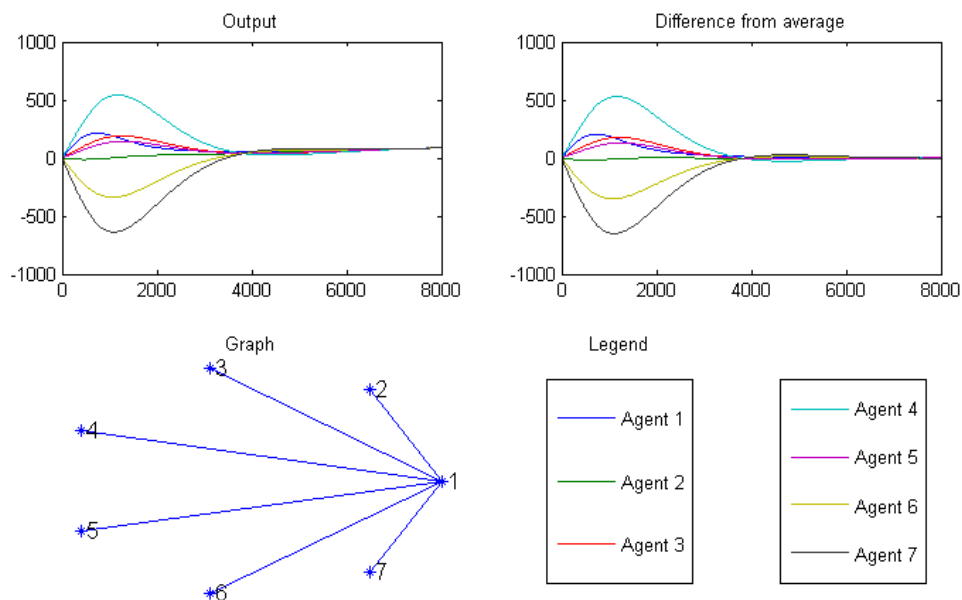




On the robust synchronisation of multi-agent systems



Bachelor Thesis in Applied Mathematics

August 2012

Student: J. Lanting

Supervisor: dr. M.K. Camlibel

Second supervisor: dr. M. Cao

Abstract

Recently H.L. Trentelman and K. Takaba proposed a protocol that robustly synchronises uncertain linear multi-agent systems. Given a network we allow each of the agents to deviate (within a certain radius) from a given nominal linear dynamic system, while still achieving synchronisation. we present an implementation of Trentelman-Takaba protocol and study several examples. We also modify the protocol to make it suitable for solving formation control problems and generate an animation of an example problem where agents moving on a plane while trying to form a formation.

Contents

1	Introduction	3
2	Preliminaries	4
2.1	Graph theory	4
2.2	System Theory	4
2.2.1	Transfer matrices	5
3	Synchronisation	5
3.1	Robust synchronisation	6
3.2	Computation	7
3.2.1	Maximizing the robust synchronisation radius η	8
3.3	Formation control	8
4	MATLAB implementation	9
4.1	Explanation of the inner workings of the tool-set	9
5	Results and examples	10
5.1	Synchronisation	10
5.2	Formation control	12
6	Conclusion	13
7	Acknowledgments	14
A	Source code listings	15
A.1	Graph related functions	15
A.2	Protocol generating functions	16
A.3	MATLAB state space object returning functions	18
A.4	Demonstrating scripts	19
B	Results	28
B.1	Additional results for the synchronisation of the double integrator	28
B.2	Additional snapshots of the formation control animation	30

1 Introduction

Recently interest has risen in networked multi-agent systems. These are systems where instead of one agent trying to achieve a goal, there are instead a number of network connected identical copies of agents trying to achieve a common goal. Each agent tries to achieve this by exchanging information with its neighbours. An agent is represented by an input/state/output system. Each agent is accompanied by its own controller. This controller uses the output of the neighbouring agents. The combination of all controllers is called a protocol. An important part of the theory of networked multi-agent systems concerns with the design of such protocols. The topology of the network is given by the network graph. The vertices represent the agents and the edges represent a connection between to agents. Depending on the problem the network graph might be a directed graph or a digraph, a weighted graph and even state- or time- dependent.

Several problems can be formulated in the setting described above. A good introduction to some of these is given in [1]. The most famous of these problems is perhaps the *consensus (or agreement) problem* where the agents are required to agree upon a certain quantity of interest. The agents could for instance be sensors and the value to agree upon could be sensor data.

Related is the *alignment problem* where the agents, which are initially moving in different directions, are required to move in the same direction i.e. to align as described in [2].

The *synchronisation problem* describes the problem of finding the conditions and protocol under which a network of agents converge to a common trajectory. This could for instance be the synchronisation of phase and frequency of coupled oscillators [1, p.218]. These coupled oscillators arise in many fields including physics, chemistry, biology, neuroscience and mathematics.

Also relevant are *flocking theory* and related the *rendezvous in space* problem. To flocks three basic rules apply [3]:

- “Flock Centering: attempt to stay close to nearby flockmates,”
- “Obstacle Avoidance: avoid collisions with nearby flockmates,”
- “Velocity Matching: attempt to match velocity with nearby flockmates.”

The rendezvous in space problem is equivalent to reaching consensus in position. In both flocking theory and the rendezvous in space problem the network graph depends on the actual position of the agents.

Finally there is also *distributed formation control* where agents are required to assume a state relative to that of their neighbours. This could for instance be a cluster of satellites orbiting in close proximity to each other around earth while keeping formation [4].

Of particular interest in all of the above problems and in systems theory in general are *robust controllers*. These are controllers that even achieve their goal if the agents have certain perturbations (within a certain neighborhood from the original)[5]. In real world examples these perturbations might for instance be caused by external forces not included in the model or minor defects that occur during the production process.

In this thesis we present an implementation of the protocol that was developed in [6] and that achieves robust synchronisation for linear multi-agent systems. We

modify the equations to make them work as a formation control problem and show some example simulations.

2 Preliminaries

2.1 Graph theory

A *graph* is a set of objects, called *nodes* or *vertices*, of which some might be connected by lines, called *edges*. If two vertices have an edge in between them we call those nodes *adjacent*. The *degree* of a node is the number of nodes it is adjacent to. More precisely a graph is an ordered pair $G = (V, E)$ with V the set of vertices and E the set of edges. In the case that each element of E is an unordered pair of elements from V we speak of an *undirected graph* and there is an edge between both vertices. If on the contrary the elements of E are ordered pairs (i, j) then there is an edge going from i to j , but not necessary back from j to i , then we call the graph *directed*. A graph is called *connected* if for every pair of distinct vertices there is a path between those vertices. In this thesis we only look at networks whose network topology is represented by undirected connected graphs.

The *adjacency matrix* is the matrix $A = (a_{ij})$ where $a_{ij} = 1$ if $(i, j) \in E$ and 0 otherwise. The *degree matrix* is the matrix $D = (d_{ij})$ where $d_{ij} = \deg(v_i)$ if $i = j$ and 0 otherwise. The *Laplacian matrix* of a graph is the matrix $L = D - A$. If the graph is undirected, then the Laplacian is a real symmetric positive semi definite matrix of rank $p - 1$, where p is the number of vertices. For each vertex i , we define the set of its neighbors as $N_i := \{j \in V | (i, j) \in E\}$.

2.2 System Theory

In this paper we consider linear time-invariant input/state/output systems of the form:

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) + Du(t) \end{aligned} \tag{2.1}$$

We call the vector $x(t)$ the state vector, $u(t)$ the input vector and $y(t)$ the output vector. Here $x(t) \in \mathbb{R}^n$ is the state, $u(t) \in \mathbb{R}^m$ is the input and $y(t) \in \mathbb{R}^q$ is the output of the system, and all the matrices involved are of appropriate sizes.

If for any initial value $x_0 \in \mathbb{R}^n$ and zero input it holds that $\lim_{t \rightarrow \infty} x(t, x_0) = 0$, then the system is called *stable*. An important property of stable systems is that they are also bounded, thus for every bounded input function $u(t)$ the output $y(t)$ will also be bounded.

A matrix is called *stable* or *Hurwitz* if every eigenvalue has strictly negative real part. If the matrix A of the system $[A, B, C, D]$ is Hurwitz, then the system is stable.

A system is called *controllable* if for any two states x_0, x_1 , a time $0 < t_1 < \infty$ and input function $u(t)$ exist such that $x(t_1, x_0, u(t_1)) = x_1$.

A weaker property is stabilizability. A system is called *stabilizable* if for any state x_0 , an input function $u(t)$ exists such that $\lim_{t \rightarrow \infty} x(t, x_0, u(t)) = 0$. Or more precisely, there exists a real $m \times n$ matrix F such that $A + BF$ is a Hurwitz matrix. Controllability implies stabilizability.

A system is called *observable* if a time $0 < t_1 < \infty$ exists such that for each input function $u(t)$, it follows from $y(t, x_0, u(t)) = y(t, x_1, u(t))$ for all $t \in [0, t_1]$, that $x_0 = x_1$.

