TSFS12 HAND-IN EXERCISE 4
Collaborative Control

September 11, 2023

1 OBJECTIVE

The objective of this hand-in exercise is to explore and implement the methods described in Lecture 8:

- Position-based formation control
- Displacement-based formation control
- Distance-based formation control

The multi-agent formation control problem will in each case be solved for a set of agents with a desired formation, and the result is a model in the form

$$\begin{aligned} \dot{x}_i &= f_i(t, x_i, u_i) \\ y_i &= h_i(x) \\ u_i &= g_i(y_i) \end{aligned}$$

$i = 1, \ldots, N$, where N is the number of agents, x_i is the state vector of the agent, y_i is the measurements available to the agent, u_i is a control signal, and

$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_N \end{pmatrix}$$

is the state vector of the complete system.

2 PREPARATIONS BEFORE THE EXERCISE

Before doing this exercise, make sure you have read and understood the material covered in

- Lecture notes on Collaborative control (Lecture 8). The notation and terminology from these lecture notes will be used in this document.
- Oh, Kwang-Kyu, et al. "*A survey of multi-agent formation control.* Automatica 53 (2015) 424–440.
- This exercise description.

To get the background code for the exercise, get the latest files from `https://gitlab.liu.se/vehsys/tsfs12` and familiarize yourself with the provided code.

You are free to choose in which language to implement these exercises. Code skeletons are available in Matlab and Python, but if you prefer to make your own implementations that is also possible. This document is written with reference to the

Matlab files, but pointers towards corresponding Python equivalents are provided in the text. In the student labs, all Python packages needed are pre-installed in the virtual environment activated by

```
source /courses/tsfs12/env/bin/activate
```

3 REQUIREMENTS

To complete this exercise you should implement the solutions to the problems described in Section 5. The exercise is examined by submitting the following on the Lisam course page.

1. Runnable code. If your code consist of several files, submit a zip-archive.
2. A short document, that does *not* have to be formatted as a self contained report:
   - answers to the questions, including relevant plots, in Section 5, and other questions that you have encountered during solving this exercise.
   - a concluding discussion.

   Submit the document in PDF format.

It is allowed to discuss the exercises on a general level with other course participants. However, code sharing outside groups is not allowed. Moreover, both students in the group should be fully involved in performing all exercises, and thereby be prepared to answer questions on all subtasks.

See Section 6 for the extra assignment needed for higher grades. The extra assignment is done individually and is submitted by a document answering the questions, including suitable figures, plots, and tables. The document is to be submitted in PDF format. The code should also be submitted as a zip-archive. There is only pass/fail on the extra exercise and the document can not be revised or extended after first submission. It is not required to get everything correct to pass the assignment, we assess the whole submission. You are of course welcome to ask us if you have questions or want us to clarify the exercise.

Since these exercises examines this course, we would like to ask you not to distribute or make your solutions publicly available, e.g., by putting them on github. A private repository on gitlab@liu is of course fine.

Thanks for your assistance!

4 CONTROLLER IMPLEMENTATION ENVIRONMENT

In the provided simulation example, `ex0.m` (Matlab), `ex0.ipynb` (Python), the model $\dot{p}_i = u_i$ is implemented with the measurement $y_i = p_i$, and the control law $u_i = k_p(p_i^* - p_i)$. In the `matlab` version, the function `ode45` is used to simulate the system:

```
[t,x] = ode45(@(t,x) multi_agent_ode(t,x,agent,n), tspan, x0);
```

The corresponding command in Python is

```
x = odeint(lambda x, t: multi_agent_ode(x, t, agent, n), x0, t)
```

where `odeint` is a function in Python module `scipy.integrate`.

In this example, `x` is the state vector of the system containing the states of the agents, in this case the positions p_i , in two-dimensions, of 4 agents $i = 1, 2, 3, 4$. The function `multi_agent_ode` computes the value of the time derivative of `x` with input arguments: time `t`, states `x`, the structure `agent`, and the dimension `n`, which is equal to 2 in this case. In Python, the `multi_agent_ode` function is defined in the provided file `collab_functions.py`.

