Analysis on distributed observers

Simulating the distributed Luenberger observer

R.H. Snijders

A thesis presented for the degree of
General mathematics Bachelor

FSE
Rijksuniversiteit Groningen
Netherlands
07-02-2023

Contents

Abstract

This paper examines the distributed Luenberger observer. This distributed observer arises when an output matrix is divided into smaller parts and observers are constructed for each of them individually with the system matrix. A difficult case arises when some or all of the smaller parts are not observable. When these observers are coupled however, it has been shown that the distributed observer converges to the overall state of the original system.

This study constructed a MATLAB code simulating the distributed Luenberger observer with $1 \times n$ agents. After analyzing the results, the study found that the distributed Luenberger observer is particularly suited to non observable agents, with a low amount of communication between them. Furthermore, the distributed Luenberger observer is quite receptive to funnel control.

1 Introduction

Ever since people have been around, they have looked for ways to control the systems around them. The first step to controlling systems, is of course the gathering of information. Even when one can measure the effect that they themselves introduce in the system as well as what the system produces, there is still something unknown happening inside. The need for seeing what was going on within systems then spawned the search for better and better state observers. The introduction of the Luenberger observer was a major advancement in this regard.

Observer design is an important field in systems theory. It has many applications in stabilization of systems and control theory. Often in systems, the input and output are known by people working with them. A challenge is to find the so called state of the system, which is what state observers are designed to do.

In today's society there is an increasing demand for decentralized systems. Take as an example the avenue of crypto currency, a currency that gained massive popularity thanks to the decentralized nature of it. Collaborations of smaller parts of groups are becoming more and more prevalent, preferred over an overarching structure that organizes everything.

To this end, there have been recent developments to create decentralized observers for system, where there is not a single observer that oversees the entire system. Rather, the system is divided up into smaller pieces that communicate with each other. The pieces themselves do not have full knowledge of the system, yet together they can help each other to determine the overall state of the system.

In this paper I will discuss some of the methods to construct distributed observers. To this end, I will first go into some systems notation, some graph theory, as well as the basic Luenberger observer. I will discuss multiple approaches to a distributed observer, but the main focus is a version based on the Luenberger observer. Lastly, I will review some results from simulations.

2 Methodology

This paper aims to research the distributed Luenberger observer. In particular it aims to answer the question if funnel control could be implemented. In order to do this the dynamics of the different agents of the distributed observer have been programmed into MATLAB. The choice has fallen on MATLAB as it is an environment with many built in features that can be used easily to these ends.

The research question of this paper is: "What are the different properties of the distributed Luenberger observer?" This question has been divided into the smaller questions:

1. "What are the different effects of different coupling gains, and different pole placements of $A - L_iC_i$ matrices?"

2. "Does a more connected graph lead to a faster convergence for the distributed Luenberger observer?"
3. "Is coupling observable agents beneficial to the convergence of the distributed Luenberger observer?"
4. "Can the distributed Luenberger observer benefit from the usage of simple funnel control?"

3 Literature review

A simple approach of constructing a distributed Luenberger observer comes from Han, Trentelman et al[6]. In this paper they discuss a method for obtaining the necessary tools to construct the observer. This paper serves as an investigation of this method, but other successful methods exist and are highlighted in this section.

Silm et al. [12] have investigated a way to make a distributed observer that does not only converge asymptotically, but converges in finite time. Just like [6] they use the assumption that the overall system is observable, and that the agents share information according to a strongly connected communication graph. The major advancement here is of course that on top of having a guarantee of convergence in infinite time, here we have a guarantee of convergence in finite time. Such an observer can be more useful in applications.

They start off with a similar approach to the one done by [6]. They introduce transformation matrices to divide agents into observable and unobservable parts as in (13). They furthermore split the T matrix in the observable and unobservable parts $T_i = \begin{bmatrix} T_{io} & T_{iu} \end{bmatrix}$. After computing the Luenberger matrices L_{io} they give the following dynamics of the distributed observer:

$$\dot{\tilde{x_i}} = A\tilde{x_i} - T_{io}L_{io}(C_i\tilde{x_i} - y_i) - \gamma_i T_{iu}T_{iu}^T \sum_{j\in\mathcal{N}_i} (\tilde{x_i} - \tilde{x_j}) \tag{1}$$

Where the γ_i are the coupling gains. Clearly, this is a far simpler way of constructing the distributed observer. The paper uses this as the beginning point to find a nonlinear version that guarantees the finite time convergence property.

Rego et al. [10] have investigated a way of computing a distributed observer of a discrete time system with a noise component. For a system of the form

$$\begin{cases} x(t+1) = Ax(t) + w(t) \\ y(t+1) = Cx(t) + v(t) \end{cases} \tag{2}$$

With similar assumptions they create a distributed observer of the following form:

$$\hat{x_i}(t+1) = A\Omega_i^{-1}(\sum_{j\in\mathcal{N}} \pi_{i,j}\bar{\Omega_i}\hat{x_j} + C_i^T R_i^{-1} y_i) \tag{3}$$

Here Ω_i, $\bar{\Omega_i}$ and R_i are appropriately chosen positive definite matrices.

They use the same assumptions as in the other approaches, with the added requirement that the dynamics matrix of the system is invertible. Dealing with noise has been a struggle for the likes of the Luenberger observer, so advancements like these could prove to be pretty significant.

4 Background information

4.1 Mathematical systems

In mathematics a system is given as the following set of equations[1]:

$$\begin{cases} \dot{x}(t) = Ax(t) + Bu(t) \\ y(t) = Cx(t) + Du(t) \end{cases} . \tag{4}$$

Here the output of the system is $y \in \mathbb{R}^p$, the input of the system is $u \in \mathbb{R}^m$ and the state of the system is given as $x \in \mathbb{R}^n$. $\dot{x}$ is the vector that contains the derivative values of the entries of x, which is dependent (usually) on x, the state and u, the input. The remaining terms are matrices. $A \in \mathbb{R}^{n\times n}$ and $B \in \mathbb{R}^{m\times n}$ are usually referred to as the system matrix and the input matrix. $C \in \mathbb{R}^{p\times n}$ and $D \in \mathbb{R}^{p\times m}$ are commonly called the output matrix and the feedthrough matrix respectively.

Take as an example the following system:

$$\begin{cases} \dot{x}(t) = \begin{bmatrix} 0 & 1 \\ -2 & 3 \end{bmatrix} x(t) + \begin{bmatrix} 1 \\ 1 \end{bmatrix} u(t) \\ y(t) = \begin{bmatrix} 1 & 1 \end{bmatrix} x(t) + 2u(t) \end{cases} \tag{5}$$

By filling in the starting state, or initial condition, x_0 and a specific input one can see what happens to the state as time starts running. If we take $x_0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ and an input of $u(t) = \sin(t)$,we can see that $\dot{x}$ at time 0 is given as $\dot{x}(0) = \begin{bmatrix} 0 & 1 \\ -2 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} \sin(0) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$. Hence we know that the state has those derivative values. This means that the derivative value of the first and second entry of x is 1, which allows us to roughly see how the entries of the state x will change. Furthermore, we can see that the output is given as $y(0) = \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} + 2\sin(0) = 2$.

