lib_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 math as m
# import string as st
# import random as r
# import re
import os
import numpy as np
# import platform
import matplotlib.pyplot as plt


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

verbose = True  # Shows more debugging information


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

def geradenschnitt(punkt1, punkt2, punkt3, punkt4):
    """
    Diese Funktion berechnet einen Geradenschnitt zwischen der Geraden von
    Punkt 1 zu Punkt 2 und der Geraden von Punkt 3 zu Punkt 4.
    Die Punkt werden jeweils in folgendem Format übergeben:
    punkt1 = {"x": FLOAT, "y": FLOAT}
    punkt2 = {"x": FLOAT, "y": FLOAT}
    punkt3 = {"x": FLOAT, "y": FLOAT}
    punkt4 = {"x": FLOAT, "y": FLOAT}
    """
    t12 = (punkt2["y"]-punkt1["y"])/(punkt2["x"]-punkt1["x"])
    t34 = (punkt4["y"]-punkt3["y"])/(punkt4["x"]-punkt3["x"])
    xs = punkt3["x"]
    xs += (((punkt3["y"]-punkt1["y"])-(punkt3["x"]-punkt1["x"])*t12)/(t12-t34))
    ys = punkt1["y"]+(xs-punkt1["x"])*t12
    punkt5 = {"y": ys, "x": xs}
    return(punkt5)


def lines_import(filename):
    if(verbose):
        print(f'[INFO] Importing file "{filename}"', end="\r")
    with open(os.path.join("data", filename)) as file:
        data = np.loadtxt(file, delimiter=";")
    lines = []
    for entry in data:
        lines.append(Line(entry[1], entry[2], entry[3], entry[4]))
    if(verbose):
        print(f'[INFO] Imported file "{filename}" successfully')
    return(lines)


def lines_export(lines, filename):
    """
    This functions takes Line-objects and writes them to a text-file.

    Args:
        lines ([Line]): A list with Line-objects
        filename (str): Filename the Lines will be written to
    """
    if(verbose):
        print(f'[INFO] Writing trajectory to "{filename}"')
    with open(os.path.join("data", filename), "w") as f:
        for index, line in enumerate(lines):
            if(index==0):
                f.write(f'{index}; {line.x1()}; {line.y1()}; {line.x2()}; {line.y2()}; {line.delta_x()}; {line.delta_y()}; {line.direction()}; 0.0; {line.length()}\n')
            else:
                f.write(f'{index}; {line.x1()}; {line.y1()}; {line.x2()}; {line.y2()}; {line.delta_x()}; {line.delta_y()}; {line.direction()}; {lines[index-1].direction()-line.direction()}; {line.length()}\n')


def plot_lines(lines, title="Line-segments", filename="plot"):
    if(verbose):
        pass
        # print(f'[INFO] Plotting lines')
    for i in lines:
        plt.plot([i.y1(), i.y2()], [i.x1(), i.x2()])
    # plt.legend(["lines"])
    plt.grid()
    plt.xlabel("Y")
    plt.ylabel("X")
    plt.title(title)
    plt.savefig(format="png", fname=os.path.join("plots", f'{filename}.png'))


def plot_lines_rgb(lines_red=None, lines_green=None, lines_blue=None, title="Line-segments", filename="plot"):
    if(verbose):
        pass
        # print(f'[INFO] Plotting lines (RGB)')
    if(lines_red != None):
        for i in lines_red:
            plt.plot([i.y1(), i.y2()], [i.x1(), i.x2()], color='red')
    if(lines_green != None):
        for i in lines_green:
            plt.plot([i.y1(), i.y2()], [i.x1(), i.x2()], color='green')
    if(lines_blue != None):
        for i in lines_blue:
            plt.plot([i.y1(), i.y2()], [i.x1(), i.x2()], color='blue')
    plt.grid()
    plt.xlabel("Y")
    plt.ylabel("X")
    plt.title(title)
    plt.savefig(format="png", fname=os.path.join("plots", f'{filename}.png'))


def plot_lines_rgb_show(lines_red=None, lines_green=None, lines_blue=None, title="Line-segments", filename="plot"):
    if(verbose):
        pass
        # print(f'[INFO] Plotting lines (RGB)')
    if(lines_red != None):
        for i in lines_red:
            plt.plot([i.y1(), i.y2()], [i.x1(), i.x2()], color='red')
    if(lines_green != None):
        for i in lines_green:
            plt.plot([i.y1(), i.y2()], [i.x1(), i.x2()], color='green')
    if(lines_blue != None):
        for i in lines_blue:
            plt.plot([i.y1(), i.y2()], [i.x1(), i.x2()], color='blue')
    plt.grid()
    plt.xlabel("Y")
    plt.ylabel("X")
    plt.title(title)
    plt.show()


def plot_graph(datasets, title_label, x_label, y_label, data_label, timestamps=None):
    """
    This function plots graphs.

    Args:
        datasets ([[float]]): A list with datasets a lists with floating-point

                              numbers
        title_label (str): This is the tile of the plot
        x_label (str): This is the label of the x-axis
        y_label (str): This is the label of the y-axis
        data_label ([str]): This is a list with labels of the datasets
        timestamps ([float], optional): By using a list of floating-point
                                        numbers the data get's plotted on a
                                        time-axis. If nothing is provided the
                                        values will be plotted equidistant.
    """
    for i, dataset in enumerate(datasets):
        if(timestamps==None):
            timestamps = range(len(dataset))
        plt.plot(timestamps, dataset)
    plt.legend(data_label)
    plt.grid()
    plt.xlabel(x_label)
    plt.ylabel(y_label)
    plt.title(title_label)
    plt.show()


def lines_to_points(lines):
    """
    This function converts a 2D trajectory of Line-objects into points.

    Args:
        lines ([Line]): Trajectory consisting of Line-objects inside a list
    """
    points = {"x": [], "y": []}
    points["x"].append(lines[0].x1())
    points["y"].append(lines[0].y1())
    for line in lines:
        points["x"].append(line.x2())
        points["y"].append(line.y2())
    return(points)


def points_to_lines(points):
    """
    This function converts a 2D trajectory of points into line-segments.

    Args:
        points ([[float, float]]): Trajectory consisting of points
                                   represented as list-entries with two
                                   float-values inside
    """
    previous_point = points.pop()
    lines = []
    for i, point in enumerate(points):
        if(verbose):
            print(f'[INFO][{i+1}/{len(points)}] Converting trajectory to line-segments', end="\r")
        lines.append(Line(previous_point[0], previous_point[1], point[0], point[1]))
        previous_point = point
    if(verbose):
        print("")
    return(lines)


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

class Line():
    def __init__(self, x1, y1, x2, y2):
        self.__x1 = float(x1)
        self.__y1 = float(y1)
        self.__x2 = float(x2)
        self.__y2 = float(y2)

    def x1(self):
        return(self.__x1)

    def y1(self):
        return(self.__y1)

    def x2(self):
        return(self.__x2)

    def y2(self):
        return(self.__y2)

    def set_x1(self, x1):
        self.__x1 = x1

    def set_y1(self, y1):
        self.__y1 = y1

    def set_x2(self, x2):
        self.__x2 = x2

    def set_y2(self, y2):
        self.__y2 = y2

    def delta_x(self):
        return(self.__x2-self.__x1)

    def delta_y(self):
        return(self.__y2-self.__y1)

    def direction(self):
        return(m.atan2(self.delta_y(), self.delta_x()))

    def length(self):
        return(m.sqrt((self.delta_x())**2+(self.delta_y())**2))


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

if __name__ == '__main__':
    pass
```