6 Application: Synchronization of DC Motors

The goal of this lab is to achieve output synchronization in a five agents network. Each agent is modelled as a SISO linear system with undamped oscillatory behavior, described by the state-space equation below.

$$\dot{x}_i(t) = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} x_i(t) + \begin{pmatrix} 0 \\ 1 \end{pmatrix} u_i(t), \quad x_i(0) = x_{0,i} \in \mathbb{R}^2, i \in \{1, \dots, 5\}$$
$$y_i(t) = \begin{pmatrix} 0 & 1 \end{pmatrix} x_i(t)$$

The initial conditions of each agent are:

$$x_{0,1} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}, \; x_{0,2} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}, \; x_{0,3} = \begin{pmatrix} 1 \\ 3 \end{pmatrix}, \; x_{0,4} = \begin{pmatrix} 2 \\ 4 \end{pmatrix} \text{ and } x_{0,5} = \begin{pmatrix} 2 \\ 5 \end{pmatrix}$$

6.1 Problem Statement

Before the lab session:

a) Design a controller that asymptotically synchronizes the output of the five homogeneous agents given above. You may choose any undirected communication graph, in which each agent has one to four neighbors.

b) Prove that your controller is able to achieve output synchronization.

c) Simulate the multi-agent network on your PC and plot the closed-loop input and output trajectories.

d) Any real implementation of the controller will rely on measurements or estimates of the states. These data points will only be given at discrete time instances. Adjust your simulation so that the state vector is sampled with a sample interval $\Delta t = 20\,\text{ms}$ before it is passed as input to the controller. Simulate the closed loop system and compare the input and output trajectories to the purely continous-time simulation. *Hint: you may assume that this sample interval is small enough such that you do not have to adjust the controller parameters.*

e) Familiarize yourself with the rest of the lab description and in particular with the `main.cpp` from Moodle.

During the lab session:

a) Implement your networked controller on Raspbery Pis. Implement the given initial conditions from above in agentModel_data.cpp.

b) Simulate the multi-agent network on the Raspberry Pis and store the output logs on your PC.

c) Compare the simulation from the Raspberry Pi network to the one from your PC.

6.2 Implementation

For the implementation of the networked system, you will utilize 5 Raspberry Pi's, each representing one of the agents with its own control action.

6.2.1 Accessing the Raspberry Pis

Retina pool: you can access the Raspberry Pis via an ssh-connection using Putty from the Retina pool at the ET/IT department of TU Dortmund. Note that it is also possible to access the Retina pool remotely from your private PC, which allows you to participate in the lab session even from home. Any student with a major in ET/IT or A&R should have access to the Retina pool via their TU-ID: `http://www.retina.e-technik.tu-dortmund.de/cms/de/Hilfe_Infos/FAQ/index.html#wie_zugriff_RTmobil_extern`.

Please write to goesta.stomberg@tu-dortmund.de, if no member of your group can access the Retina pool. This applies to students from other universities than TU Dortmund, for instance.

ssh credentials: One member per group should email to goesta.stomberg@tu-dortmund.de to ask for the ssh credentials of the Raspberrry Pis.

6.2.2 Code Overview

The controller and dynamical system are implemented in C++. All necessary files are located in the agent.zip archive on Moodle. Please download the archive and transfer it via Putty to the Raspberry Pis. Place the content inside the folder '/home/pi/agent/'. You may have to create the agent directory under /home/pi first. The archive contains

- `main.cpp`: incomplete main file of an agent; you will edit this file
- `agentModel.*`: exported Simulink model of the SISO plant. The simulink model contains a discretized model with a sample interval of 20 ms and a zero-order-hold parameterization of the control input.
- `agentModel_data.cpp`: contains the discretized state-space system as well as the initial condition in line 45
- and further files that you may go through at your own discretion

The file 'main.cpp' is where you will implement your controller. To assist with the implementation, we provide a code template on Moodle. Each Raspberry Pi must contain exactly one main.cpp. In addition to the main.cpp file, you will also have to edit the agentModel_data.cpp to adjust the initial conditions of the agents.
The main file is composed of two concurrent threads. The first thread continuously simulates the system dynamics and the controller. The second thread exclusively listens to the communication sockets and updates the values received from the neighbors.
The first thread is where you will add your control law, before the function that calculates the next output value of the plant is called ('agentModel_step(1)', on line 135).

```
for(int i = 0; i < count; i++)
{
    //start clock
    startTime = clock();

    //Yor control output calculations will go here:
    //u =(...)
    agentModel_U.Input = u;
    agentModel_step(1);

    (...)
}
```

For the calculation of the control variable, you will need the current value of the neighbors' and the own agent's outputs. The agent's own output value is stored in the variable 'agentModel_Y.Output'. The

neighbor's (abbreviated as N) outputs are saved inside the receiver class object 'received' and can be accessed on the main thread through: received.N1, received.N2 and received.N3 and so on.

```
//Example of usage of variables
u = 5*received.N1 - 2*received.N2 + 7*agentModel_Y.Output;
```

6.2.3 How to run the program

Before running the program it is necessary to compile the code on each of the Raspberry Pis with the command 'sudo make':

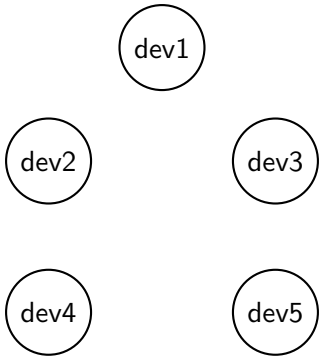
1. cd /home/pi/agent/

2. sudo make

After compilation you'll need to run the program passing the communication configurations as arguments, according to the order shown below.

- *sudo ./agent 'numNeighbors' 'own name' 'name N1' 'sending port N1' 'receiving port N1' 'name N2' 'sending port N2' 'receiving port N2' 'output file name'(.txt)*

The Raspberry Pi's are named 'dev1', 'dev2', 'dev3', 'dev4' and 'dev5'. An undirected edge in the communication graph is implemented via two UDP connections between the neighbors. You can use the incomplete graph bellow to keep track of the connections between agents.



During execution the agents will print out the values received from their neighbors and their own output.

6.2.4 Plotting the solutions

After running the simulation, you should plot the results save on the output file to check if output synchronization was achieved. The output file consists of columns separated by '\tab': The simulation time, the output information received from the neighbors and lastly the agents own calculated output.

Tip: Matlab can read .txt files into tables with the command 'readtable(filename)'