In this project we are most interested in the effect of something called observability. Before we get to the concept of observability, we must first state what distinguishes states:

Definition 1. Distinguishability
An input $u(t) \in \mathbb{R}^m$ on the interval $[t_1, t_2]$, $t_1, t_2 \geq t_0$ distinguishes between $x(t_0) = x_0$ and $x(t_0) = x_0'$ at time t_2 if they produce different outputs at time

t_2.
Furthermore, the initial states (x_0, t_0) and (x_0', t_0) are distinguishable on the interval $[t_1, t_2]$, if there exist a time $t \in [t_1, t_2]$ and an input $u(t)$ on the interval $[t_1, t]$ that distinguishes between x_0 and x_0' at time t. [8]

For the majority of this paper, we consider systems with no input matrices and no feedthrough matrices (so we have $B, D = 0$). Systems of that format hence have the following equation system:

$$\Omega : \begin{cases} \dot{x}(t) = Ax(t) \\ y(t) = Cx(t) \end{cases} \tag{6}$$

Systems of this format with initial condition x_0 have the following solution:

$$y(t; x_0, 0) = Ce^{At}x_0 \tag{7}$$

Since these systems no longer have an input, the definition of distinguishability changes slightly. One can now distinguish between two initial conditions simply if there exists a time where they produce different outputs.

With the definition of distinguishability, we can define observability for these systems as follows:

Definition 2. Observability
The system (A,C) is called observable on the interval $[0, T]$, if any two states $x_0, x_0' \in \mathbb{R}^n$ are indistinguishable on $[0, T]$ only if $x_0 = x_0'$ [1]

This means that any two distinct initial conditions will have different output trajectories. Hence, if a system (A,C) is observable, one can in some way determine the initial condition based on the trajectory of the output, as there only is exactly one possible initial condition that could have provided that trajectory. Take as an example the following system matrix, output matrix and these two initial conditions:

$$A = \begin{bmatrix} -4 & 2 & 2 \\ 1 & 2 & -3 \\ -3 & 3 & 0 \end{bmatrix}, C = \begin{bmatrix} -3 & 2 & 1 \\ -2 & 3 & -1 \end{bmatrix}, x_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \text{ and } x_0' = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

We have $Ax_0 = Ax_0' = \mathbf{0}$ and $Cx_0 = Cx_0' = \mathbf{0}$. Since the derivative of the state has reached zero this means that the state will no longer change. Furthermore, both of these initial conditions have the same output and since the state will not change they will have the same output trajectory. Thus, we cannot distinguish between these two states, yet they are not equal. We conclude that this is an example of an unobservable system. This example serves to show that a zero output trajectory does not imply that the trajectory of the state is equal to zero. Interestingly, the line spanned by ax_0', $a \in \mathbb{R}$ satisfies $Aax_0' = aAx_0' = a\mathbf{0} = \mathbf{0}$ and similarly, $Cax_0' = aCx_0' = a\mathbf{0} = \mathbf{0}$. This means that it is impossible to distinguish between any two points on that line, as they all have the same output trajectory of zero.

Suppose now that we have a system where there are m such lines with the property that any points on them share the same output trajectory of zero. Any combination of the points on the lines will then satisfy $A(a_1x'_{0,1} + ...a_mx'_{0,m}) = Aa_1x'_{0,1} + ... + Aa_mx'_{0,m} = \mathbf{0}$. Hence, these combinations of the points on the lines also share the output trajectory of zero. We refer to the space spanned by these lines as the unobservable space.

If a system is not observable, then there exists an unobservable subspace with the property of that if any state trajectory finds itself in it, it is not possible to determine precisely the initial conditions of the state. A way to determine the unobservable subspace is by constructing something called the observer matrix. The observer matrix is defined as follows for our (A,C) type systems:

$$\begin{bmatrix} C \\ CA \\ \vdots \\ CA^{n-1} \end{bmatrix} \tag{8}$$

From this matrix, we can determine if systems are observable, and if not, how unobservable they are. This we can do thanks to the following theorem:

Theorem 1. *The unobservable subspace N_T is independent of $T > 0$, and is equal to the null space of the observability matrix. [1]*

Since the unobservable subspace is precisely equal to the null space of the observer matrix, we can see by the size of the null space just how unobservable the system is. This also means that, since a matrix A with a full rank only has trivial solutions to $Ax = 0$, a system is precisely observable if it has an observer matrix with full rank.

4.2 Observers

Often in a physical system, the input is known, since it usually is controlled. Furthermore, one can see the output, since it is the product of the system. What often is not known however is the state of the system. To get this information, one could use what is called an observer. A mathematical observer is defined as the following system with one additional property:

Definition 3. Observer

$$\Omega : \begin{cases} \dot{w}(t) = Pw(t) + Qu(t) + Ry(t) \\ \xi(t) = Sw(t) \end{cases} \tag{9}$$

A system Ω as written above is called a state observer for Σ if, for any pair of initial conditions x_0, w_0 satisfying $e(0) = Sw_0 - x_0 = 0$ and any input function $u(t)$, we have that $e(t) = 0$ for all $t \geq 0$. [1]

An observer is a whole new system, where the output ξ is an approximation of the state of the initial system. The state $w(t)$ of this new system is dependent on itself and the input and output of our original system. The main idea of observers is that one takes a guess, namely $Sw_0 = \xi_0$, at what the state is so that the observer can improve on the precision of the guess. For such a system to be called an observer, it must satisfy that if the initial guess happens to be exactly correct that for the rest of time the observer follows the trajectory of the original system.

If we however want an observer that gives us useful information regardless of where we place our original guess, we must take a look at stable observers. These are defined as follows:

Definition 4. Stable observer
A state observer Ω is called stable if for each pair of initial values w_0, x_0 and any input function $u(t)$, we have that $\lim_{t\to\infty} e(t) = 0$ [1]

Stable observers have the property that for any initial guess that one makes, as time increases the disparity between the guessed state and the original state will decrease to zero. Hence, the system will converge to the state of our original system for any initial guess.

4.3 Luenberger observer

The Luenberger observer for a system of format (A,B,C) arises when one introduces a matrix $L \in \mathbb{R}^{n\times p}$. The observer is constructed as follows [7]: We take $P = A - LC, Q = B, R = L$ and $S = I_n$. This yields the system:

$$\Omega : \begin{cases} \dot{w} = (A - LC)w(t) + Bu(t) + Ly(t) \\ \xi(t) = w(t) \end{cases}$$

We can simplify this a bit further, and introduce some new notation. In most literature, the Luenberger observer is referred to as having state $\hat{x}(t)$ rather than $w(t)$. This gives us the following equation for the Luenberger observer dynamics:

$$\dot{\hat{x}}(t) = A\hat{x}(t) + Bu(t) + L(y(t) - C\hat{x}(t)) \tag{10}$$

One can see that if the initial guess happens to be exactly correct, the equation reverts back to the original system, or indeed [3]:

$$\begin{aligned} &\dot{\hat{x}}(0) = A\hat{x}(0) + Bu(0) + L(y(0) - C\hat{x}(0)) \\ &= Ax(0) + Bu(0) + L(y(0) - Cx(0)) \\ &= Ax(0) + Bu(0) + L(y(0) - y(0)) \\ &= Ax(0) + B(0) = \dot{x}(0) \end{aligned}$$

Furthermore, when the L matrix is taken precisely so that the eigenvalues of $A - LC$ are in the left half complex plane, we obtain a stable observer. One can see this when we consider the error dynamics of the observer:

$$
\begin{aligned}
&e = \hat{x} - x\\
&\dot{e} = \dot{\hat{x}} - \dot{x} = A\hat{x} + Bu + L(y - C\hat{x}) - (Ax + Bu)\\
&= A\hat{x} - Ax + L(Cx - C\hat{x})\\
&= Ae - LCe = (A - LC)e
\end{aligned}
$$

We can see that the error behaves in accordance to the matrix $A - LC$, which we have taken to have eigenvalues in the left half plane. Matrices with such a property are called stable and will make any state converge to zero over time. Hence we have that the error will converge to zero as time goes to infinity. In particular, one can see that if the initial guess was correct and thus the error at time t_0 was exactly zero then the error will not change, since the derivative $\dot{e}$ will remain zero.

