

## lib\_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 math as m
018 # import string as st
019 # import random as r
020 # import re
021 import os
022 import numpy as np
023 # import platform
024 import matplotlib.pyplot as plt
025
026
027 # -----
028 # Debugging-Settings
029
030 verbose = True # Shows more debugging information
031
032
033 # Functions
034 # -----
035
036 def geradenschnitt(punkt1, punkt2, punkt3, punkt4):
037     """
038     Diese Funktion berechnet einen Geradenschnitt zwischen der Geraden von
039     Punkt 1 zu Punkt 2 und der Geraden von Punkt 3 zu Punkt 4.
040     Die Punkt werden jeweils in folgendem Format übergeben:
041     punkt1 = {"x": FLOAT, "y": FLOAT}
042     punkt2 = {"x": FLOAT, "y": FLOAT}
043     punkt3 = {"x": FLOAT, "y": FLOAT}
044     punkt4 = {"x": FLOAT, "y": FLOAT}
045     """
046     t12 = (punkt2["y"]-punkt1["y"])/(punkt2["x"]-punkt1["x"])
047     t34 = (punkt4["y"]-punkt3["y"])/(punkt4["x"]-punkt3["x"])
048     xs = punkt3["x"]
049     xs += (((punkt3["y"]-punkt1["y"])-(punkt3["x"]-punkt1["x"])*t12)/(t12-t34))
050     ys = punkt1["y"]+(xs-punkt1["x"])*t12
051     punkt5 = {"y": ys, "x": xs}
052     return(punkt5)
053
054
055 def lines_import(filename):
056     if(verbose):
057         print(f'[INFO] Importing file "{filename}"', end="\r")
058     with open(os.path.join("data", filename)) as file:
059         data = np.loadtxt(file, delimiter=";")
060     lines = []
061     for entry in data:
062         lines.append(Line(entry[1], entry[2], entry[3], entry[4]))
063     if(verbose):
064         print(f'[INFO] Imported file "{filename}" successfully')
065     return(lines)
066
067
068 def lines_export(lines, filename):
069     """
070     This functions takes Line-objects and writes them to a text-file.
071
072     Args:
073         lines ([Line]): A list with Line-objects
074         filename (str): Filename the Lines will be written to
075     """
076     if(verbose):
077         print(f'[INFO] Writing trajectory to "{filename}"')
078     with open(os.path.join("data", filename), "w") as f:
079         for index, line in enumerate(lines):
080             if(index==0):
```

```

081         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')
082     else:
083         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')
084
085
086 def plot_lines(lines, title="Line-segments", filename="plot"):
087     if(verbose):
088         pass
089         # print(f'[INFO] Plotting lines')
090     for i in lines:
091         plt.plot([i.y1(), i.y2()], [i.x1(), i.x2()])
092     # plt.legend(["lines"])
093     plt.grid()
094     plt.xlabel("Y")
095     plt.ylabel("X")
096     plt.title(title)
097     plt.savefig(format="png", fname=os.path.join("plots", f'{filename}.png'))
098
099
100 def plot_lines_rgb(lines_red=None, lines_green=None, lines_blue=None, title="Line-segments",
    filename="plot"):
101     if(verbose):
102         pass
103         # print(f'[INFO] Plotting lines (RGB)')
104     if(lines_red != None):
105         for i in lines_red:
106             plt.plot([i.y1(), i.y2()], [i.x1(), i.x2()], color='red')
107     if(lines_green != None):
108         for i in lines_green:
109             plt.plot([i.y1(), i.y2()], [i.x1(), i.x2()], color='green')
110     if(lines_blue != None):
111         for i in lines_blue:
112             plt.plot([i.y1(), i.y2()], [i.x1(), i.x2()], color='blue')
113     plt.grid()
114     plt.xlabel("Y")
115     plt.ylabel("X")
116     plt.title(title)
117     plt.savefig(format="png", fname=os.path.join("plots", f'{filename}.png'))
118
119
120 def plot_lines_rgb_show(lines_red=None, lines_green=None, lines_blue=None, title="Line-segments",
    filename="plot"):
121     if(verbose):
122         pass
123         # print(f'[INFO] Plotting lines (RGB)')
124     if(lines_red != None):
125         for i in lines_red:
126             plt.plot([i.y1(), i.y2()], [i.x1(), i.x2()], color='red')
127     if(lines_green != None):
128         for i in lines_green:
129             plt.plot([i.y1(), i.y2()], [i.x1(), i.x2()], color='green')
130     if(lines_blue != None):
131         for i in lines_blue:
132             plt.plot([i.y1(), i.y2()], [i.x1(), i.x2()], color='blue')
133     plt.grid()
134     plt.xlabel("Y")
135     plt.ylabel("X")
136     plt.title(title)
137     plt.show()
138
139
140 def plot_graph(datasets, title_label, x_label, y_label, data_label, timestamps=None):
141     """
142     This function plots graphs.
143
144     Args:
145         datasets ([[float]]): A list with datasets a lists with floating-point
146                                numbers
147         title_label (str): This is the tile of the plot
148         x_label (str): This is the label of the x-axis
149         y_label (str): This is the label of the y-axis
150         data_label ([str]): This is a list with labels of the datasets
151         timestamps ([float], optional): By using a list of floating-point
152                                         numbers the data get's plotted on a
153                                         time-axis. If nothing is provided the
154                                         values will be plotted equidistant.
155     """
156
157     for i, dataset in enumerate(datasets):
158         if(timestamps==None):

```