In Matlab the structure `agent` contains all information about the dynamic model of the agent, available sensors, control laws, and control objectives. The objective of this assignment is to adapt the structure `agent` to the different cases described in the exercises in Section 5. For each agent $i = 1, \ldots, N$, the fields in the structure `agent` are:

- `agent(i).f` is an implementation of the dynamic equations $\dot{x}_i = f_i(t, x_i, u_i)$ with input arguments: time `t`, the state of the agent `x`, the input signal `u`, and the structure `agent(i).mdlpar`, which contains different parameters in the dynamic model.
- `agent(i).h` is an implementation of the measurement function $y_i = h(x)$ with input arguments: the state vector of the complete system and the structure `agent(i).measpar`, which contains parameters in the measurement equations. In this example the indices in the states in the state vector measured by agent i.
- `agent(i).g` is an implementation of the control law $u_i = g(y_i)$ with input arguments: the measurement `y`, the structure `agent(i).ctrlpar`, which contains parameters in the control law. Furthermore, `agent(i).xref` contains information about the control objectives, in this a desired position of agent i.

In Python, the correspoding data structure is a list of agents where each agent is represented by a dictionary, similar to the matlab structure above, that gathers all information about a single agent.

To help the creation of such a dictionary for an agent, a helper function `CreateAgent` is provided in the file `collab_functions.py`

```
CreateAgent(f, mdlpar, g, ctrlpar , h, measpar, xref):
    """Create an agent dictionary

    Returns a dictionary collecitng all information needed to
    integrate the behavior of an agent.

    Inputs:
        f - Dynamic function for the agent: f(t, x, u, mdlpar)
        mdlpar - Dictionary with model parameters used by f
        g - Control function: g(y, xref, ctrlpar)
        ctrlpar - Dictionary with control parameters used by g
        h - Measurement function: h(x, measpar)
        measpar - Dictionary with parameters used by h
        xref - Reference specification
```

In `ex0.ipynb`, an agent is created using a command similar to

```
  CreateAgent(f1, modelparam, g1, controller_par, h1,
              formation_measurement,
              agent_position_reference)
```

where `f1` is the dynamic model for the agent, `modelparam` the model parameters dictionary, `g1` the controller function, `controller_par` the controller parameter dictionary, `h1` the measurement function. The arguments, `formation_measurement` a representation of agent connections and `agent_position_reference` which represents references for the controllers are both described further in the next section.

5 ASSIGNMENTS

This section summarizes a number of discussion topics and questions you should implement, investigate, and reflect upon to pass this exercise. Hand in one separate file for each assignment, and a brief report discussing your results.

5.1 Position-based formation control

Exercise 5.1. The directory `functions` contains the functions `f1`, `g1`, and `h1` that are used in the definition of the structure `agent`. In Python, all functions are defined directly in the notebook.

Firstly, you need to create new functions `f2`, `g2`, and `h2`, where the single-integrator model is replaced with a double-integrator

$$\ddot{p}_i = u_i,$$

the control law

$$u_i = k_v(v_i^* - v_i) + k_p(p_i^* - p_i)$$

is used, and the position p_i and velocity v_i are measured.

Then, add a constant force, caused by a wind, in the x-direction to the model, and simulate the model (hint: the force can be regarded as the extra control input). Plot the trajectories of the agents with and without wind force.

5.2 Displacement-based formation control

Exercise 5.2. In this exercise, there are 8 agents modelled with a double integrator $\ddot{p}_i = u_i$. The desired formation is as vertical string with relative distance one and relative velocity zero. There are two types of agents in the system. The agent at the top of the string that can measure its absolute position and the objective is to reach the point $(0, 4)$. The other agents can measure the relative displacement to their two nearest neighbours, and the objective for them is to keep the same relative displacement as the desired string formation.

5.2.1 *Matlab*

In the file `Exercise5_2_Part1.m` (or in `Exercise5_2_Part2.m`) you can find an example how to represent the information described above. First the cell array `idx` is defined

```
idx = {1:4, 5:8, 9:12, 13:16, 17:20, 21:24, 25:28, 29:32};
```

where `idx{i}` are the indices of the states of agent i in the state vector.