We conclude that the Luenberger observer will follow the trajectory of our initial system when the initial guess is exactly correct. The starting point and the trajectory will be the same in the case of a perfect initial guess, and the error always converges to zero, regardless of the initial guess. This means that the Luenberger observer is a stable observer by the earlier defined definition of observers.

Remark. A short proof that stable matrices have states converge to zero: *An $n\times n$ matrix will have n linearly independent eigenvectors each with its own (not necessarily unique) eigenvalue. This means that the equation $\dot{x} = Ax$ can be written as $\dot{x} = A(\alpha_1 v_1 + \cdots + \alpha_n v_n)$, where all v_i are eigenvectors. Let λ_i be the corresponding eigenvalues for the eigenvectors v_i Since all those vectors are eigenvectors, we can rewrite the expression as: $\dot{x} = \lambda_1\alpha_1 v_1 + \ldots \lambda_n \alpha_n v_n$. Here we can use the fact that all the eigenvalues are in the left half complex plane. It should be noted that writing x in terms of the eigenvalues is essentially rewriting x in a different basis. If any α_i is positive, it will decrease the quantity of v_i over time, since the λ_i is negative. Vice versa, if α is negative, the quantity of v_i will increase. Hence, the quantity of all of the eigenvectors will decrease in x, which means that x will tend to zero.*

4.4 Basic graph theory

A graph is formally defined as a set of vertices and edges, or in other words, $\mathcal{G} = (V, E)$. Here V is the set of all the vertices, and E is the set of all the vertices that are connected to each other. If for example vertices A an B are connected, then E will contain the element $\{A, B\}$. As an example, take Figure 1. Each circle with a number represents a vertex, where the connections between them represent an edge between the vertices. This graph consists of the following sets:

$$
\begin{aligned}
&V = \{1, 2, 3, 4, 5\}\\
&E = \{\{1, 2\}, \{1, 3\}\{2, 3\}, \{2, 5\}\}
\end{aligned}
$$

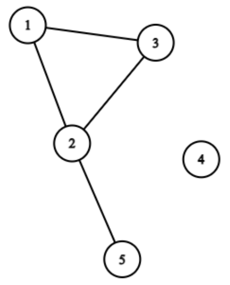


Figure 1: Example of a graph

You can see that E contains the element $\{1,2\}$, and as we can see in the graph indeed 1 and 2 share an edge. Since vertex 4 does not share any edges with any other vertex, we can see that it does not appear in E, but it is still included in V. Two vertices are referred to as being adjacent if they share an edge and independent if they do not. We denote the set of all adjacent vertices of vertex i as $\mathcal{N}_i$.

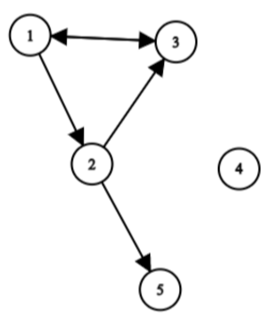


Figure 2: Example of a directed graph

A graph is called directed if every edge it has has a direction. If there is an edge from A to B in the direction of B we now write it as the ordered pair (A,B). Rather than saying that A and B are adjacent to each other, we now say that A is adjacent to B, but B is not adjacent to A. The sets corresponding to the graph in Figure 2 are:

$$V = \{1,2,3,4,5\}$$
$$E = \{\{1,2\},\{1,3\},\{2,3\},\{2,5\}\{3,1\}\}$$

We can see that now the ordering becomes very important. Furthermore, while the edge between vertex 1 and 2 is only contained once, the edge between vertex 1 and vertex 3 is contained twice. Once to show that vertex 1 is adjacent to vertex 3, and then again to show that vertex 3 is also adjacent to vertex 1.

In graphs, one can "travel" from one vertex to another adjacent one. A path is defined as the route that can be traveled from one vertex to a certain other vertex, with the added property that no vertices can be repeated (with the exception of the first and last vertex). One writes down a path by listing all the vertices that are on the route. Two examples of paths in the graph of Figure 1 from vertex 3 to vertex 2 can be constructed as $3,1,2$ and $3,1,2,3,2$. This is however not a path in the graph of Figure 2, as there one can not travel from vertex 3 to 2. Two examples of paths in the graph of Figure 2 starting at 1 and ending at 5 are written as $1,3,1,2,5$ and $1,2,5$.

With the concept of directed graphs and paths, comes the final property in graph theory that needs to be discussed in this research, namely connectedness. An undirected graph is called connected if one can construct a path from any vertex to any other vertex. In directed graphs, we distinguish between strongly and weakly connected graphs. A graph is a weakly connected directed graph if one obtains a connected graph when one removes the directed property of the edges. A graph is strongly connected if one can construct a path from any vertex to any other vertex even with the directed property of the edges.

In the graph corresponding to the leftmost illustration in Figure 4.4 we can easily construct paths from any of the vertices to any of the other vertices, hence it is a connected graph. If we remove the directed property of the edges of the graph corresponding to the middle illustration of Figure 4.4 we obtain the graph corresponding with the leftmost illustration. Since we already have seen that

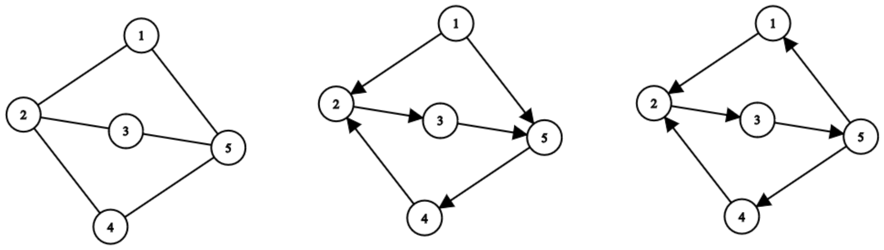


Figure 3: Examples of a connected graph, a weakly (but not strongly) connected directed graph and a strongly connected directed graph respectively

the graph was connected, we can conclude that the graph corresponding to the middle illustration is weakly connected. It is however not strongly connected, as we can see by means of a counterexample: We can not construct a path between vertex 5 and vertex 1. The graph belonging to the rightmost illustration however is strongly connected, as it now is possible to construct paths between any of the vertices.

