Exercise 3: formation control

```
# header to start
%matplotlib notebook

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import matplotlib as mp

import pickle
import IPython

import scipy.linalg
```

We wish the control the formation of 4 robots randomly distributed in the environment to keep the formation shown in the figure of Exercise 2.

Question 1

Assume each agent has state space dynamics $\dot{\mathbf{p}}_i = \mathbf{u}_i$, with $\mathbf{u}_i$ in $\mathbb{R}^2$ and $\mathbf{u} = [\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3, \mathbf{u}_4]$.

Implement the second order linear control law seen in the class

$$\mathbf{u} = -k\mathbf{L}\mathbf{x} + k\mathbf{D}\mathbf{z}_{ref}$$

where $k > 0$ is a positive gain and $\mathbf{D}$ is the incidence matrix of the graph.

Simulate the control law for several random initial conditions of the agents (in 2D). What do you observe? How does it compare to the same control law but for a framework with a complete graph?

Question 2

Assume each agent has state space dynamics $\dot{\mathbf{p}}_i = \mathbf{u}_i$, with $\mathbf{u}_i$ in $\mathbb{R}^2$ and $\mathbf{u} = [\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3, \mathbf{u}_4]$.

We now consider the following control law

$$\mathbf{u} = \mathbf{R}_{\mathcal{G}}^T(\mathbf{p})(\mathbf{g}_d - \mathbf{g}_{\mathcal{G}}(\mathbf{p}))$$

where $\mathbf{R}_{\mathcal{G}}$ is the rigidity matrix associated to the graph of the framework, $\mathbf{g}_d$ is the vector of desired square distance between agents and $\mathbf{g}_{\mathcal{G}}$ is the measured square distance between each agent.

Simulate the control law for several random initial conditions of the agents (in 2D). What do you observe? How does it compare to the same control law but for a framework with a complete graph?

Question 3

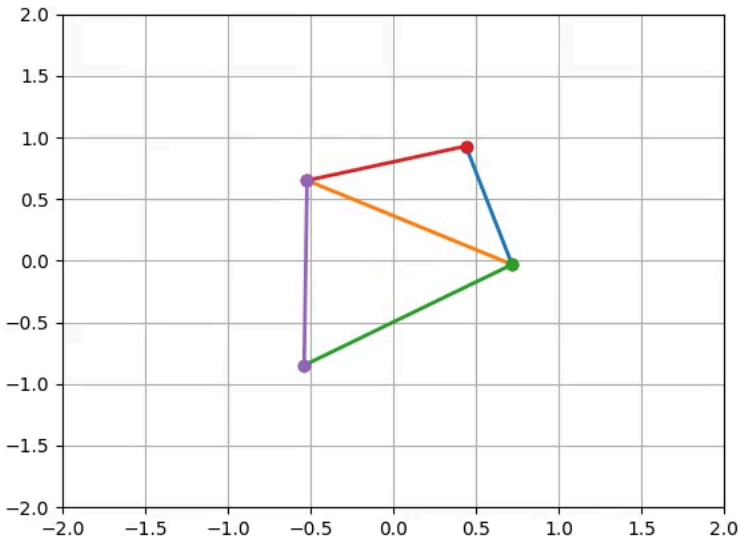
How would you compare both control laws? What are the pros and cons of each of them?

Helper function to display results

This function can be used to display the behavior of the robots in 2D

```
def make_animation(plotx,E,xl=(-2,2),yl=(-2,2),inter=25, display=False):
    '''
    takes a graph and motion of vertexes in 2D and returns an animation
    E: list of edges (each edge is a pair of vertexes)
    plotx: a matrix of states ordered as (x1, y1, x2, y2, ..., xn, yn) in the rows and time in columns
    xl and yl define the display boundaries of the graph
    inter is the interval between each point in ms
    '''
    fig = mp.figure.Figure()
    mp.backends.backend_agg.FigureCanvasAgg(fig)
    ax = fig.add_subplot(111, autoscale_on=False, xlim=xl, ylim=yl)
    ax.grid()

    list_of_lines = []
    for i in E: #add as many lines as there are edges
        line, = ax.plot([], [], 'o-', lw=2)
        list_of_lines.append(line)

    def animate(i):
        for e in range(len(E)):
            vx1 = plotx[2*E[e][0],i]
            vy1 = plotx[2*E[e][0]+1,i]
            vx2 = plotx[2*E[e][1],i]
            vy2 = plotx[2*E[e][1]+1,i]
            list_of_lines[e].set_data([vx1,vx2],[vy1,vy2])
        return list_of_lines

    def init():
        return animate(0)


    ani = animation.FuncAnimation(fig, animate, np.arange(0, len(plotx[0,:])),
        interval=inter, blit=True, init_func=init)
    plt.close(fig)
    plt.close(ani._fig)
    if(display==True):
        IPython.display.display_html(IPython.core.display.HTML(ani.to_html5_video()))
    return ani


# for example assume that you have simulated a formation control in 2D and stored the data in a file
# we load the data needed for the display
with open('example_animation.pickle', 'rb') as f:
    data = pickle.load(f)

# this is the list of edges (as we usually define them for an undirected graph)
E = data['E']
print('the list of edges is:')
print(E)

# this is the time of simulation
t = data['t']

# this is an array containing the evolution of the states of the robot
# x[0,:] contains the time evolution of the x variable of robot 1
# x[1,:] contains the time evolution of the y variable of robot 1
# x[2,:] contains the time evolution of the x variable of robot 2
# etc
x = data['x']

# since we simulated with a small delta t = 0.001, we want to subsample for display
# we just take data every 50ms
plotx = x[:,::50]

make_animation(plotx, E, inter=50, display=True)

# a video showing the behavior of the robots and the connection between the robots should be displayed below
```

```
the list of edges is:
[[0, 1], [0, 2], [0, 3], [1, 2], [2, 3]]
```


2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0