```

159         timestamps = range(len(dataset))
160         plt.plot(timestamps, dataset)
161         plt.legend(data_label)
162         plt.grid()
163         plt.xlabel(x_label)
164         plt.ylabel(y_label)
165         plt.title(title_label)
166         plt.show()
167
168
169 def lines_to_points(lines):
170     """
171     This function converts a 2D trajectory of Line-objects into points.
172
173     Args:
174         lines ([Line]): Trajectory consisting of Line-objects inside a list
175     """
176     points = {"x": [], "y": []}
177     points["x"].append(lines[0].x1())
178     points["y"].append(lines[0].y1())
179     for line in lines:
180         points["x"].append(line.x2())
181         points["y"].append(line.y2())
182     return(points)
183
184
185 def points_to_lines(points):
186     """
187     This function converts a 2D trajectory of points into line-segments.
188
189     Args:
190         points ([[float, float]]): Trajectory consisting of points
191                                     represented as list-entries with two
192                                     float-values inside
193     """
194     previous_point = points.pop()
195     lines = []
196     for i, point in enumerate(points):
197         if(verbose):
198             print(f'[INFO][{i+1}/{len(points)}] Converting trajectory to line-segments', end="\r")
199         lines.append(Line(previous_point[0], previous_point[1], point[0], point[1]))
200         previous_point = point
201     if(verbose):
202         print("")
203     return(lines)
204
205
206 # Classes
207 # -----
208
209 class Line():
210     def __init__(self, x1, y1, x2, y2):
211         self.__x1 = float(x1)
212         self.__y1 = float(y1)
213         self.__x2 = float(x2)
214         self.__y2 = float(y2)
215
216     def x1(self):
217         return(self.__x1)
218
219     def y1(self):
220         return(self.__y1)
221
222     def x2(self):
223         return(self.__x2)
224
225     def y2(self):
226         return(self.__y2)
227
228     def set_x1(self, x1):
229         self.__x1 = x1
230
231     def set_y1(self, y1):
232         self.__y1 = y1
233
234     def set_x2(self, x2):
235         self.__x2 = x2
236
237     def set_y2(self, y2):
238         self.__y2 = y2
239
240     def delta_x(self):
241         return(self.__x2-self.__x1)

```

```

242
243     def delta_y(self):
244         return(self.__y2-self.__y1)
245
246     def direction(self):
247         return(m.atan2(self.delta_y(), self.delta_x()))
248
249     def length(self):
250         return(m.sqrt((self.delta_x())**2+(self.delta_y())**2))
251
252
253 # Beginning of the Programm
254 # -----
255
256 if __name__ == '__main__':
257     pass

```