4.5 Distributed observer

Rather than constructing an observer for the system (A, C), we can divide this system up into different smaller systems. We can split our original C matrix into smaller parts as follows: $C = \begin{bmatrix} C_1 \\ \vdots \\ C_N \end{bmatrix}$. This gives rise to a collection of N new systems (A, C_i). We define $\mathcal{N} = \{1, \dots, N\}$ as the set containing all the agents.

$$\Sigma_i : \begin{cases} \dot{x_i} = Ax_i \\ y_i(t) = C_i x \end{cases} \tag{11}$$

It is important to note that if the overall system (A,C) is observable, it does not provide any guarantees on whether the individual (A, C_i) are observable too. With that in mind, the idea behind the distributed observer is to create an observer split into N parts out of all these separate systems, which we will refer to as agents.
If all of the agents were observable, we could construct a regular Luenberger observer for them and we could find the original state of the system. However, when some or all of the agents are unobservable, something else must be done. If we want to ensure convergence to the state, the agents have to share information with some or all of the other agents. The

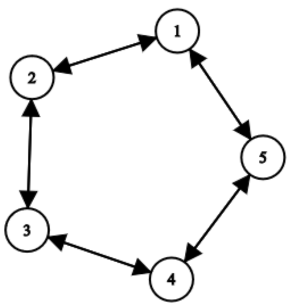


Figure 4: Communication schematics

way that the agents share their information is written in a communication graph G. An example of such a graph is given in Figure 4. In this example we have that each agent both shares and receives information from their adjacent numbers. For example, agent 2 receives information from agents 1 and 3, and shares information with them as well. This graph is represented in the following Adjacency matrix:

$$\mathcal{A} = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

As you can see, since agent 1 receives information from agent 2, in the matrix that represents the graph there is a 1 on the first row, second column, but agent 3 does not share their information with agent 5, hence the 5th row, 3rd column is a 0 in the matrix.

We define the diagonal weight matrix $\mathcal{D}$ as the matrix with on its diagonal entry d_i the total weight of all the vertices that share the information with agent i. Hence, the diagonal weight matrix of the graph represented in Figure 4 is given as:

$$\mathcal{D} = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 2 \end{bmatrix}$$

Lastly, we introduce something called a Laplacian matrix Λ. It is defined as $\Lambda = \mathcal{D} - \mathcal{A}$. Thus, the Laplacian matrix of the graph represented in Figure 4 is given as follows:

$$\Lambda = \begin{bmatrix} 2 & -1 & 0 & 0 & -1 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ -1 & 0 & 0 & -1 & 2 \end{bmatrix}$$

A way of constructing a stable distributed observer arrises when one expands on the Luenberger observer. Besides the L matrix that we have seen in the Luenberger observer, we introduce a coupling gain γ, gain matrices M_i and the information sets for each of the individual agents $\mathcal{N}_i$, consisting of the agents that share their information with agent i. We construct the dynamics of each of the individual agents as follows:

$$\dot{\hat{x}}_i = A\hat{x}_i + L_i(y_i - C\hat{x}_i) + \gamma M_i \sum_{j \in \mathcal{N}_i} (\hat{x}_j - \hat{x}_i) \tag{12}$$

As you can see, the first part of the equation is essentially the regular Luenberger observer. The second part of the equation corresponds to the information sharing that happens between the agents. When the L matrices and M matrices are constructed appropriately, we will see that the agents all converge to each other and the state. These M_i and L_i matrices and the coupling gain in this research are computed using the following crucial theorem [6]:

Theorem 2. Computation of the required matrices for the distributed Luenberger observer: *Let $\alpha > 0$. If (A, C) is observable and $\mathcal{G}$ is a strongly connected directed graph with Λ as the Laplacian matrix representation, then there exists a distributed observer that achieves omniscience asymptotically while all solutions of the error system converge to zero with decay rate at least α. Such an observer is obtained as follows.*

1. *For each $i \in \mathcal{N}$, choose an orthogonal matrix T_i such that*

$$T_i^T A T_i = \begin{bmatrix} A_{io} & 0 \\ A_{ir} & A_{iu} \end{bmatrix}, C_i T_i = \begin{bmatrix} C_{io} & 0 \end{bmatrix} \tag{13}$$

   Where (A_{io}, C_{io}) are observable

2. *Compute the positive row vector $r = [r_1, ..., r_N]$ such that $r\Lambda = 0$ and $r\mathbb{1}_N = N$*

3. *Put $g_i = 1, i \in \mathcal{N}$ and take $\epsilon > 0$ such that property (17) holds.*

4. *Take $\gamma > 0$ sufficiently large so that for all $i \in \mathcal{N}$*

$$A_{iu} + A_{iu}^T - (\frac{\gamma}{r_i}\epsilon - 2\alpha)I_{n-v_i}) + \frac{1}{\frac{\gamma}{r_i}\epsilon - 2\alpha} A_{ir} A_{ir}^T < 0 \tag{14}$$

5. *Choose L_{io} such that all eigenvalues of $A_{io} L_{io} C_{io}$ lie in the region $\{s \in C | Re(s) < -\alpha\}$*

6. *For all $i \in \mathcal{N}$, solve the Lyapunov equation*

$$P_{io} S_i + S_i^T P_{io}^T + (\gamma - 2\alpha) I_{v_i} = 0 \tag{15}$$

   Where $S_i = A_{io} - L_{io} C_{io} + \alpha I_{v_i}$, and $P_{io} > 0$.

7. *Define*

$$L_i = \begin{bmatrix} L_{io} \\ 0 \end{bmatrix}, M_i = T_i \begin{bmatrix} P_{io}^{-1} & 0 \\ 0 & I_{n-v_i} \end{bmatrix} T_i^T, i \in \mathcal{N} \tag{16}$$

In step 3, the property referenced is the following lemma:

Lemma 1. *Let Λ be the Laplacian matrix associated with the strongly connected directed graph G. For all $g_i > 0$, $i \in \mathcal{N}$, there exists an $\epsilon > 0$ such that:*

$$T^T (R\Lambda + \Lambda^T R) \otimes I_n T + G > \epsilon I_{nN} \tag{17}$$

Where $\otimes$ symbolizes the Kronecker product. Here T is a $n^2 \times n^2$ matrix of zeros with the transformation matrices T_i on the diagonal. R is the $n \times n$ matrix with the entries of r on its diagonal and G is a special matrix with on its diagonal the matrices G_i, defined as $G_i = \begin{bmatrix} g_i I_{v_i} & 0 \\ 0 & 0_{n-v_i} \end{bmatrix}$, where $i \in \mathcal{N}$

This method has been proposed by Han, Trentelman et al. [6] and proven to create a stable distributed observer.

4.6 Funnel control in basic systems

Often in systems with an input the goal is to design inputs such that a certain output is achieved. The input that satisfies this is called a feedback law. Take as an example the following one by one system:

$$\begin{cases} \dot{x} = bu \\ y = cx \end{cases} \tag{18}$$

If one wanted to design a feedback law that returned an output y to zero, one introduces a reference signal $y_{ref} = 0$ to obtain an error $e = y(t) - y_{ref} = y(t)$. The goal is of course to design an input that minimizes the error. In this case, a good feedback law is given as $u(t) = -\frac{y(t)}{bc}$. One sees this when computing the derivative of the error.

$$\dot{e}(t) = \dot{y}(t) = c\dot{x} = cbu(t) = -y(t) = -e(t)$$

Such a derivative equation will converge to 0, and hence so will the error in this case. This means that this indeed is a successful feedback law.