A weaker condition is detectability. A system is called *detectable* if those parts of the linear space that form the state vector that are not observable are stable. I.e. there exists a real $n \times q$ matrix K such that the matrix $A - KC$ is Hurwitz. Observability implies detectability.

If A is an $m \times n$ matrix and B is a $p \times q$ matrix, then the *Kronecker product* $A \otimes B$ is the $mp \times nq$ matrix given by:

$$A \otimes B = \begin{pmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{pmatrix}$$

2.2.1 Transfer matrices

An alternative representation is given in the form of a *transfer matrix*. Although the transfer matrix can be calculated each time by hand by applying a Laplace transformation, for the system (2.1) the transfer matrix $G(s)$ is simply given by $G(s) = C(sI - A)^{-1}B + D$ and in this case each of the elements of the transfer matrix takes the form of a rational function.

A *proper transfer matrix* is a transfer matrix for which $\lim_{t \rightarrow \infty} H(s)$ is well-defined. And a transfer matrix is called *rational* if all of its elements are rational functions. A rational transfer matrix thus is proper if for each of the elements the degree of the numerator does not exceed the degree of the denominator of an element.

In the case of a system of the form (2.1) the properties of the transfer matrix (such as stability and eigenvalues) are inherited from the state space representation.

We denote the set of all proper and stable rational transfer matrices by $R\mathcal{H}_\infty$. The norm that is of interest to us on this set is given by $\|G\|_\infty = \sup_{\text{Re}(\lambda) \geq 0} \|G(\lambda)\|$.

For a more complete introduction to systems theory we would like to refer the reader to [7] and for an introduction to the Laplace transformation to [8].

3 Synchronisation

Consider a network of p identical systems. The network topology given by an undirected connected graph with graph Laplacian L and the dynamics of each agent are given by:

$$\begin{aligned} \dot{x}_i &= Ax_i + Bu_i \\ y_i &= Cx_i \end{aligned} \tag{3.1}$$

The assumption is made that this agent is both detectable and stabilizable. The proof that the requirement of stabilizability (and not controllability) is sufficient can, together with a stabilizing protocol, be found in [9]. A protocol of the following form is suggested in [6]:

$$\dot{w}_i = Aw_i + BF \sum_{j \in N_i} (w_i - w_j) + G \left(\sum_{j \in N_i} (y_i - y_j) - Cw_i \right), u_i = Fw_i \tag{3.2}$$

the closed loop dynamics of the whole network are thus given by:

$$\begin{aligned}\dot{x} &= (I \otimes A)x + (I \otimes B)u \\ y &= (I \otimes C)x \\ \dot{w} &= [I \otimes (A - GC) + L \otimes BF]w + (L \otimes G)y \\ u &= (I \otimes F)w\end{aligned}$$

With F and G matrices of size $m \times n$ and $n \times q$ and I the identity matrix of size p . The previous can be assembled into a single system:

$$\begin{pmatrix} \dot{x} \\ \dot{w} \end{pmatrix} = \begin{pmatrix} I \otimes A & I \otimes BF \\ L \otimes GC & I \otimes (A - GC) + L \otimes BF \end{pmatrix} \begin{pmatrix} x \\ w \end{pmatrix}$$

The network is said to be synchronised by the protocol if for all $i, j = 1, 2, \dots, p$ we have $x_i - x_j \rightarrow 0$ and $w_i - w_j \rightarrow 0$ as $t \rightarrow \infty$.

3.1 Robust synchronisation

For this section we again consider a network where the individual agent dynamics are given by (3.1). However, we now take into consideration that even though the systems might be more or less equal, there still might be minor differences. We limit these differences to within a certain ball η around our original agent with respect to the H_∞ norm. Recall that if we put the system (3.1) in it's transfer matrix form we get $G(s) = C(sI - A)^{-1}B$. We now consider the perturbed system $G(s) + \Delta_i(s)$, where $\Delta_i \in R\mathcal{H}_\infty$ and $\|\Delta_i\| \leq \eta$, where η is the uncertainty radius. Of course if we write:

$$\Delta_i(s) = C_{\Delta_i}(sI - A_{\Delta_i})^{-1}B_{\Delta_i} + D_{\Delta_i}$$

then we can easily put the perturbation in state space form:

$$\begin{aligned}\dot{\xi}_i &= A_{\Delta_i}\xi_i + B_{\Delta_i}z_i \\ d_i &= C_{\Delta_i}\xi_i + D_{\Delta_i}z_i\end{aligned}\tag{3.3}$$

which is interconnected with the original agent as follows:

$$\begin{aligned}\dot{x}_i &= Ax_i + Bu_i \\ y_i &= Cx_i + d_i \\ z_i &= u_i\end{aligned}\tag{3.4}$$

Trentelman and Takaba [6] modify the protocol (3.2) to include a weighting factor on the Laplacian L , which leads to the protocol

$$\begin{aligned}\dot{w}_i &= Aw_i + BF \sum_{j \in N_i} \frac{1}{N} (w_i - w_j) + G \left(\sum_{j \in N_i} \frac{1}{N} (y_i - y_j) - Cw_i \right) \\ u_i &= Fw_i\end{aligned}\tag{3.5}$$

The closed loop dynamics of the whole network are thus given by:

$$\begin{aligned}
\dot{x} &= (I \otimes A)x + (I \otimes B)u \\
y &= (I \otimes C)x + (I \otimes I)d \\
z &= u \\
\dot{w} &= [I \otimes (A - GC) + \frac{1}{N}L \otimes BF]w + \frac{1}{N}(L \otimes G)y + \frac{1}{N}(L \otimes G)d \\
u &= (I \otimes F)w \\
d &= \begin{pmatrix} \Delta_1 & 0 & \cdots & 0 \\ 0 & \Delta_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \Delta_p \end{pmatrix} z
\end{aligned} \tag{3.6}$$

Or alternatively combining the agents \dot{x} and protocols \dot{w} :

$$\begin{pmatrix} \dot{x} \\ \dot{w} \end{pmatrix} = \begin{pmatrix} I \otimes A & I \otimes BF \\ \frac{1}{N}L \otimes GC & I \otimes (A - GC) + \frac{1}{N}L \otimes BF \end{pmatrix} \begin{pmatrix} x \\ w \end{pmatrix} + \begin{pmatrix} 0 \\ \frac{1}{N}L \otimes G \end{pmatrix} d \tag{3.7}$$

$$z = \begin{pmatrix} 0 & I \otimes F \end{pmatrix} \begin{pmatrix} x \\ w \end{pmatrix} \tag{3.8}$$

3.2 Computation

Computation of suitable F, G and N goes as follows[6]:

1. Take λ_2 and λ_p respectively the eigenvalues of smallest non zero magnitude and largest magnitude of L ,
2. Choose $N > \frac{\lambda_p^2}{\lambda_2^2}$,
3. Choose γ in the interval $\left(\frac{\lambda_p^2}{N^2}, \frac{\lambda_2^2}{N}\right)$,
4. Compute the maximal real symmetric solutions $\bar{P}(\sigma)$ and $Q(\tau)$:

$$A^T \bar{P} + \bar{P}A - \bar{P}BB^T \bar{P} + \sigma I = 0 \tag{3.9}$$

$$AQ + QA^T - QC^T CQ + \tau I = 0 \tag{3.10}$$

(although any Q for which the previous is smaller than 0 will do)
and let

$$P(\gamma) := \frac{1}{\gamma} \bar{P} \tag{3.11}$$

where σ and τ are positive real numbers and we take $\lim_{\sigma, \tau \rightarrow 0}$. The equations (3.9) and (3.10) are forms of continuous-time algebraic Riccati equations. There are algorithms available that can produce solutions to these problems and a numerical solver is for instance included within the matlab control system toolbox [10].

