the projected 3D ellipsoid points.
    """
    cv = ConvexHull(ellipsoid_points3D[:, axes[:2]])
    points = cv.points
    vertex_ixs = cv.vertices
    vertex_ixs = np.append(vertex_ixs, vertex_ixs[0])
    return np.stack((points[vertex_ixs, 0], points[vertex_ixs, 1]), axis=1)


def calc_fk_2D(
    link_lens: NDArray[np.double], config: NDArray[np.double]
) -> tuple[NDArray[np.double], NDArray[np.double]]:
    """
    Calculates the x and y positions of all four frames of a planar 3R manipulator
    (including end effector and base joint) given manipulator link lengths and a
    configuration

    Parameters
    ----------
    link_lens
        A NumPy array of shape (3,) containing the three link lengths of the planar 3R
        manipulator (ordered from link 1 to link 3)

    config
        A NumPy array of shape (3,) containing the three joint positions of the planar
        3R manipulator (ordered from theta1 to theta3)

    Returns
    -------
    A tuple of two NumPy array both with shape (4,). The first and second element of the
    tuple contain the x coordinates and y coordinates for the origin of each frame
    (joints and end-effector) of the planar 3R manipulator, respectively.
    """
    joint_xs, joint_ys = np.zeros((4,)), np.zeros((4,))
```

```
199
200     angle_sum = 0
201     for i, L in enumerate(link_lens):
202         angle_sum += config[i]
203         joint_xs[i + 1] = joint_xs[i] + L * math.cos(angle_sum)
204         joint_ys[i + 1] = joint_ys[i] + L * math.sin(angle_sum)
205
206     return np.asarray(joint_xs), np.asarray(joint_ys)
207
```