Suppose now that we have a system of the form (4), where $D = 0$, and suppose that we want to design an input u(t) such that the output converges to a certain value y_{ref}. We define the error again as the difference between our current y value and the reference y, or $e = y - y_{ref}$. A way to control such errors in basic systems is via the concept of funnel control. The idea is to restrain errors to the inside of a funnel shaped area defined by a function $\varphi(t)$ to ensure that with time the error decreases as the funnel gets narrower and narrower. This $\varphi(t)$ function has to satisfy the following conditions [11]:

$\varphi \in \Phi_k = \{\varphi \in C^k(\mathbb{R}_{\geq 0} \to \mathbb{R}) | \varphi, \dot{\varphi}, \ldots, \varphi^k$ are bounded and $\varphi(\tau) > 0$ for all $\tau > 0$, and $\liminf_{\tau \to \infty} \varphi(\tau) > 0\}$

When φ is chosen in such a manner then the funnel area is given by:

$$\mathcal{F}^p_\varphi = \{(t, e) \in \mathbb{R}_{\geq 0} \times \mathbb{R}^p | \varphi(t) \|e\| < 1\} \tag{19}$$

Such a funnel area has the upper bound given by $\frac{1}{\varphi}$, and the lower bound given by $-\frac{1}{\varphi}$. Furthermore, since the function φ has a limit inferior that is larger than zero there exists a value of the narrowest part of the funnel, ρ_∞.

With such a φ,and with the initial error lying somewhere in that boundary, there is a way to contain the entire trajectory of the error within the funnel area. The main idea is to introduce a feedback law that increases in magnitude when the error is close to the funnel boundary and decreases when the error is closer to zero. This means that if the error is large, the feedback will cause a larger change, hence bringing the error down. A basic way that that can be done is by introducing $\kappa(t)$ as follows [11]:

$$\kappa(t) = \frac{1}{1 - \varphi(t)^2 \|e(t)\|^2}, \text{ and } u(t) = -\kappa(t)e(t) \tag{20}$$

A funnel control of this kind will usually only work on systems of relative degree one, where it works off of the high-gain property of such systems. The high gain property is the property that a system stabilizes with a gain chosen arbitrarily high (enough). As we can see however in the computation of the gamma, our distributed observer satisfies this property, as gamma must simply be selected large enough, with no restrictions.

An example of how funnel control works is provided by Han et al. [5] in Figure 5. It showcases the funnel with its bounds as well as the error staying within the bounds of the funnel.

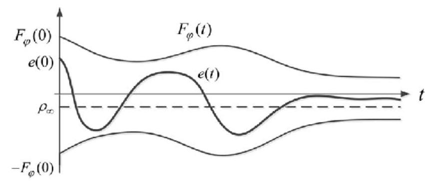


Figure 5: Funnel control concept

4.7 Ackermann's formula

For systems with a nonzero input matrix, the controllability matrix is defined as follows:

$$\begin{bmatrix} B & AB & \dots & A^{n-1}B \end{bmatrix}$$

This matrix is used in Ackermann's formula, another very important tool for the construction of the distributed Luenberger observer. We use Ackermann's formula in our computation of the L_i matrices. The formulation of the formula is as follows:

> Let $\dot{x}(t) = Ax(t) + Bu(t)$ be a linear time invariant dynamical system, with $x(t), B \in \mathbb{R}^n$ and $A \in \mathbb{R}^{n \times n}$. If the observer matrix is invertible, then the matrix $\hat{A} = A - BK^T$ has the user-defined eigenvalues $\{\lambda_1, \dots, \lambda_p\}$, with algebraic multiplicities $q_1, \dots, q_p$, where $K = (\prod_{i=1}^{p}(A - \lambda_i I)^{q_i})(Co(A,B)^{-1})^T \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$ (Co(A,B) denotes the controllability matrix)

We use the formula in the computation of the L matrices when we have a C matrix in $\mathbb{R}^{1 \times n}$. To do this however, we still need to alter it a little bit, since Ackermann's formula gives you a matrix K that lets $A - BK^T$ have all negative eigenvalues, where we need a matrix L that lets $A - LC$ have only negative eigenvalues. We fix this problem by using Ackermann's formula using A^T and $B = C^T$. When we use the formula now, we are given a K that lets $A^T - BK^T$ have negative real part eigenvalues. Since eigenvalues are conserved under transposition, we can see that $(A^T - BK^T)^T = A - KB^T = A - KC$ will have the same eigenvalues. Hence, using the formula in this fashion will provide us with the L matrices needed in the $C \in \mathbb{R}^{1\times n}$ case. In this research the choice was made to take the $C_i \in \mathbb{R}^{1\times n}$. Hence, in the case that (A, C_i) is observable, we can apply Ackermann's formula directly to find our L_i. However, when this is not the case it is slightly trickier. We must now apply the formula to (A_{io}, C_{io}), a system that we know is observable per design.

4.8 Brief overview of the Euler method

When one is presented with a differential equation in the format of $\frac{dy}{dt} = f(y(t), t)$ and an initial value $y(t_0) = y_0$, it is difficult to determine exactly how the trajectory looks of $y(t)$. A way of approximating the solution to this so called initial value problem was given by the mathematical titan Leonhard Euler himself. He developed what is now called the Euler method in 1769 [4]. The method is based on the definition of the derivative:

$$f(y(t), t) = \frac{dy}{dt} = \lim_{\Delta t \to 0} \frac{y(t + \Delta t) - y(t)}{\Delta t} \tag{21}$$

This means that for a small enough nonzero Δt we have that $\frac{y(t+\Delta t)-y(t)}{\Delta t} \approx f(y(t), t)$. Euler used this fact to approximate the solution of the initial value problem as follows:

$$y(t_0 + \Delta t) \approx y(t_0) + \Delta t f(y(t_0), t_0) \quad [2] \tag{22}$$

When one has computed the value of the $y(t_0 + \Delta t)$ it is possible to compute the value of $y(t_0 + \Delta t + \Delta t)$ by using equation (22) and plugging in $t_0 = t_0 + \Delta t$. Using this we can iteratively determine an approximation of y at any time t.

This means that for observers of the format (9) supplied with the initial value of the state of the observer at time zero that we can compute an approximation of what the state of the observer will be at later times. To do this, usually a time interval that researchers wish to inspect as well as an amount of steps is selected to determine the accuracy of the method. We get that $\Delta t = \frac{\text{time interval}}{\text{step size}}$ For the distributed Luenberger approximation the agents (which have been provided with a starting value) will then look as follows:

$$\hat{x_i}(t + \frac{\text{time interval}}{\text{step size}}) \approx \hat{x_i}(t) + \frac{\text{time interval}}{\text{step size}}(\dot{\hat{x_i}}(t)) \tag{23}$$

A big advantage of the Euler method is that it is the easiest to compute of any of the ways we know to solve initial value problems. Other methods of solving the initial value problem like the Taylor method or the Runge-Kutta of order 4 do outperform the standard Euler method [9], but are more computational and complicated.