5. Choose a value of the synchronisation radius $\eta < \frac{1}{\sqrt{\rho(P(\gamma)Q)}}$, where the function ρ returns the spectral radius.
6. Compute:

$$F := -B^T P(\gamma) \quad (3.12)$$

$$G := (I - \eta^2 Q P(\gamma))^{-1} Q C^T \quad (3.13)$$

3.2.1 Maximizing the robust synchronisation radius η

From 3.2 we learn that we must choose $\eta < \frac{1}{\sqrt{\rho(P(\gamma)Q)}} = \frac{\sqrt{\gamma}}{\sqrt{\rho(\bar{P}Q)}}$. It can be seen that if γ increases then η increases as well. Thus, if we take a value $\delta > 0$ as small as possible, optimal choices are:

$$N_{opt} = \frac{\lambda_p^2 + \delta}{\lambda_2} \quad (3.14)$$

$$\gamma_{opt} = \frac{\lambda_2}{N} = \frac{\lambda_2^2}{\lambda_p^2 + 2\delta} \quad (3.15)$$

$$\eta = \frac{1}{\sqrt{\rho(P(\gamma)Q)}} - \delta = \frac{\lambda_2}{\sqrt{\lambda_p^2 + 2\delta}} \frac{1}{\sqrt{\rho(\bar{P}Q)}} - \delta \quad (3.16)$$

3.3 Formation control

In this section we try to modify the previous problem in a problem of formation control. Recall that in our previous problem what we in fact were trying to do was to minimize the differences in the output:

$$\sum_{j \in N_i} (y_i - y_j)$$

In the formation control problem we instead require the outputs y_i and y_j to be a certain vector r_{ij} apart:

$$\sum_{j \in N_i} (y_i - y_j - r_{ij}) = \sum_{j \in N_i} (y_i - y_j) - \sum_{j \in N_i} r_{ij} = \sum_{j \in N_i} (y_i - y_j) - b_i$$

We call b_i the *bias* on the i th output. A nonzero bias has no effect on the choice of the protocol [1]. If we apply this new theory to our previously computed state space system (3.7), that represented the global network dynamics, we get:

$$\begin{aligned} \begin{pmatrix} \dot{x} \\ \dot{w} \end{pmatrix} &= \begin{pmatrix} I \otimes A & I \otimes BF \\ \frac{1}{N} L \otimes GC & I \otimes (A - GC) + \frac{1}{N} L \otimes BF \end{pmatrix} \begin{pmatrix} x \\ w \end{pmatrix} \\ &+ \begin{pmatrix} 0 & 0 \\ \frac{1}{N} L \otimes G & \frac{1}{N} I \otimes G \end{pmatrix} \begin{pmatrix} d \\ b \end{pmatrix}. \end{aligned} \quad (3.17)$$

Instead of manually trying to compute the bias for the neighbours of each agent we create a reference signal y_{ref} that contains the formation though can be shifted

along each of the q axis.

$$\begin{pmatrix} \dot{x} \\ \dot{w} \end{pmatrix} = \begin{pmatrix} I \otimes A & I \otimes BF \\ \frac{1}{N}L \otimes GC & I \otimes (A - GC) + \frac{1}{N}L \otimes BF \end{pmatrix} \begin{pmatrix} x \\ w \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ \frac{1}{N}L \otimes G & -\frac{1}{N}L \otimes G \end{pmatrix} \begin{pmatrix} d \\ y_{\text{ref}} \end{pmatrix} \quad (3.18)$$

4 MATLAB implementation

In the following section we provide a small toolbox around the above protocol in MATLAB [11, 12]. MATLAB was chosen for its advanced capabilities in linear algebra and its large set of well-written toolboxes. These toolboxes (most notably the Simulink and control systems toolboxes) allowed us to focus on the problem itself rather than on language quirks and badly documented external libraries. Using MATLAB also meant we could use the same environment we used to compute our results to visualise our data. Another important factor for the choice was the familiarity of the author of this report with the environment.

4.1 Explanation of the inner workings of the tool-set

The goal of the tool-set is to see the protocol described by Trentelman and Takaba ‘in action’. The working of tool-set is globally explained by Figure 1. Each block represents the data that needs to be generated at that point. The MATLAB functions and scripts that achieve this are included in the appendices. Obviously

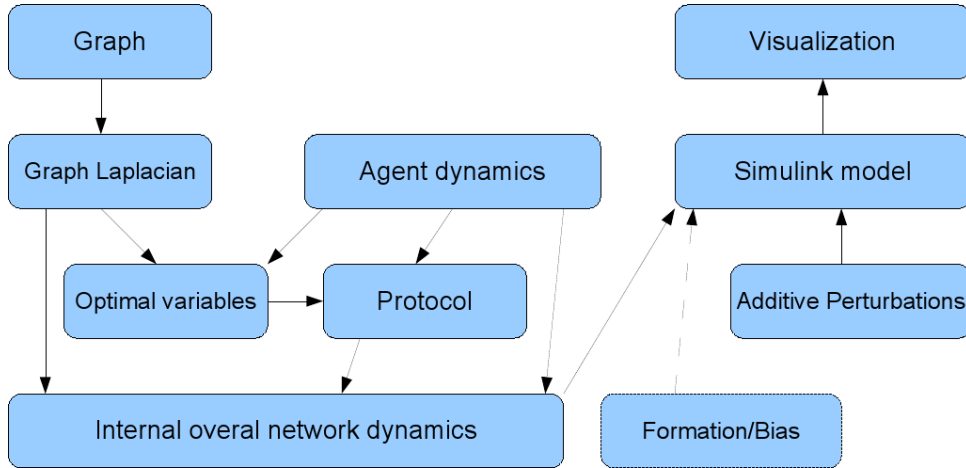


Figure 1

visualisation of the generated data in a nice manner is problem dependent. Two example problems will be reviewed in the next section. Instead of actually calculating the limits in equations (3.9) and (3.10), we force a fixed small value of σ and τ . MATLABs internal `care` function is used for computing the solutions to the Riccati equations. We also choose a fixed small value for δ when we try to maximise the radius of stability η .

After all the data is gathered, it's then all put together in a Simulink model, where the system of additive perturbations and the function or constant containing the formation are interconnected with the main system (figure 2). A second output is added to the main state space system to allow output to the workspace. In our examples this output is the same as the output of the agents visible to the controllers. The Simulink system is then simply 'run' using the default integrator and the output of the Simulink model can then be further processed.

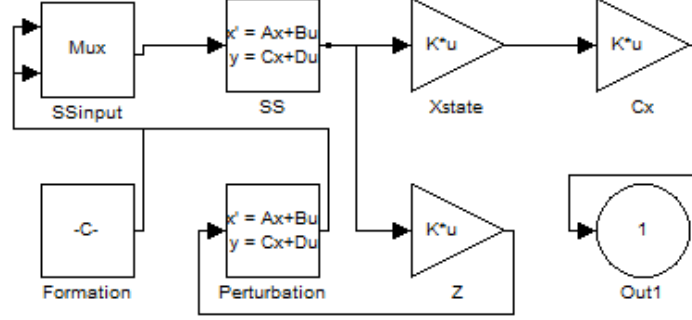


Figure 2

5 Results and examples

Two example problems are implemented in MATLAB scripts. We first look at a synchronisation problem and then modify that into a formation control problem.

5.1 Synchronisation

Let the agent dynamics of the nominal system be given by:

$$\begin{aligned} \dot{x} &= \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} x + \begin{pmatrix} 0 \\ 1 \end{pmatrix} u \\ y &= \begin{pmatrix} 1 & 0 \end{pmatrix} x \end{aligned} \tag{5.1}$$

The above system is called a double integrator. In the demo file any network topology can be given, but a full graph, circle graph and star graph of n nodes are predefined and easily selectable (figure 3). The simulink system is then seeded with a random initial condition and run for a fixed amount of time. For all three of the above types of graphs synchronisation was achieved in our tests. One thing of particular interest that we see in the results is that for graphs of the same type, but with a different amount of nodes, synchronisation is reached at about the same time. The results of one of those runs is given in figure 4 and further results are given in appendix B.

As a second test we slightly modify our previous system. Our new system is

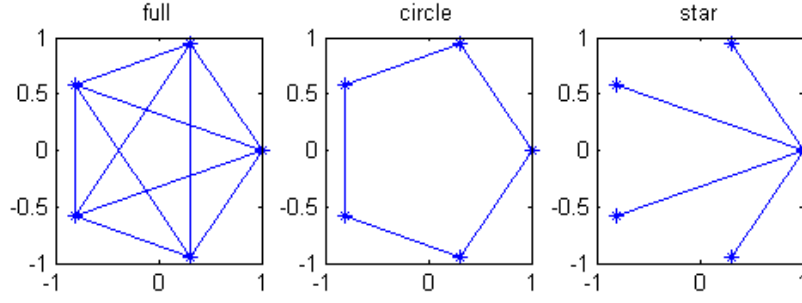


Figure 3: Different types of common graphs.

