

noising_trajectory.py

```
001 # Main-Script
002 # #####
003
004 # This python script automatically launches all other python scripts in the
005 # right order and computes the entire task.
006
007 # Authors:
008 # Christopher Mahn
009 # Silas Teske
010 # Joshua Wolf
011
012 # #####
013
014 # Import of Libraries
015 # -----
016
017 import main as settings
018 # import string as st
019 # import random as r
020 # import re
021 # from turtle import position
022 # from scipy import interpolate
023 # from concurrent.futures import process
024 # from turtle import position
025 import numpy as np
026 import math as m
027 # import sys
028 import os
029 # import matplotlib.pyplot as plt
030 # from scipy.fft import fft, fftfreq
031 # from scipy import signal
032 # import multiprocessing as mp
033 # import copy
034 import lib_trajectory as t
035
036
037 # -----
038 # Debugging-Settings
039
040 verbose = True # Shows more debugging information
041
042
043 # Functions
044 # -----
045
046
047 def scale_trajectory(trajecotory, mean, stdev):
048     """
049     This function scales a trajectory with a mean and standard deviation.
050
051     Args:
052         trajecotory ([Line]): The trajectory consisting of Line-objects in a list.
053         mean (float): The mean scaling factor for each step.
054         stdev (float): the standard deviation for the scaling factor of each step.
055     """
056     if(verbose):
057         print(f'[INFO] Scaling trajectory')
058     trajecotory_new = []
059     while(trajecotory != []):
060         # Selecting and scaling one line at a time
061         current_line = trajecotory.pop(0)
062         current_scale = np.random.normal(mean, stdev)
063         trajecotory_new.append(t.Line(current_line.x1(),
064                                     current_line.y1(),
065                                     current_line.x1() + (current_line.delta_x()*current_scale),
066                                     current_line.y1() + (current_line.delta_y()*current_scale)))
067         delta_x = (current_line.x1() + (current_line.delta_x()*current_scale)) - current_line.x2()
068         delta_y = (current_line.y1() + (current_line.delta_y()*current_scale)) - current_line.y2()
069
070         # Scaling the rest of the lines attached to the current line
071         if(trajecotory != []):
072             for index, line in enumerate(trajecotory):
073                 trajecotory[index] = t.Line(line.x1()+delta_x,
074                                             line.y1()+delta_y,
075                                             line.x2()+delta_x,
076                                             line.y2()+delta_y)
077     return(trajecotory_new)
078
079
080 def rotate_trajectory(trajecotory, mean, stdev):
```

```

081     """
082     This function rotates a trajectory with a mean and standard deviation.
083
084     Args:
085         trajectory ([Line]): The trajectory consisting of Line-objects in a list.
086         mean (float): The mean rotation factor for each step.
087         stdev (float): the standard deviation for the rotation factor of each step.
088     """
089     if(verbose):
090         print(f'[INFO] Rotating trajectory')
091     trajectory_new = []
092     while(trajectory != []):
093         # Selecting and rotating one line at a time
094         current_line = trajectory.pop(0)
095         current_rotation = np.random.normal(mean, stdev)
096         trajectory_new.append(t.Line(current_line.x1(),
097                                     current_line.y1(),
098                                     +current_line.x1()+(m.cos(current_line.direction()
099                                     +current_rotation)*current_line.length()),
100                                     current_line.y1()+(m.sin(current_line.direction()
101                                     +current_rotation)*current_line.length()))
102         delta_x = trajectory_new[-1].x2()-current_line.x2()
103         delta_y = trajectory_new[-1].y2()-current_line.y2()
104
105         # Scaling the rest of the lines attached to the current line
106         if(trajectory != []):
107             for index, line in enumerate(trajectory):
108                 trajectory[index] = t.Line(line.x1()+delta_x,
109                                             line.y1()+delta_y,
110                                             +current_line.x1()+delta_x+m.cos(line.direction()
111                                             +current_rotation)*line.length(),
112                                             line.y1()+delta_y+m.sin(line.direction()
113                                             +current_rotation)*line.length())
114                 delta_x = (line.x1()+delta_x+m.cos(line.direction()+current_rotation)*line.length())-
115                 line.x2()
116                 delta_y = (line.y1()+delta_y+m.sin(line.direction()+current_rotation)*line.length())-
117                 line.y2()
118     return(trajectory_new)
119
120 def write_noise_information(smean, sstd, rmean, rstd, filename):
121     """
122     This function writes the virtual noise-information that has been generated
123     to a file.
124
125     Args:
126         smean (float): the value of the mean scaling-factor
127         sstd (float): the value of the standard-deviation of the scaling-factor
128         rmean (float): the value of the mean rotation
129         rstd (float): the value of the standard-deviation of the rotation
130         filename (str): the filename where the text is written to. (Including file-extension)
131     """
132     if(verbose):
133         print(f'[INFO] Writing noise-information to "{filename}"')
134     with open(os.path.join("data", f'{filename}'), "w") as f:
135         f.write(f'distance; mean; {smean}\n')
136         f.write(f'distance; sdev; {sstd}\n')
137         f.write(f'rotation; mean; {rmean}\n')
138         f.write(f'rotation; sdev; {rstd}\n')
139     return(None)
140
141 # Classes
142 # -----
143
144 # Beginning of the Programm
145 # -----
146
147 if __name__ == '__main__':
148     # Dataset-Information
149     projectnames = settings.project_filenames
150     dataset_length = settings.trajectories_per_project
151     training_data_length = settings.datasets_per_trajectory # Number of noised trajectories to
152     generate for training
153
154     # Import of individual trajectories
155     for dataset_index, projectname in enumerate(projectnames):
156         for trajectory_index in range(dataset_length):
157             trajectory = t.lines_import(f'trajectory_{projectname}_{trajectory_index+1:05d}_ground-
158             truth.csv')
159
160             # Generating sensor-noise parameters

```

```

156         rotation_drift = np.random.normal(0, 0.1/200*m.pi) # One-sided drift at each step
157         rotation_noise = abs(np.random.normal(0, 0.2/200*m.pi)) # Rotation angle noise
158         step_length_scale = np.random.normal(1, 0.05) # Scale of each step
159         step_length_noise = abs(np.random.normal(0, 0.025)) # Random scale of each step
160         if(verbose):
161             print(f'[INFO][CONFIG] Rotation: {rotation_drift:.5f} ± {rotation_noise:.5f} rad,
Scale: {step_length_scale:.5f} ± {step_length_noise:.5f} x')
162         write_noise_information(step_length_scale, step_length_noise, rotation_drift,
rotation_noise, f'trajectory_{projectname}_{trajectory_index+1:05d}_applied-noise.log')
163
164         # Generating and exporting of the noised-trajectories.
165         for noise_index in range(training_data_length):
166             noised_trajectory = scale_trajectory(trajectory.copy(), step_length_scale,
step_length_noise)
167             noised_trajectory = rotate_trajectory(noised_trajectory, rotation_drift,
rotation_noise)
168             t.lines_export(noised_trajectory,
f'trajectory_{projectname}_{trajectory_index+1:05d}_training_{noise_index+1:05d}.csv')
169             del noised_trajectory

```