5 Results

In this section the results of the simulations will be discussed. Before we can get to the results, we must first go over some preliminaries that have been taken.

5.1 Preliminaries

In this research I have cases where all of the agents are in $\mathbb{R}^{1\times n}$. For the computation of the trajectories of the agents the Euler method has been used. Unless otherwise specified, the standard step size used has been $\frac{1}{100000}$. I have chosen for the Euler method as it is computationally quick and easy. Furthermore, the step size of this magnitude should be accurate enough for the purposes of this research.

5.2 Coupling gain vs placed poles

As was mentioned in the background information on the distributed observer, the distributed observer given with the dynamics of equation (12) consists of 2 parts. There is the classic Luenberger observer part, $A\hat{x_i} + L_i(y_i - C\hat{x_i})$. This part of the equation is responsible for ensuring that the distributed observer converges to the state. Then there is the second part $\gamma M_i \sum_{j\in\mathcal{N}_i}(\hat{x_j} - \hat{x_i})$, responsible for the convergence of the agents towards each other. When the coupling gain γ is taken large enough, we can see convergence of the agents.

As one can imagine, the coupling gain determines how fast the agents converge towards each other. A larger coupling gain however, does not necessarily guarantee a faster convergence of the entire distributed observer towards the state. The other side of the coin of the distributed observer is the eigenvalue placement of $(A - L_iC_i)$. When the eigenvalues are placed very negatively, the convergence towards the state will be increased, but does not guarantee that the agents will find consensus quickly.

In Figure 6 the first 5 dimensions of the state (thick lines) and the agents (striped lines) is displayed in 4 scenarios. The communication graph used was the one displayed in 4 and the system and output matrices can be found in the appendix. The top left showcases the scenario with not very negatively placed eigenvalues of $(A - L_iC_i)$ (between -1 and -10) and a coupling gain that is high enough to ensure convergence, but relatively is quite low (368.4957 in this case). As we can see, it takes long for the agents to find each other and the state. In the top right, the coupling gain is increased (to about 3 times the lowest possible, or 1105.486), but the eigenvalue placement has been left as is. We now see that, as expected, the agents find each other much more quickly. The convergence to the state however sees no major advancements. Similarly, the bottom left with the low coupling gain and very negatively placed eigenvalues of $(A-L_iC_i)$ (now between -20 and -30), shows an increase in convergence to the state, but some of the agents have not found each other before they get very close to the state. This is however not the case in the bottom right example, where the eigenvalues

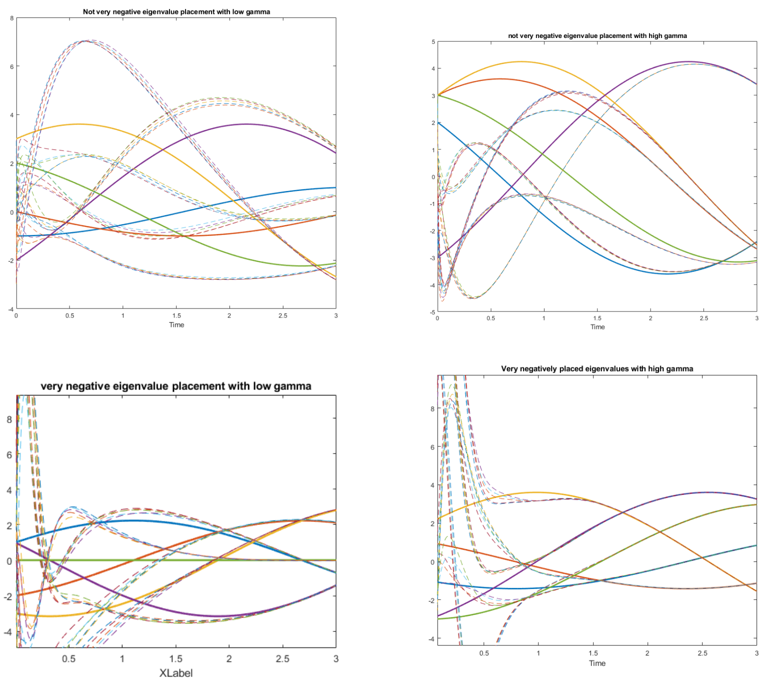


Figure 6: Figures displaying the effect of eigenvalue placement and coupling gains.
Left top: Not very negative eigenvalue placement with small coupling gain,
Left bottom: Very negative eigenvalue placement with small coupling gain,
Right top: Not very negative eigenvalue placement with high coupling gain,
Right bottom: Very negative eigenvalue placement with high coupling gain

of $(A - L_iC_i)$ have been placed very negatively, and the coupling gain has been taken very high.

5.3 Effect of the amount of connections in the graph

When looking at the distributed Luenberger observer, one could reasonably postulate that an increase in the sharing of information would lead to a faster convergence rate. To this end, we have taken a system with no observable agents. After fixing a gamma (making sure that it is still generally large enough) and the eigenvalues of the $A - L_iC_i$ matrices, we added a random connection to the adjacency matrix until every agent was connected with every other agent. By plotting the total error of the individual agents, we get the plot as given in Figure 7.

As we can see from the plot, more connections leads to a faster convergence in the beginning, but as we get further on, the fewer connections seem to perform

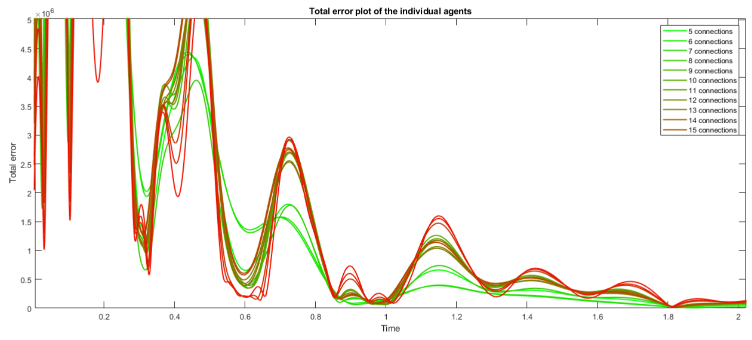


Figure 7: Greener lines have graphs with fewer connections, while redder lines have graphs with more.

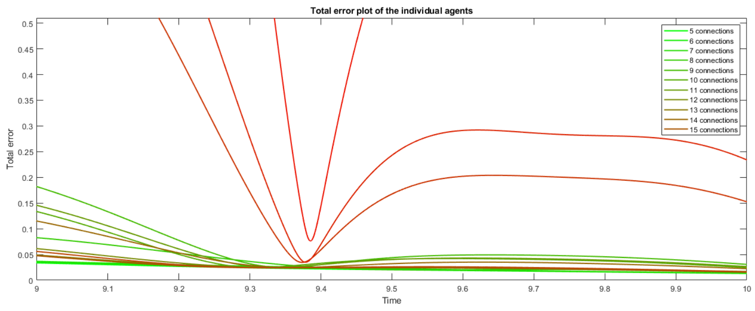


Figure 8: Same plot as in Figure 7, at the time interval $[9, 10]$

better. Furthermore, the final error at time $t = 10$ for these adjacency plots was given as in table 1.