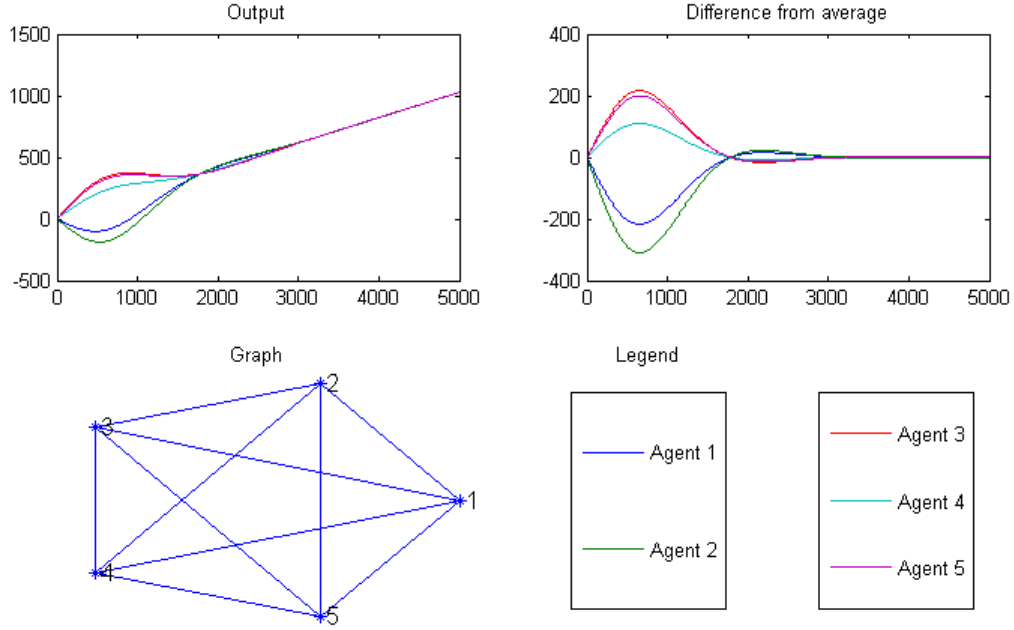


Figure 4: Synchronisation of a full graph with 5 nodes.

now given by:

$$\begin{aligned} \dot{x} &= \begin{pmatrix} 0.1 & 1 \\ 0 & 0.1 \end{pmatrix} x + \begin{pmatrix} 0 \\ 1 \end{pmatrix} u \\ y &= \begin{pmatrix} 1 & 0 \end{pmatrix} x \end{aligned} \tag{5.2}$$

The eigenvalues of the new system are now both 0.1 instead of purely imaginary as with our previous system. Even though this system is both controllable and observable (and thus stabilizable and detectable) the model failed to run. Figure 5 shows why. Even though the differences in output of the different agents behave just like our previous model, the overall value they're trying to agree upon quickly grows out of bounds. Manually checking if the protocol is indeed working by checking the stability of the system in equation (27) in [6] shows that for all but $\lambda = 0$ equation (27) is indeed stable. MATLAB thus appears not suitable for integrating unbounded

systems.

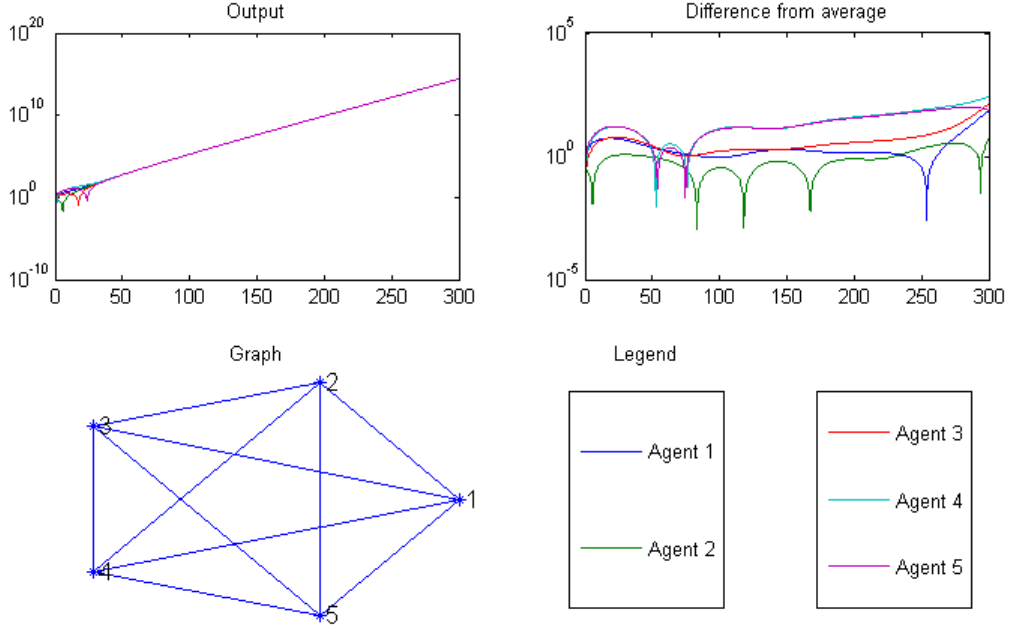


Figure 5: The tool-set fails for unbounded systems. Note the logarithmic scale used here.

5.2 Formation control

The previous example of the double integrator makes perfect sense when modified into a formation control problem: If an agent is made out of two double integrators, then the input might represent acceleration or force in the x and y direction and the output the position. The bias might thus represent the actual difference in position of two different agents. We thus have:

$$\begin{aligned} \dot{x} &= \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} x + \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix} u \\ y &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} x \end{aligned} \tag{5.3}$$

As a bias we have chosen for a constant value. Since we already had the code to generate the coordinates of n nodes on a circle, we chose the circle as our formation. In our example we've also chosen for a circle graph as our network topology. The initial condition of each agent is a random position, and no initial velocity. The whole state is rendered as an animation over time. The final result can be seen in figure 6 with additional snapshots of the animation available in appendix B.

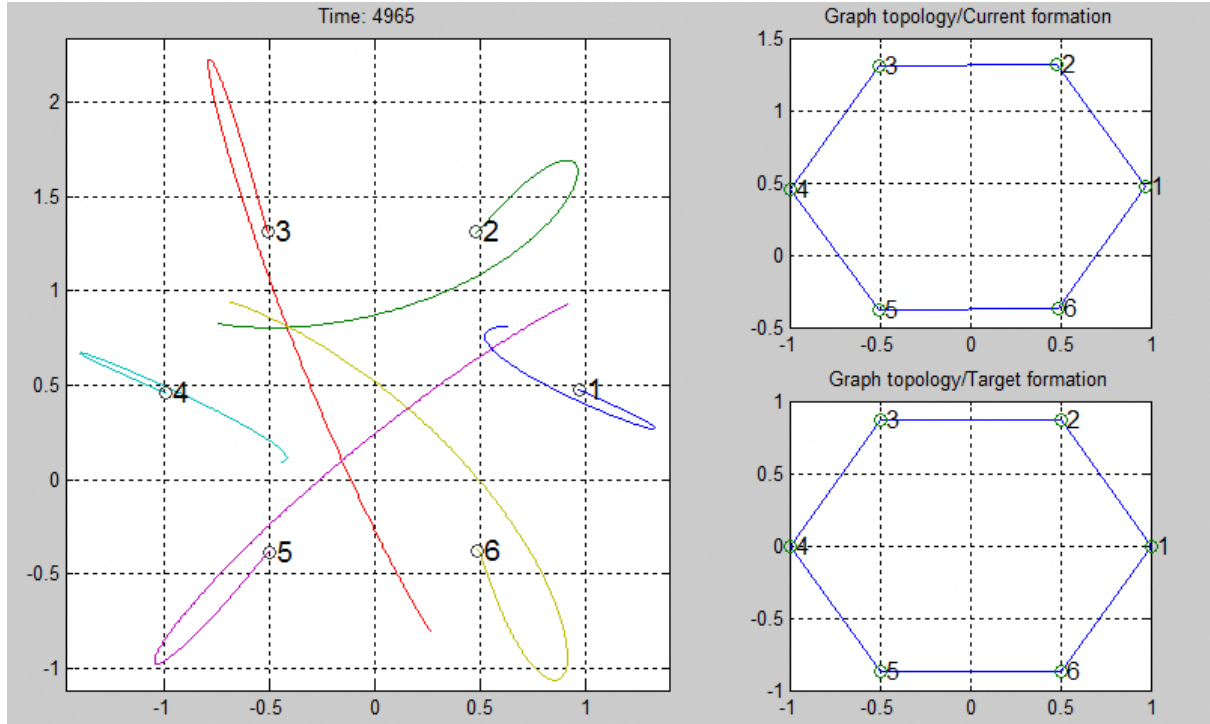


Figure 6: The colored lines represent the path traveled by each node.

6 Conclusion

In this thesis we have looked into a numerical implementation of the protocol developed by H.L. Trentelman and K. Takaba described in [6]. Even though MATLAB is capable of computing the protocol for all our tested problems, we encountered problems integrating the state space system containing the network dynamics that followed. Even though the output of the different agents might be synchronised, the value of this output might still grow out of bounds quickly. This limits the use of the MATLAB implementation, since we basically can't use agents which have eigenvalues on the right side of the imaginary plane. This is a problem inherent to finite precision. Furthermore agents for which all eigenvalues have a negative real part are a bit boring to investigate, since those are exponentially stable by themselves. This leaves us with agents for which at least some of the eigenvalues are purely imaginary and have no eigenvalues with a positive real part, such as the double integrator.

We also modified the problem into a formation control problem while keeping most of the original protocol untouched. This allowed us to use the same solvers for the matrices F and G .

For the double integrator both demonstration scripts generated good results. Apart from generating static images we also generated an illustrative animation to go with the formation control problem.

7 Acknowledgments

The author would like to thank dr M.K. Camlibel, for all his support and excellent guidance, dr. M. Cao, for willing to be the second supervisor, Prof. dr. H.L. Trentelman and K. Takaba, for providing me the article to base my thesis on and S. Baars BSc., for willing to proofread my thesis on such short notice.