Agent 1, the one at the top of the string, measures its own 4 states, position and velocity, and this is represented as

```
agent(1).measpar.idx = idx{1};
```

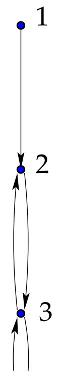
and that the goal state is `[0 4 0 0]'` is represented as

```
agent(1).xref = @(t) [0 4 0 0]';
```

Agent 2 measures the relative positions to agents 1 and 3, i.e., there one edge from node 1 to node 2, and one edge from node 3 to node 2 in the directed graph. This is represented in the following way

```
agent(2).measpar.idx = cat(3,[idx{2};idx{1}],[ idx{2};idx{3}]);
```

For each j, in this case $j = 1, 2$, the two rows in `agent(2).measpar.idx(:,:,j)`, are the indices in the state vector `x` of two nodes of an edge. The relative displacements and velocities are given by the difference between the states of the two nodes.



The two columns in `agent(2).xref` are the desired relative displacement and relative velocity:

```
agent(2).xref = @(t) [0 0;1 -1; 0 0;0 0];
```

i.e., the columns are

$$\begin{pmatrix} p_j^* - p_i^* \\ v_j^* - v_i^* \end{pmatrix}$$

where $i = 2$ is the index of the agent and $j = 1, 3$ are indices of the neighbors. The corresponding fields for the other agents are defined analogously.

5.2.2 Python

For Python, the corresponding information is defined in file `Exercise5_2_Part1.ipynb` (or `Exercise5_2_Part2.ipynb`). For example, the reference positions, as described above is defined for each of the 8 agents as

```
formation_references = [lambda t: np.array([0, 4, 0, 0]).reshape(4, -1),
                        lambda t: np.array([[0, 0], [1, -1], [0, 0], [0, 0]]),
                        lambda t: np.array([[0, 0], [1, -1], [0, 0], [0, 0]]),
                        lambda t: np.array([[0, 0], [1, -1], [0, 0], [0, 0]]),
                        lambda t: np.array([[0, 0], [1, -1], [0, 0], [0, 0]]),
                        lambda t: np.array([[0, 0], [1, -1], [0, 0], [0, 0]]),
                        lambda t: np.array([[0, 0], [1, -1], [0, 0], [0, 0]]),
                        lambda t: np.array([0, 1, 0, 0]).reshape(4, -1)]
```

where the first item in the list is for the first agent and so on.

In files `Exercise5_2_Part1.ipynb` and `Exercise5_2_Part2.ipynb`, `formation_measurement_index` and `formation_references` should be used in the same way as `simple_formation_measurement_index` and `simple_formation_references` used in `ex0.ipynb`.

5.2.3 Exercise

Note that in this exercise you have to create two pairs of measurement and control functions, since there are two different kinds of agents. You need to create these four functions and assign them to the correct agents.

It is assumed that all positions p_i and velocities v_i are measured by the first agent and the control law

$$u_i = k_v(v_i^* - v_i) + k_p(p_i^* - p_i)$$

is used for this agent. The control law

$$u_i = k_p \sum_{j\in\mathcal{N}_i} (p_j - p_i - p_j^* + p_i^*) + k_v \sum_{j\in\mathcal{N}_i} (v_j - v_i - v_j^* + v_i^*),$$

is used for the other agents, where $p_j - p_i$ and $v_j - v_i$ are measured. $\mathcal{N}_i$ is the set of neighbours to agent i.

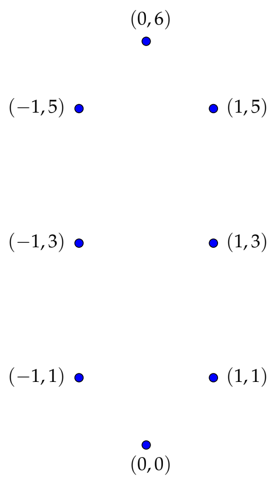
(1). In `Exercise5_2_Part1.m` (or `Exercise5_2_Part1.ipynb`), simulate the system $k_p = k_v = 2$, and with initial position $(i, 0)$ and initial velocity equal to zero for agent i, $i = 1, \ldots, 8$.

(2). In `Exercise5_2_Part2.m` (or `Exercise5_2_Part2.ipynb`), remove all edges from node $i - 1$ to node i, $i = 3, \ldots, 8$, i.e., all measurements of the relative state of the agent below are removed. The desired position of agent 1 was previously fixed to $(0, 4)$. Add a harmonic oscillation in the y-direction with amplitude 0.2 to this position and simulate the system with angular frequencies $1\ rad/s$ and $2\ rad/s$. Choose the simulation time large enough so that the formation is located on a vertical line at the end of the simulation. Plot the vertical velocities for the agents for the two frequencies. Describe how the behavior of the agents differs between the two cases.

