noising_trajectory.py

```python
# Main-Script
# #############################################################################

# This python script automatically launches all other python scripts in the
# right order and computes the entire task.

# Authors:
# Christopher Mahn
# Silas Teske
# Joshua Wolf

# #############################################################################

# Import of Libraries
# -----------------------------------------------------------------------------

import main as settings
# import string as st
# import random as r
# import re
# from turtle import position
# from scipy import interpolate
# from concurrent.futures import process
# from turtle import position
import numpy as np
import math as m
# import sys
import os
# import matplotlib.pyplot as plt
# from scipy.fft import fft, fftfreq
# from scipy import signal
# import multiprocessing as mp
# import copy
import lib_trajectory as t


# -----------------------------------------------------------------------------
# Debugging-Settings

verbose = True  # Shows more debugging information


# Functions
# -----------------------------------------------------------------------------


def scale_trajectory(trajectory, mean, stdev):
    """
    This function scales a trajectory with a mean and standard deviation.

    Args:
        trajectory ([Line]): The trajectory consisting of Line-objects in a list.
        mean (float): The mean scaling factor for each step.
        stdev (float): the standard deviation for the scaling factor of each step.
    """
    if(verbose):
        print(f'[INFO] Scaling trajectory')
    trajectory_new = []
    while(trajectory != []):
        # Selecting and scaling one line at a time
        current_line = trajectory.pop(0)
        current_scale = np.random.normal(mean, stdev)
        trajectory_new.append(t.Line(current_line.x1(),
                                     current_line.y1(),
                                     current_line.x1() + (current_line.delta_x()*current_scale),
                                     current_line.y1() + (current_line.delta_y()*current_scale)))
        delta_x = (current_line.x1() + (current_line.delta_x()*current_scale)) - current_line.x2()
        delta_y = (current_line.y1() + (current_line.delta_y()*current_scale)) - current_line.y2()

        # Scaling the rest of the lines attached to the current line
        if(trajectory != []):
            for index, line in enumerate(trajectory):
                trajectory[index] = t.Line(line.x1()+delta_x,
                                           line.y1()+delta_y,
                                           line.x2()+delta_x,
                                           line.y2()+delta_y)
    return(trajectory_new)


def rotate_trajectory(trajectory, mean, stdev):
    """
    This function rotates a trajectory with a mean and standard deviation.

    Args:
        trajectory ([Line]): The trajectory consisting of Line-objects in a list.
        mean (float): The mean rotation factor for each step.
        stdev (float): the standard deviation for the rotation factor of each step.
    """
    if(verbose):
        print(f'[INFO] Rotating trajectory')
    trajectory_new = []
    while(trajectory != []):
        # Selecting and rotating one line at a time
        current_line = trajectory.pop(0)
        current_rotation = np.random.normal(mean, stdev)
        trajectory_new.append(t.Line(current_line.x1(),
                                     current_line.y1(),
                                     current_line.x1()+(m.cos(current_line.direction()+current_rotation)*current_line.length()),
                                     current_line.y1()+(m.sin(current_line.direction()+current_rotation)*current_line.length())))
        delta_x = trajectory_new[-1].x2()-current_line.x2()
        delta_y = trajectory_new[-1].y2()-current_line.y2()

        # Scaling the rest of the lines attached to the current line
        if(trajectory != []):
            for index, line in enumerate(trajectory):
                trajectory[index] = t.Line(line.x1()+delta_x,
                                           line.y1()+delta_y,
                                           line.x1()+delta_x+m.cos(line.direction()+current_rotation)*line.length(),
                                           line.y1()+delta_y+m.sin(line.direction()+current_rotation)*line.length())
                delta_x = (line.x1()+delta_x+m.cos(line.direction()+current_rotation)*line.length())-line.x2()
                delta_y = (line.y1()+delta_y+m.sin(line.direction()+current_rotation)*line.length())-line.y2()
    return(trajectory_new)


def write_noise_information(smean, sstd, rmean, rstd, filename):
    """
    This function writes the virtual noise-information that has been generated
    to a file.

    Args:
        smean (float): the value of the mean scaling-factor
        sstd (float): the value of the standard-deviation of the scaling-factor
        rmean (float): the value of the mean rotation
        rstd (float): the value of the standard-deviation of the rotation
        filename (str): the filename where the text is written to. (Including file-extension)
    """
    if(verbose):
        print(f'[INFO] Writing noise-information to "{filename}"')
    with open(os.path.join("data", f'{filename}'), "w") as f:
        f.write(f'distance; mean; {smean}\n')
        f.write(f'distance; sdev; {sstd}\n')
        f.write(f'rotation; mean; {rmean}\n')
        f.write(f'rotation; sdev; {rstd}\n')
    return(None)


# Classes
# ----------------------------------------------------------------------------


# Beginning of the Programm
# ----------------------------------------------------------------------------

if __name__ == '__main__':
    # Dataset-Information
    projectnames = settings.project_filenames
    dataset_length = settings.trajectories_per_project
    training_data_length = settings.datasets_per_trajectory  # Number of noised trajectories to generate for training

    # Import of individual trajectories
    for dataset_index, projectname in enumerate(projectnames):
        for trajectory_index in range(dataset_length):
            trajectory = t.lines_import(f'trajectory_{projectname}_{trajectory_index+1:05d}_ground-truth.csv')

            # Generating sensor-noise parameters
            rotation_drift = np.random.normal(0, 0.1/200*m.pi)  # One-sided drift at each step
            rotation_noise = abs(np.random.normal(0, 0.2/200*m.pi))  # Rotation angle noise
            step_length_scale = np.random.normal(1, 0.05)  # Scale of each step
            step_length_noise = abs(np.random.normal(0, 0.025))  # Random scale of each step
            if(verbose):
                print(f'[INFO][CONFIG] Rotation: {rotation_drift:.5f} ± {rotation_noise:.5f} rad, Scale: {step_length_scale:.5f} ± {step_length_noise:.5f} x')
            write_noise_information(step_length_scale, step_length_noise, rotation_drift, rotation_noise, f'trajectory_{projectname}_{trajectory_index+1:05d}_applied-noise.log')

            # Generating and exporting of the noised-trajectories.
            for noise_index in range(training_data_length):
                noised_trajectory = scale_trajectory(trajectory.copy(), step_length_scale, step_length_noise)
                noised_trajectory = rotate_trajectory(noised_trajectory, rotation_drift, rotation_noise)
                t.lines_export(noised_trajectory, f'trajectory_{projectname}_{trajectory_index+1:05d}_training_{noise_index+1:05d}.csv')
                del noised_trajectory
```