References

- [1] R. Olfati-Saber, J. Fax, and R. Murray, “Consensus and cooperation in networked multi-agent systems,” *Proceedings of the IEEE*, vol. 95, no. 1, pp. 215–233, 2007.
- [2] A. Jadbabaie, J. Lin, and A. S. Morse, “Coordination of groups of mobile autonomous agents using nearest neighbor rules,” *IEEE Transactions on Automatic Control*, vol. 48, pp. 988–1001, June 2003.
- [3] R. Olfati-Saber, “Flocking for multi-agent dynamic systems: Algorithms and theory,” *IEEE Transactions on Automatic Control*, vol. 51, pp. 401–420, March 2006.
- [4] R. Burns, C. McLaughlin, J. Leitner, and M. Martin, “Techsat21: Formation design, control, and simulation,” *Proceedings of the IEEE Aerospace Conference*, pp. 19–25, 2000.
- [5] H. Trentelman, A. Stoorvogel, and M. Hautus, *Control Theory for Linear Systems*. London: Springer, 2001.
- [6] H. Trentelman and K. Takaba, “Robust synchronisation of uncertain linear multi-agent systems.” accepted for publication in *IEEE Transactions in Automatic Control*, December 2011.
- [7] H. Trentelman, A. Stoorvogel, and M. Hautus, *Mathematical Systems Theory*. Delft: VSSD, 3rd ed., 2005.
- [8] H. ter Morsche and G. Meinsma, “Signals and systems.” lecture notes, 2001.
- [9] Z. Li, G. Chen, and L. Huang, “Consensus of multi-agent systems and synchronisation of complex networks: a unified point of view,” *IEEE Transactions on Circuits and Systems*, vol. 57, pp. 213–224, January 2010.
- [10] MathWorks, <http://www.mathworks.nl/help/toolbox/control/>, *Control Systems Toolbox Documentation*.
- [11] MathWorks, <http://www.mathworks.nl/help/techdoc/>, *MATLAB Documentation*.
- [12] MathWorks, <http://www.mathworks.nl/help/toolbox/simulink/>, *Simulink Documentation*.

Appendices

A Source code listings

A.1 Graph related functions

These functions were used to generate some different types of graphs and calculate their Laplacian.

../MATLAB/CircleGraph.m

```
1 function G = CircleGraph(P)
2 % Returns the adjoint matrix for a circle graph with P nodes
3   G = VEtoAdj(P, [[1:P]', [2:P, 1]'] );
4 end
```

../MATLAB/CompleteGraph.m

```
1 function G = CompleteGraph(P)
2 % Returns the adjoint matrix for a complete graph with P
   nodes
3   G = ones(P) - eye(P);
4 end
```

../MATLAB/StarGraph.m

```
1 function G = CompleteGraph(P)
2 % Returns the adjoint matrix for a complete graph with P
   nodes
3   G = zeros(P);
4   G(1,2:P)=1;
5   G(2:P,1)=1;
6 end
```

../MATLAB/VEtoAdj.m

```
1 function Adj = VEtoAdj(P, E)
2 % Adj = VEtoAdj(P, E)
3 % Returns the adjacency matrix of the undirected graph with
   p vertices together with its set of edges.
4 % P is the number of vertices
5 % E is an nx2 matrix consisting of the n pairs of vertices
   forming the edges.
6   Adj = zeros(P,P);
7   for i = E'
8       Adj(i(1),i(2)) = 1;
9       Adj(i(2),i(1)) = 1;
10  end
11 end
```

../MATLAB/GraphLaplacian.m

```

1 function L = GraphLaplacian(P, E)
2 % L = GraphLaplacian(P, E)
3 % Returns the laplacian of the undirected graph with p
   vertices together with its set of edges.
4 % P is the number of vertices
5 % E is an nx2 matrix consisting of the n pairs of vertices
   forming the edges.
6 %
7 % L = GraphLaplacian(A)
8 % Returns the graph laplacian of the adjacency matrix A
9
10 if nargin == 2
11     A = VEtoAdj(P, E);
12 else
13     A = P;
14 end
15 L = diag(sum(A)) - A;
16 end

```

A.2 Protocol generating functions

The following two functions calculate the matrices F and G together with their variables that maximise the radius of robust synchronisation.

../MATLAB/RobustSyncControl.m

```

1 function [F G] = RobustSyncControl(A, B, C, L, N, eta, gamma
   , ssigma, tau)
2 % [F G] = RobustSyncControl(A, B, C, L, N, eta, gamma)
3 % Returns the gains F and G given by (33) and (34)
4 %
5 % [F G] = RobustSyncControl(A, B, C, L, N, eta, gamma, sigma
   )
6 % [F G] = RobustSyncControl(A, B, C, L, N, eta, gamma, sigma
   , tau)
7 % The optional arguments 'sigma' and 'tau' force  $AQ+QA'-QC'$ 
    $CQ = -\tau I$  thus making
8 % sure the condition  $AQ+QA'-QC'CQ < 0$  is met. (defaults to
    $\tau=10^{-10}$ )
9 % and  $A'P + PA - \gamma PBB'P = -\sigma I$  making computation
   easier if A has (close to)
10 % purely imaginary eigenvalues (defaults to  $\sigma=10^{-10}$ )
11
12 % compute the eigenvalues of the graph laplacian.
13 E = sort(real(eig(L)));
14 lambda2 = E(2);
15 lambdaP = E(end);
16
17 % test for suitable choices of N and gamma (eq (30) and
   (31))

```



```

18 if N <= lambdaP^2/lambda2
19     warning('N out of bounds');
20 end
21 if ~(( lambdaP^2/N^2 <= gamma) && (gamma < lambda2/N))
22     warning('gamma out of bounds');
23 end
24
25 % setting ssigma, tau
26 if nargin < 8
27     tau = 10^-10;
28 end
29 if nargin < 9
30     ssigma = 10^-10;
31 end
32 % calculate P and Q as given by (28) and (29)
33 P = care(A, B, ssigma*eye(size(A)), 1/gamma*eye(size(B,2))
    );
34 Q = care(A', C', tau*eye(size(A')));
35
36 % test if our eta is a good choice (eq 32)
37 if eta >= 1/sqrt(max(abs(eig(P*Q))))
38     warning('eta out of bounds');
39 end
40
41 % calculate F and G (eq (33) and (34))
42 F = -B'*P;
43 G = inv(eye(size(Q*P)) - eta^2*Q*P) *Q*C';
44 end

```

../MATLAB/RobustSyncOptimalVars.m

```

1 function [N, eta, gamma] = RobustSyncOptimalVars(A, B, C, L,
    delta, ssigma, tau)
2 % [N, eta, gamma] = RobustSyncOptimalvars(A, B, C, L)
3 % gives choices for N, eta and gamma maximising eta using
   section 6 of the article.
4 % Default values for are used for delta, and sigma and tau.
5 %
6 % [N, eta, gamma] = RobustSyncOptimalVars(A, B, C, L, delta,
   sigma, tau)
7 % gives optimal choices for N, eta and gamma maximising eta.
8
9 %select default values for delta, sigma, tau
10 if nargin < 5
11     delta = 10^-6;
12     ssigma = 10^-6;
13     tau = 10^-6;
14 end
15
16 % compute the eigenvalues the graph laplacian.

```

```

17 E = sort(real(eig(L)));
18 lambda2 = E(2);
19 lambdaP = E(end);
20
21 % calculate P and Q as given by (48) and (49)
22 P = care(A, B, ssigma*eye(size(A)));
23 Q = care(A', C', tau*eye(size(A')));
24
25 eta = lambda2 / sqrt(lambdaP^2 + 2 * delta) * 1/sqrt(max(
    abs(eig(P*Q))) - delta;
26 gamma = lambda2^2/(lambdaP^2 + 2*delta);
27 N = (lambdaP^2 + delta)/lambda2;
28
29 %testing if the found numbers are sane
30 if ~(0 < eta) && (eta < (1/sqrt(max(abs(eig(P*Q))))))
31     warning('eta out of bounds. Choose a smaller value delta
        ');
32 end
33 if N <= lambdaP^2/lambda2
34     warning('N out of bounds. Choose a smaller value delta')
        ;
35 end
36 if ~(lambdaP^2/N^2 <= gamma) && (gamma < lambda2/N)
37     warning('gamma out of bounds. Choose a smaller value
        delta');
38 end
39 end

```

A.3 MATLAB state space object returning functions

../MATLAB/CreateMAS.m

```

1 function MAS = CreateMas( A, B, C, L, F, G, N)
2 % MAS = CreateMas( A, B, C, L, F, G, N)
3 % creates an SS object MAS representing the continuous-time
   state-space model described in eq.(19) (20) (21)
4 p = size(L,1);
5 I = eye(p);
6
7 SSA = [ kron( I, A),          kron( I, B * F);
8         1/N * kron( L, G * C), kron( I, A - G * C) + kron(
           1/N * L, B * F)];
9 SSB = [zeros( p * size( G)); kron( 1/N * L, G)];
10 SSC = [zeros( p * size( F)), kron( I, F)];
11 SSD = 0;
12 MAS = ss(SSA, SSB, SSC, SSD);
13 end

