ME-GY 7943
Network Robotic Systems, Cooperative Control and Swarming

Exercise series 3

Exercise 1

In this exercise, the goal is to simulate in Python the control of a formation with 5 robots using the rigidity based control law

$$\dot{\mathbf{p}} = \mathbf{R}_{\mathcal{G}}^{T}(\mathbf{p})(\mathbf{g}_d - \mathbf{g}_{\mathcal{G}}(\mathbf{p})) \tag{1}$$

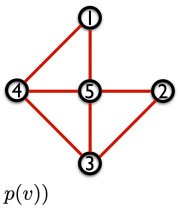
where $\mathbf{p}_i$ is the position vector (x and y) of robot i, $\mathbf{p}$ is the vector of stacked positions of all robots, i.e. $\mathbf{p} = \begin{bmatrix} \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_5 \end{bmatrix}$, $\mathbf{g}_d$ is the vector of desired constraints (defined as squared distances) between neighbors (i.e. it is a constant vector), $\mathbf{g}_{\mathcal{G}}$ is the vector of current squared distance measurements between robots and $\mathbf{R}_{\mathcal{G}}$ is the rigidity matrix evaluated at the current position of the robots. We would like the robots to have the formation defined by the framework shown in Figure 1.

a) Assuming that each edge of the graph represents a constraint, write the vector of desired constraints $\mathbf{g}_d$ for the particular framework shown in Figure 1.

b) Write a Python function that computes $\mathbf{g}_{\mathcal{G}}(p)$ (i.e. a function that takes as argument the vector of position $\mathbf{p}$ and returns a vector of squared distances).

c) Write a Python function that computes the rigidity matrix of the graph as a function of the position of the robots (i.e. which takes as argument the vector of position $\mathbf{p}$ and returns the rigidity matrix as an array).

d) Using the previous functions, simulate (using matrix notations) the formation controller defined in Equation 1, taking as initial condition $\mathbf{p} = [0, 1, 1, 0, 2, 2, 3, 2, 4, 0]$, until the desired formation is reached. Plot the x and y positions of all the robots as a function of time. You may also use the function *make_animation_obstacles* provided in the companion Jupyter file to display an animation of the behavior (use an empty list for the obstacles).



Framework $(\mathcal{G}, p(v))$

$\mathcal{G} = (\mathcal{V}, \mathcal{E})$

$\mathcal{V} = \{v_1, v_2, v_3, v_4, v_5\}$

$\mathcal{E} = \{(v_1, v_4), (v_1, v_5), (v_2, v_3), (v_2, v_5), (v_3, v_5), (v_4, v_5), (v_4, v_3)\}$

$p(v) = \{v_1 \to (0, 2),\ v_2 \to (1, 1),\ v_3 \to (0, 0),$
$v_4 \to (-1, 1),\ v_5 \to (0, 1)\}$

Figure 1: Desired formation of the 5 robots and framework definition.

Exercise 2

We would like to write a controller for 4 robots that need to keep a square formation while navigating obstacles towards a desired goal. We will use the following control law for each robot

$$\begin{bmatrix} \ddot{x}_i \\ \ddot{y}_i \end{bmatrix} = F_{formation,i} + F_{target,i} + F_{obstacle,i} \tag{2}$$

which is composed of three distinct terms: i) a formation control law, ii) a control law to move towards the target and iii) an obstacle avoidance control law.

i) The formation control law is a linear formation control as seen in the class

$$F_{formation,i} = \begin{bmatrix} K_F(Dz_{x,desired} - Lx)_i + D_F(D\dot{z}_{x,desired} - L\dot{x})_i \\ K_F(Ddz_{y,desired} - Ly)_i + D_F(D\dot{z}_{y,desired} - L\dot{y})_i \end{bmatrix} \tag{3}$$

where K_F and D_F and positive gains, L is the graph Laplacian, D is the incidence matrix and $z_{desired}$ is the desired set of desired direction vectors associated to each edges. Here we wrote the control law in matrix form where $x = [x_1, x_2, \cdots, x_n]^T$ and $y = [x_1, x_2, \cdots, x_n]^T$ and it is understood that we take the ith line of the results to apply to robot i.

ii) The part controlling the desired motion towards the target is a proportional derivative control law to move the formation towards the target

$$F_{target,i} = \begin{bmatrix} K_T * min(x_{goal} - x_i, 1.) - D_T\dot{x}_i \\ K_T * min(y_{goal} - y_i, 1.) - D_T\dot{y}_i \end{bmatrix} \tag{4}$$

where K_T and D_T are positive gains, $[x_{goal}, y_{goal}]$ is the position of the goal and we limit the error to the distance to the goal in magnitude to be lower than 1.0 (here we assume min is the min in absolute value terms). This ensures that the maximum acceleration does not grow when the goal is further.

iii) Finally the obstacle avoidance control law is derived from a potential similar to what we saw in class. We use the potential function

$$V_O = \begin{cases} (\ln d + d_{max}/d) & \text{if } d < d_{max} \\ (\ln d_{max} + d_{max}/d_{max}) & \text{otherwise} \end{cases} \tag{5}$$

where $d = \sqrt{(x_i - x_O)^2 + (y_i - y_O)^2}$ is the distance of robot i to the obstacle (where x_O and y_O are the coordinates of the obstacle) and is the d_{max} distance on which the potential is active. From the potential function, we can derive the force applied on robot i due to obstacle j by following the negative gradient of the potential function. So each obstacle j contributes on robot i

$$F_{O_j,i} = \begin{cases} -K_O \frac{\partial}{\partial x_i, y_i} V_O = -K_O(\frac{d_{O_j,i} - d_{max}}{d^3_{O_j,i}}) \begin{bmatrix} x_i - x_{O,j} \\ y_i - y_{O,j} \end{bmatrix} & \text{if } d < d_{max} \\ 0 \quad \text{otherwise} \end{cases} \tag{6}$$

and where $d_{O_j,i} = \sqrt{(x_i - x_O)^2 + (y_i - y_O)^2}$ is the distance of robot i to obstacle j. The resulting part of the controller for obstacle avoidance on robot i is

$$F_{obstacle,i} = \sum_{j \in \text{obstacles}} F_{O_j,i} \tag{7}$$

where K_O is a positive gain to be chosen.

Using this control law, simulate 4 robots maintaining a square formation which start with initial positions $[0, 0, -0.2, -0.2, 0.2, 0, 0, 0.1]$ and 0 initial velocity and tries to reach a goal located at $[10, 10]$.

a) Simulate the behavior of the robots when there are no obstacles. Plot the motion of each robot as a function of time. You may also generate a video of the formation using the function *make_animation_obstacles*

b) Same question when there are 8 point obstacles located at locations [4,0], [4,4.7], [4,8], [6,2], [7,6], [6,10], [8,4], and [8,12] respectively.

c) Add a ninth obstacle at location [8.1,5.6]. What happens in this case? Is the formation able to reach the target?

For all the experiments you may use $K_F = 10$, $K_T = 5$,$K_O = 10$, $D_F = 2\sqrt{K_F}$, $D_T = 2\sqrt{K_T}$, $d_{max} = 1.0$. Use an integration step of $dt = 0.001$. When creating the video, you may not want to use all the points generated by the simulation but plot only every 100 points (i.e. a point every 100 milliseconds) in order to make computation faster for the video rendering.