5.3 Distance-based formation control

Exercise 5.3.

The desired formation in this case is given by the figure below



The first step is to construct a minimally rigid graph for the nodes. Note that a single-integrator model

$$\dot{p}_i^i = u_i^i, \quad i = 1, \ldots, N$$

will be used in this exercise, where p_i^i and u_i^i denote the position and control signal with respect to the local coordinate system, as described in Lecture 8. It assumed that the agent at the top can measure its absolute position, and it will use the control law

$$u_i = k_p(p_i^* - p_i)$$

The control signal for the other agents is computed using a gradient control law:

$$u_i^i = -\nabla_{p_i^i}\phi = k_p \sum_{j\in\mathcal{N}_j} \gamma'_{ij}(\|p_j^i - p_i^i\|)\frac{p_j^i - p_i^i}{\|p_j^i - p_i^i\|}$$

The local potential function is defined by

$$\phi_i = \frac{k_p}{2}\sum_{j\in\mathcal{N}_i} \gamma_{ij}(\|p_j^i - p_i^i\|)$$

with γ_{ij} defined as

$$\gamma_{ij}(\|p_j - p_i\|) = (\|p_j - p_i\|^2 - \|p_j^* - p_i^*\|^2)^2$$

(1). In `Exercise5_3_Part1.m` (or `Exercise5_3_Part1.ipynb`), assume that the desired position of the agent at the top is moving on a circle of radius 2. Simulate the model in this case with $k_p = 1$ and $k_p = 5$, and plot the distance between the top node and one of it neighbors. Explain the difference.

(2). In `Exercise5_3_Part2.m` (or `Exercise5_3_Part2.ipynb`), it is assumed that the agent at the bottom also can measure absolute position. Use this additional information to control the agents so the orientation of the formation is preserved.

6 EXTRA ASSIGNMENT

The extra assignment has two parts: Exercise 6.1 and 6.2.

6.1 Measurement errors

Exercise 6.1. Replace the control law for agent at the top in Exercise 5.3 with the same gradient control law as the other agents. The agent at position $(1, 3)$ has an error in the measurement of the distance to the agent at position $(1, 5)$ and the measurement is 20% larger that the correct value. Add this measurement error to the model an simulate the system.

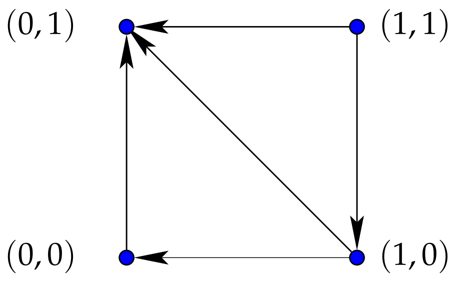
a) Describe the consequences of the measurement error and explain why the formation behaves the way it does.

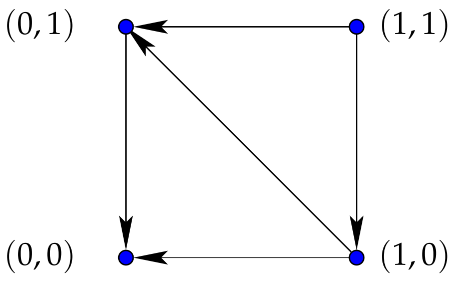
b) Give an example where to measurement errors make the formation move in the y-direction.

c) Give an example where two measurement errors make the agents rotate around the center of the formation.

6.2 Distance based directed formation control

Exercise 6.2. In Exercise 5.3 and 6.1, distance based formation control was used with undirected graphs that specified the set of neighbours for each agent. In this exercise we shall use distance based formation control with directed graphs. Consider the following two directed graph with vertices at the corners of the unit square:





As in Exercise 5.2 this will define the desired formation and the direction of the arrows defines which distance measures that are available for each agent. For example, the agent located at position $(0, 1)$ in the graph to the left can measure the distance to the other three agents.

Use the same gradient control law as before for the four agents and simulate the system with measurements available as represented by one of the two graphs at a time. Try some different initial conditions and describe the formation the agents converge to. Explain why the results are different for the two graphs.