```

../MATLAB/perturbation.m

```

1 function [sys] = perturbation(m, q, eta)
2 % [sys] = perturbation(m, q, eta)
3 % Returns a random stable space state system with m inputs
   and q outputs such that its norm is smaller than eta
4
5
6   n=max(m,q);
7   A=rand(n,n);
8   B=rand(n,m);
9   C=rand(q,n);
10  D=rand(q,m);
11  A = A - eye(n)*2*max(real(eig(A))); %move the eigenvalues
   to left of the imaginary axis
12  A = - rand * eta * A / min(real(eig(A))); %scales the
   eigenvalues s.t. ||sys|| < eta
13  sys = ss(A,B,C,D);
14 end

```

../MATLAB/perturbations.m

```

1 function [sys] = perturbations(p, m, q, eta)
2 % function [sys] = perturbations(p, m, q, eta)
3 % Returns p random stable space state systems with m inputs
   and q outputs such that
4 % each its norm is smaller than eta and A is hurwiz, then
   puts them all in
5 % a single block diagonal system.
6
7   n=max(m,q);
8   sys = ss(zeros(p*n), zeros(p*n,p*m), zeros(p*q, p*n),
   zeros(p*q, p*m));
9   for i=0:p-1,
10     tmpsys = perturbation(m, q, eta);
11     sys.A(((i*n)+1):((i+1)*n), ((i*n)+1):((i+1)*n)) = tmpsys
   .A;
12     sys.B(((i*n)+1):((i+1)*n), ((i*m)+1):((i+1)*m)) = tmpsys
   .B;
13     sys.C(((i*q)+1):((i+1)*q), ((i*n)+1):((i+1)*n)) = tmpsys
   .C;
14     sys.D(((i*q)+1):((i+1)*q), ((i*m)+1):((i+1)*m)) = tmpsys
   .D;
15 end
16 end

```

A.4 Demonstrating scripts

demoDoubleIntegratorSynchronisation.m and demoFormationControl.m both set up the complete simulink model and run the simulation.

demoDoubleIntegratorSynchronisation.m then generates a nice plot of its results. If an annimation of the results of demoFormationControl.m should

be generated and then be saved as an .avi file the following commands should be executed in MATLAB:

```

1 demoFormationControl;
2 animateFormationControlFancy;
3 movie2avi(M, 'DemoFormationControl.avi');

    ../MATLAB/demoDoubleIntegratorSynchronisation.m
1 %create a small simulink model to test if a system of
  integrators synchronises:
2
3 %%individual agent dynamics (double integrator)
4 A = [0 1; 0 0]; B = [0; 1]; C = [1 0];
5
6 %uncomment to select the network topology and number of
  agents
7 Graph=CompleteGraph(10);
8 %Graph=CircleGraph(9);
9 %Graph=StarGraph(7); %star graph
10
11 L = GraphLaplacian(Graph);
12
13 %some useful numbers
14 n=size(A,1);
15 m=size(B,2);
16 q=size(C,1);
17 p = size(L,1);
18
19 StopTime = '5000'; %time to run the simulation
20 OutputTimes = mat2str(0:5:5000); %points that should be
  saved
21
22 %setting up the initial condition(w(t=0) is left 0, x(t=0)
  is seeded with random numbers (-1,1))
23 X0=zeros(2*p*n,1);
24 X0(1:p*n)=2*rand(p*n,1)-1;
25
26 %Compute the protocol
27 [N, eta, gamma] = RobustSyncOptimalVars(A, B, C, L);
28 [F G] = RobustSyncControl(A, B, C, L, N, eta, gamma);
29
30 %compute the added perturbations
31 pertsys = perturbations(p, m, q, 0*eta);
32
33 %compute the overal network dynamics
34 I = eye(p);
35 SSA = [ kron( I, A),          kron( I, B * F);

```

```

36         1/N * kron( L, G * C), kron( I, A - G * C) + kron(
           1/N * L, B * F)];
37 SSB = [zeros( p * size( G)); kron( 1/N * L, G)];
38 SSC = [zeros( p * size( F)), kron( I, F)];
39
40 %build the simulink system:
41 sys = 'Model';
42 bdclose(sys)
43 new_system(sys) % Create the model
44 open_system(sys) % Open the model
45
46 %some constants defining the spacing in simulink
47 x = 50;
48 y = 50;
49 w = 50;
50 h = 50;
51 offset = 100;
52
53 %adding blocks and links
54 pos = [(x+offset*0) (y+offset*0) (x+offset*0)+w (y+offset*0)
        +h];
55 add_block('built-in/State-Space',[sys '/SS'],'Position',pos)
        ;
56
57 pos = [(x+offset*1) (y+offset*0) (x+offset*1)+w (y+offset*0)
        +h];
58 add_block('built-in/Gain',[sys '/Xstate'],'Position',pos);
59
60 pos = [(x+offset*1) (y+offset*1) (x+offset*1)+w (y+offset*1)
        +h];
61 add_block('built-in/Gain',[sys '/Z'],'Position',pos);
62
63 pos = [(x+offset*0) (y+offset*1) (x+offset*0)+w (y+offset*1)
        +h];
64 add_block('built-in/State-Space',[sys '/Perturbation'],'
        Position',pos);
65
66 pos = [(x+offset*2) (y+offset*0) (x+offset*2)+w (y+offset*0)
        +h];
67 add_block('built-in/Scope',[sys '/Scopel'],'Position',pos);
68
69 pos = [(x+offset*2) (y+offset*1) (x+offset*2)+w (y+offset*1)
        +h];
70 add_block('built-in/Gain',[sys '/Cx'],'Position',pos);
71
72 pos = [(x+offset*2) (y+offset*2) (x+offset*2)+w (y+offset*2)
        +h];
73 add_block('built-in/Outport',[sys '/Out1'],'Position',pos);
74

```

```

75 add_line(sys,'SS/1','Xstate/1','autorouting','on');
76 add_line(sys,'Xstate/1','Scope1/1','autorouting','on');
77 add_line(sys,'SS/1','Z/1','autorouting','on');
78 add_line(sys,'Z/1','Perturbation/1','autorouting','on');
79 add_line(sys,'Perturbation/1','SS/1','autorouting','on');
80 add_line(sys,'Xstate/1','Cx/1','autorouting','on');
81 add_line(sys,'Cx/1','Out1/1','autorouting','on');
82
83 %adding parameters
84 set_param([sys '/SS'], 'A', mat2str(SSA), 'B', mat2str(SSB),
    'C', mat2str(eye(size(SSA))), 'D', mat2str(zeros(size(
    SSB))), 'X0', mat2str(X0));
85 set_param([sys '/Z'], 'Multiplication', 'Matrix(K*u)', 'Gain
    ', mat2str(SSC));
86 set_param([sys '/Perturbation'], 'A', mat2str(pertsys.A), 'B
    ', mat2str(pertsys.B), 'C', mat2str(pertsys.C), 'D',
    mat2str(pertsys.D), 'X0', mat2str(zeros(size(pertsys.A
    ,1),1)));
87 set_param([sys '/Xstate'], 'Multiplication', 'Matrix(K*u)',
    'Gain', mat2str([eye(size(A)*p), zeros(size(A)*p)]));
88 set_param([sys '/Cx'], 'Multiplication', 'Matrix(K*u)', '
    Gain', mat2str(kron(I,C)));
89 set_param(sys, 'StopTime', StopTime, 'OutputOption', '
    SpecifiedOutputTimes', 'OutputTimes', OutputTimes);
90 %enable simulink accelerator mode. Requires a compatible C
    compiler. Run 'mex -setup' if matlab isn't yet
    configured
91 %set_param(sys, 'SimulationMode', 'accelerator');
92
93 %run the simulation
94 sim(sys);
95
96 %plotting the output:
97 fig=figure;
98 subplot(2,2,1);
99 plot1=plot(tout, yout);
100 title('Output');
101
102 %plot the difference between each node and the average
103 subplot(2,2,2);
104 plot2=plot(tout, bsxfun(@minus,yout,mean(yout')));
105 title('Difference from average');
106
107 %plot the graph
108 subplot(2,2,3);
109 %put the nodes equally spaced on a circle
110 xy = [cos(0:2*pi/size(L,1):2*pi-2*pi/size(L,1))', sin(0:2*pi
    /size(L,1):2*pi-2*pi/size(L,1))'];
111 gplot(Graph, xy,'-*');

```

```

112 axis off;
113 for j = 1:p,
114     text(xy(j,1), xy(j,2), [' ' int2str(j)], 'FontSize',12);
115 end
116 title('Graph');
117
118 %add a legend
119 sh=subplot(2,4,7);
120 s = cell(1, floor(p/2));
121 for i = 1:p,
122     s{i} = sprintf('Agent %d', i);
123 end
124 leg1=legend(sh,plot1(1:floor(p/2)),s{1:floor(p/2)});
125 set(leg1,'position', get(sh,'position'));
126 title(sh,'Legend');
127 axis(sh,'off');
128 sh=subplot(2,4,8);
129 leg2=legend(sh,plot1(floor(p/2+1):p),s{floor(p/2+1):p});
130 set(leg2,'position', get(sh,'position'));
131 axis(sh,'off');
132
133 width=800;
134 height=480;
135 set(fig, 'Position', [0 0 width+1 height+1]);