As can be seen in the table, this data does not support the idea that more connections lead to an increase in convergence. All examples that were tested after this shared a similar outcome, where always the most connected graph performed the worst. As could be seen in the graph, the error plot seems to oscillate downwards, where the more connected plots oscillate more aggressively. One could argue that the outcome of the table could be explained by the fact that there is a peak at time $t = 10$. This however seems to not be the case when we look at the time interval $[9, 10]$ in figure 8. As we can see, no such peak can be found. Another explanation could be that with the increase in connections, the agents begin to pull each other in different directions, which would weaken the effect of the coupling, hence convergence slows down.

Connections	errors at t=10
5	0.0130
6	0.01359
7	0.0132
8	0.0169
9	0.0307
10	0.0245
11	0.0264
12	0.0165
13	0.0153
14	0.0223
15	0.0160
16	0.0166
17	0.1525
18	0.2340
19	0.7332

Table 1: Table containing the different total errors at time $t = 10$, compared to the connections of the graph.

5.4 Observable vs Unobservable

Another interesting case to consider is the difference between the observer applied to a system with only observable agents, the observer applied to a system with only unobservable agents, and a mix of the first 2 cases. In the case that all agents are observable we would expect that when the eigenvalues of the $A - L_iC_i$ matrices are set negatively enough that the convergence rate is significantly faster than the case where some or all agents are observable. However, in the case that the eigenvalues are all set moderately negative (between -15 and -25), it remains to be seen which would converge faster. Furthermore, since observable agents converge without coupling by design, one could wonder whether convergence is faster when the observable agents are not coupled or if convergence is faster when they are. To this purpose, we have tested 1200 cases of random system and input matrices.

In this comparison the communication graph, the eigenvalues of the $A-L_iC_i$, the convergence rate alpha and the amount of steps taken in the Euler method have been taken the same to try and see the impact with no other factors. The only things that has been varied are the starting positions of the agents and the state, which have been taken randomly, the system matrix $A \in \mathbb{R}^{10\times 10}$ and the output matrix $C \in \mathbb{R}^{5\times 10}$, which ensures the variance of the observability of the agents, and the coupling gain, since a specific value was required for convergence in the first place. First we tested coupling only the unobservable agents. We find the averages of the errors at time 10 as follows:

Next, we tried coupling the Observable agents as well. Since there is no method of computing the gamma for the case where all agents are observable, so the agents have remained uncoupled as a control. Similarly, the case where

Observable agents not coupled	
amount of observable agents	average total error of the agents at time 10
5	-2.9284
4	-2.5067
3	-3.0183
2	-3.1408
1	-3.5383
0	-3.5829

all agents are unobservable has remained the same and has been left in as a control.

Observable agents coupled	
amount of observable agents	average degree of error of the agents at time 10
5	-2.2701
4	-2.6626
3	-2.8873
2	-3.3460
1	-3.4169
0	-4.0280

Table 2: Amount of unobservable agents compared to their convergence rates

As we can see, when the observable agents are not coupled it results in a higher error for the cases where there is a mix of observable and unobservable agents, but as is expected, the error decreases with the decrease in the amount of observable agents. There is however barely any difference in the coupled and uncoupled results, indicating that this coupling of the observable agents is neither beneficial nor detrimental for the convergence rates. In either case we can see that the more unobservable agents are present in the distributed observer, the stronger the convergence is. An explanation for this is that the more unobservable agents the distributed observer has, the higher the chance is that one of the agents requires a large gamma for convergence. Hence a higher average gamma could cause a faster average convergence.

5.5 Funnel control

We can use funnel control not only to control a system with an input, but also systems that make use of coupling gains. In our case, the different agents all have dynamics that we can expand from (12) into the following:

$$\dot{\hat{x}}_i = A\hat{x}_i + L_i(y_i - C\hat{x}_i) + M_i \sum_{j\in\mathcal{N}_i} \gamma_{ji}(\hat{x}_j - \hat{x}_i) \tag{24}$$

Rather than having one global coupling gain, we divide the coupling gain up into smaller parts. Thus, for every agent we can set as the reference signals the states of the other agents in its information set. This gives as the error the expression $x_j - x_i$, with $j \in \mathcal{N}_i$. We can use the same basic funnel feedback law as it has been introduced in the theory section to attempt to improve on the convergence rates. We try using $\gamma_{ji} = \frac{\gamma}{1-\varphi\|e(t)\|}$, where γ is the gamma computed which is put in to ensure convergence. We implement it with the φ function given as $\varphi(t) = 10^{-m} - e^{\ln(10^{-m}-0.01)-0.1t}$. As can be seen quite easily, the function is continuous and its derivatives and itself are bounded. Furthermore, $\varphi(0) = 0.01$, which means that the initial upper bound is given as 100 and the lower bound as -100. This means that using this function allows the agents to have an initial distance between each other of 100. The variable m is the order of magnitude that one would like the error to be like. Lastly, it is fairly straightforward to see that the limit inferior is greater than zero.

Taking $m = 2$ and a random system with unobservable agents, we obtain the results displayed in Figure 9. As can be seen, the funnel control implementation yields a decent improvement. Of course however, the distributed observer without funnel control catches up later on.

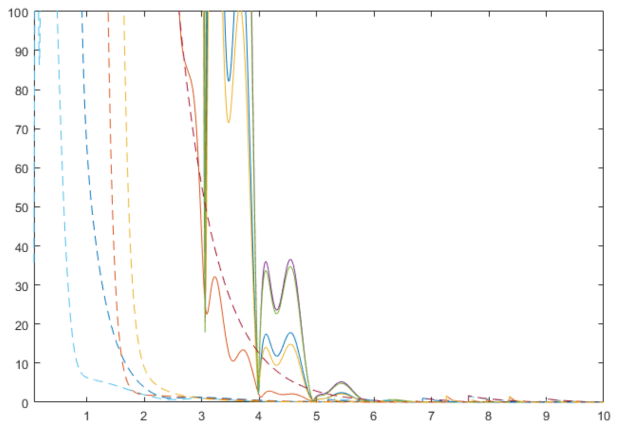


Figure 9: Error of the agents over time, striped lines are the agents with funnel coupling, unstriped are the agents without.

6 Conclusion

The distributed observer is a promising observer that will be increasingly more used in the future. The Luenberger observer is a good starting point for the design of these, and it is quite susceptible to all sorts of different improvements. The method investigated in this paper is best applied to systems with a low number of observable agents, with a graph that is not overly connected. It could be improved upon slightly with the introduction of simple funnel control.

7 Discussion

In section 5.4 one should note that there is a pretty significant disparity between the controls of the two tests. The difference between uncoupled fully observable agents and coupled fully observable agents should have acted as a control on the two computations, as they are both computed the same way. The same holds for the fully unobservable agents. We can see however that the difference is relatively high. It should thus be noted that this could indicate a too low sample size. Due to the large amount of computations that are required, no larger sample size has been attempted yet in this research.

For future research, it could prove to be interesting to investigate not only cases where the agents are $1 \times n$. Furthermore, studying systems of a greater magnitude could reap some interesting results.

References

[1] B. Besselink. *Linear systems lecture notes.*

[2] BN Biswas et al. “A discussion on Euler method: A review”. In: *Electronic Journal of Mathematical Analysis and Applications* 1.2 (2013), pp. 2090–2792.