```

../MATLAB/demoFormationControl.m

```

1 %creates a small simulink model to test robust formation
  controll.
2 %2 double integrators are used to simulate acceleration —>
  speed —> position of of the agent in the x and y
  direction.
3
4 %individual agent dynamics
5 A = [0 1 0 0; 0 0 0 0; 0 0 0 1; 0 0 0 0];
6 B = [0 0; 1 0; 0 0; 0 1];
7 C = [1 0 0 0; 0 0 1 0];
8
9 %uncomment to select the network topology and number of
  agents
10 %Graph=CompleteGraph(6);
11 Graph=CircleGraph(15);
12 %Graph=StarGraph(6); %star graph
13
14 L = GraphLaplacian(Graph);
15
16 %some useful numbers
17 n=size(A,1);
18 m=size(B,2);
19 q=size(C,1);

```

```

20 p = size(L,1);
21
22 StopTime = '8000'; %time to run the simulation
23 OutputTimes = mat2str(0:8:8000); %points that should be
    saved
24
25 %initial condition
26 X0=zeros(2*p*n,1);
27 X0(1:2:p*n)=2*rand(p*n/2,1)-1;
28
29 xy = [cos(0:2*pi/size(L,1):2*pi-2*pi/size(L,1))', sin(0:2*pi
    /size(L,1):2*pi-2*pi/size(L,1))']; %coordinates of the
    nodes (on a circle)
30 Formation = reshape(xy', 1, numel(xy))';
31
32 %Compute the protocol
33 [N, eta, gamma] = RobustSyncOptimalVars(A, B, C, L);
34 [F G] = RobustSyncControl(A, B, C, L, N, eta, gamma);
35
36 %compute the added perturbations
37 pertsys = perturbations(p, m, q, eta);
38
39 %compute the overal network dynamics
40 I = eye(p);
41 SSA = [ kron( I, A), kron( I, B * F);
42         1/N * kron( L, G * C), kron( I, A - G * C) + kron(
            1/N * L, B * F)];
43 SSB = [zeros( p * size( G)) zeros( p * size( G)); kron( 1/N
    * L, G) kron( -1/N * L, G)];
44 SSC = [zeros( p * size( F)), kron( I, F)];
45
46 %build the simulink system:
47 sys = 'Model';
48 bdclose(sys)
49 new_system(sys) % Create the model
50 open_system(sys) % Open the model
51
52 %some constants defining the spacing in simulink
53 x = 50;
54 y = 50;
55 w = 50;
56 h = 50;
57 offset = 100;
58
59 %adding blocks and links
60 pos = [(x+offset*1) (y+offset*0) (x+offset*1)+w (y+offset*0)
    +h];
61 add_block('built-in/State-Space',[sys '/SS'],'Position',pos)
    ;

```



```

62
63 pos = [(x+offset*2) (y+offset*0) (x+offset*2)+w (y+offset*0)
        +h];
64 add_block('built-in/Gain',[sys '/Xstate'],'Position',pos);
65
66 pos = [(x+offset*2) (y+offset*1) (x+offset*2)+w (y+offset*1)
        +h];
67 add_block('built-in/Gain',[sys '/Z'],'Position',pos);
68
69 pos = [(x+offset*1) (y+offset*1) (x+offset*1)+w (y+offset*1)
        +h];
70 add_block('built-in/State-Space',[sys '/Perturbation'],'
        Position',pos);
71
72 pos = [(x+offset*3) (y+offset*0) (x+offset*3)+w (y+offset*0)
        +h];
73 add_block('built-in/Gain',[sys '/Cx'],'Position',pos);
74
75 pos = [(x+offset*3) (y+offset*1) (x+offset*3)+w (y+offset*1)
        +h];
76 add_block('built-in/Outport',[sys '/Out1'],'Position',pos);
77
78 pos = [(x+offset*0) (y+offset*1) (x+offset*0)+w (y+offset*1)
        +h];
79 add_block('built-in/Constant',[sys '/Formation'],'Position',
        pos);
80
81 pos = [(x+offset*0) (y+offset*0) (x+offset*0)+w (y+offset*0)
        +h];
82 add_block('built-in/Mux',[sys '/SSinput'],'Position',pos);
83
84 add_line(sys,'SS/1','Xstate/1','autorouting','on');
85 add_line(sys,'SS/1','Z/1','autorouting','on');
86 add_line(sys,'Z/1','Perturbation/1','autorouting','on');
87 add_line(sys,'Perturbation/1','SSinput/1','autorouting','on'
        );
88 add_line(sys,'Xstate/1','Cx/1','autorouting','on');
89 add_line(sys,'Cx/1','Out1/1','autorouting','on');
90 add_line(sys,'SSinput/1','SS/1','autorouting','on');
91 add_line(sys,'Formation/1','SSinput/2','autorouting','on');
92
93 %adding parameters
94 set_param([sys '/SS'],'A', mat2str(SSA), 'B', mat2str(SSB),
        'C', mat2str(eye(size(SSA))), 'D', mat2str(zeros(size(
        SSB))), 'X0', mat2str(X0));
95 set_param([sys '/Z'],'Multiplication', 'Matrix(K*u)', 'Gain
        ', mat2str(SSC));
96 set_param([sys '/Perturbation'],'A', mat2str(pertsys.A), 'B
        ', mat2str(pertsys.B), 'C', mat2str(pertsys.C), 'D',

```

```

        mat2str(pertsys.D), 'X0', mat2str(zeros(size(pertsys.A
        ,1),1)));
97 set_param([sys '/Xstate'], 'Multiplication', 'Matrix(K*u)',
    'Gain', mat2str([eye(size(A)*p), zeros(size(A)*p)]));
98 set_param([sys '/Cx'], 'Multiplication', 'Matrix(K*u)', '
    Gain', mat2str(kron(I,C)));
99 set_param([sys '/SSinput'], 'Inputs', '2');
100 set_param([sys '/Formation'], 'Value', mat2str(Formation));
101 set_param(sys, 'StopTime', StopTime, 'OutputOption', '
    SpecifiedOutputTimes', 'OutputTimes', OutputTimes);
102
103 %run the simulation
104 sim(sys);

```

../MATLAB/animateFormationControlFancy.m

```

1 %creates an animation from the output of
  demoFormationControl
2
3 %plotting the output:
4
5 width=800;
6 height=480;
7
8 clear M
9
10 fig=figure;
11 set(fig, 'Position', [0 0 width+1 height+1]);
12
13 %path plot size
14 pathmaxx=max(max(yout(:,1:2:end)))*1.05;
15 pathminx=min(min(yout(:,1:2:end)))*1.05;
16 pathmaxy=max(max(yout(:,2:2:end)))*1.05;
17 pathminy=min(min(yout(:,2:2:end)))*1.05;
18
19 pathaxishandle=axes('position', [0.05 0.05 0.5 0.9]);
20 currentformationaxishandle=axes('position', [0.65 0.55 0.3
    0.4]);
21 targetformationaxishandle=axes('position', [0.65 0.05 0.3
    0.4]);
22
23
24 axes(targetformationaxishandle);
25 gplot(Graph, xy);
26 hold on;
27 scatter(xy(:,1), xy(:,2));
28 hold off;
29 for j = 1:p,
30     text(xy(j,1), xy(j,2), [' ' int2str(j)], 'FontSize',12);
31 end

```

```

32 title('Graph topology/Target formation');
33 grid on;
34
35 for k=1:size(tout,1)
36     axes(pathaxishandle);
37     plot(yout([1 1:k],1:2:end),yout([1 1:k],2:2:end));
38     hold on;
39     scatter(yout(k,1:2:end),yout(k,2:2:end));
40     hold off;
41     set(pathaxishandle,'xlim',[pathminx pathmaxx], 'ylim',[
        pathminy pathmaxy]);
42     grid on;
43     title(['Time: ' num2str(round(tout(k)))]);
44     for j = 1:p,
45         text(yout(k,2*j-1),yout(k,2*j),[' ' int2str(j)],'
            FontSize',14);
46     end
47
48     axes(currentformationaxishandle);
49     gplot(Graph, vec2mat(yout(k,:),2));
50     hold on;
51     scatter(yout(k,1:2:end), yout(k,2:2:end));
52     hold off;
53     title('Graph topology/Current formation');
54     for j = 1:p,
55         text(yout(k,2*j-1),yout(k,2*j),[' ' int2str(j)],'
            FontSize',12);
56     end
57     grid on;
58
59     M(k) = getframe(gcf);
60 end

```

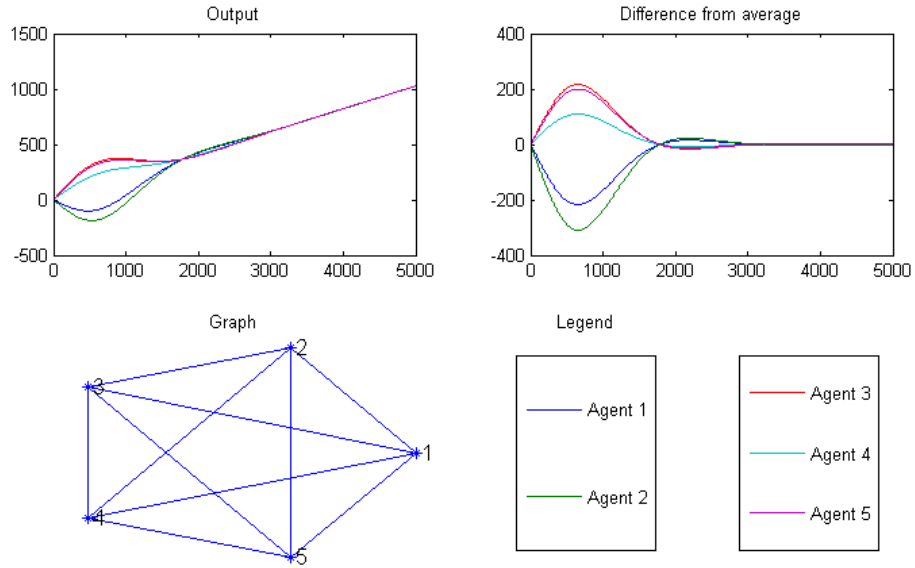
B Results

B.1 Additional results for the synchronisation of the double integrator

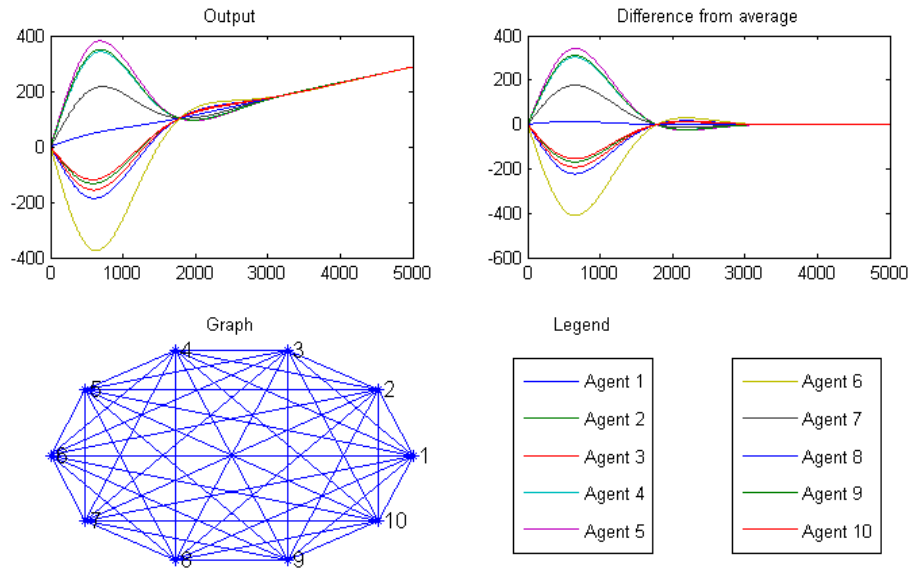
Additional results for the system

$$\dot{x} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} x + \begin{pmatrix} 0 \\ 1 \end{pmatrix} u, \quad y = \begin{pmatrix} 1 & 0 \end{pmatrix} x$$

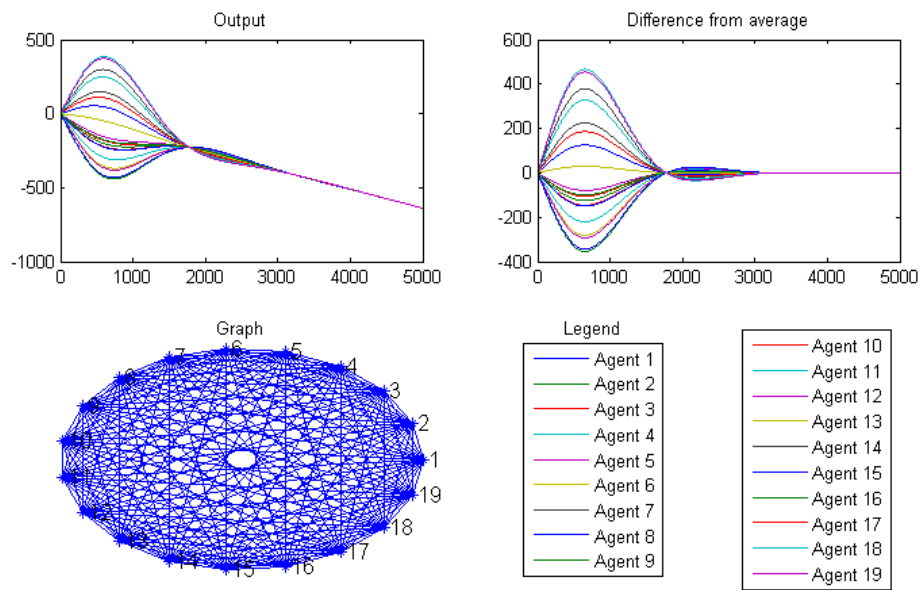
Full graph with 5 nodes:



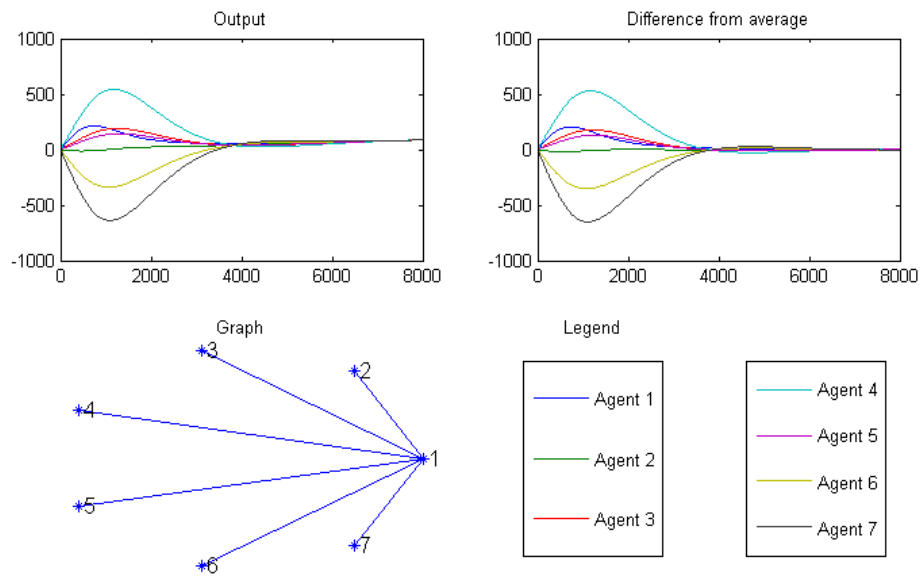
Full graph with 10 nodes:



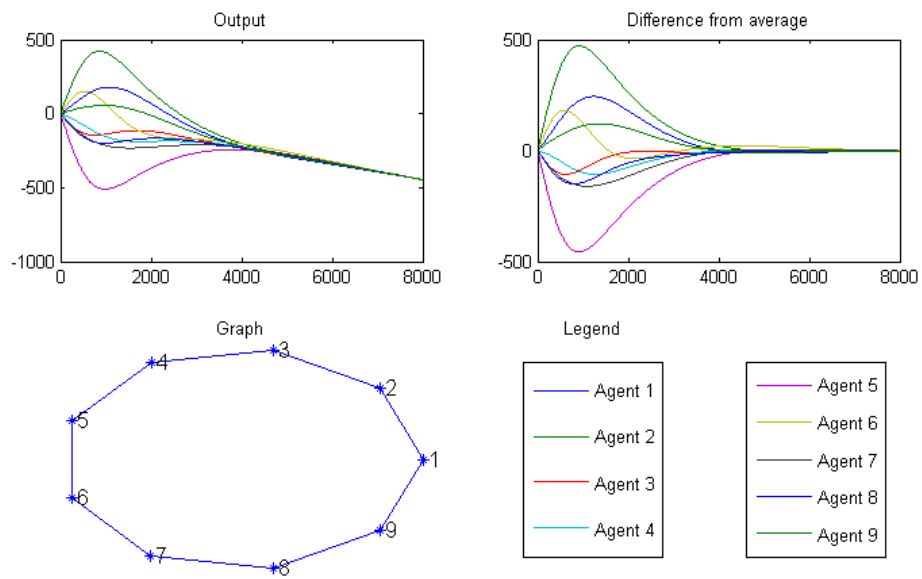
Full graph with 19 nodes



Star graph with 7 nodes



Circle graph with 9 nodes



B.2 Additional snapshots of the formation control animation