[3] George Ellis. *Observers in control systems: a practical guide.* Elsevier, 2002.

[4] Leonhard Euler. *Institutionum calculi integralis volumen primum...* Vol. 2. 1769.

[5] Seong Han and Jang Lee. “Fuzzy Echo State Neural Networks and Funnel Dynamic Surface Control for Prescribed Performance of a Nonlinear Dynamic System”. In: *Industrial Electronics, IEEE Transactions on* 61 (Feb. 2014), pp. 1099–1112. DOI: 10.1109/TIE.2013.2253072.

[6] Weixin Han et al. *A simple approach to distributed observer design for linear systems.* 2017. DOI: 10.48550/ARXIV.1708.01459. URL: https://arxiv.org/abs/1708.01459.

[7] David Luenberger. “Observers for multivariable systems”. In: *IEEE Transactions on Automatic Control* 11.2 (1966), pp. 190–197.

[8] Richard M. Murray. *Feedback Systems: Notes on Linear Systems Theory.* Oct. 2020.

[9] Theresah oppong-kyekyeku. *Numerical analysis on initial Value Problem.* Feb. 2016. DOI: 10.13140/RG.2.1.2290.4082.

[10] Francisco F. C. Rego et al. “A design method for distributed luenberger observers”. In: *2017 IEEE 56th Annual Conference on Decision and Control (CDC).* 2017, pp. 3374–3379. DOI: 10.1109/CDC.2017.8264153.

[11] Timo Reis, Sara Grundel, and Sebastian Schöps. *Progress in Differential-Algebraic Equations II.* Jan. 2020. ISBN: 978-3-030-53904-7. DOI: 10.1007/978-3-030-53905-4.

[12] Haik Silm et al. “A simple finite-time distributed observer design for linear time-invariant systems”. In: *Systems Control Letters* 141 (July 2020). DOI: `10.1016/j.sysconle.2020.104707`.

8 appendix

Matrices used in their respective sections

Section 5.2

$$A = \begin{bmatrix} 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

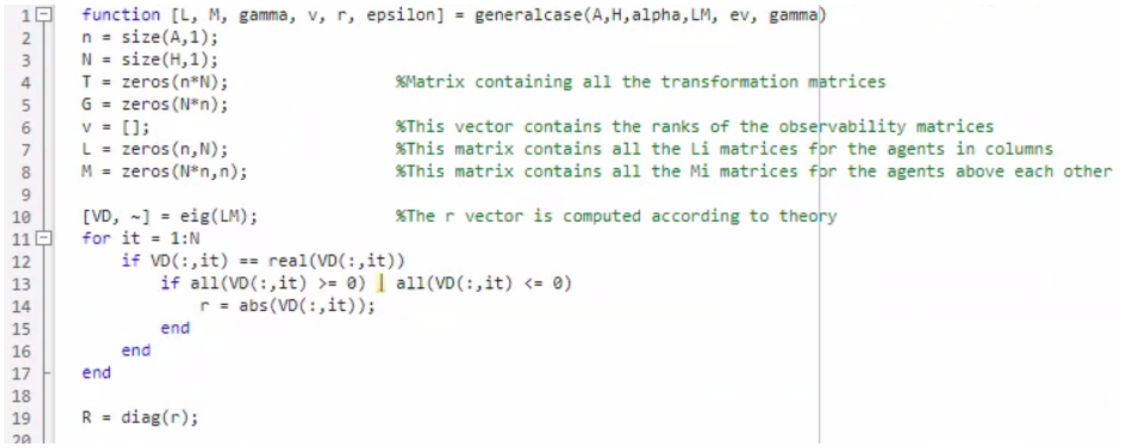
$$C = \begin{bmatrix} 7 & -8 & -7 & -8 & 3 & 5 & 4 & 7 & -1 & 0 \\ 9 & -5 & 10 & -2 & -10 & 5 & -10 & 4 & -2 & -1 \\ -8 & 1 & 10 & 9 & 7 & -2 & -5 & -4 & 6 & 3 \\ 9 & 10 & 0 & 6 & 9 & 3 & -10 & 9 & 6 & 4 \\ 3 & 10 & 6 & 10 & 4 & -7 & -8 & -10 & -7 & 5 \end{bmatrix}$$

Section 5.3

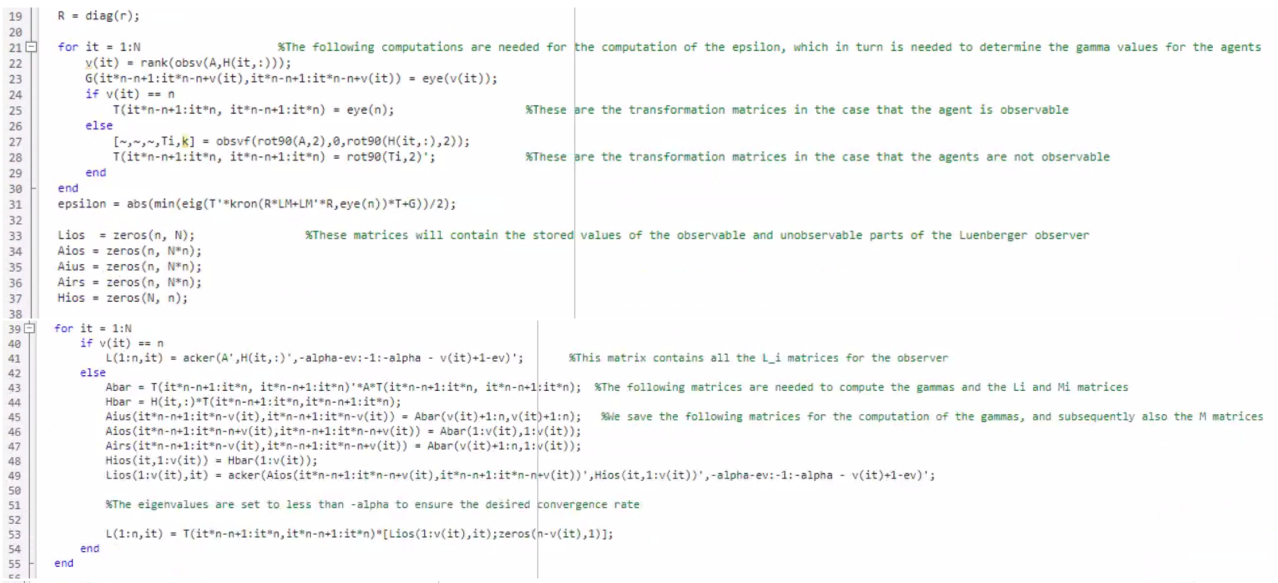
$$A = \begin{bmatrix} -4.0000 & 10.0000 & 7.0000 & 0 & 5.0000 & -3.0000 & -8.0000 & 4.0000 & -10.0000 & -7.0000 \\ -4.1465 & -4.8565 & -0.5237 & 1.2678 & -1.5241 & 0.1720 & 3.4204 & -1.1041 & -1.3261 & 2.6303 \\ -10.4394 & -17.5696 & 6.4288 & 4.8035 & 4.4278 & 1.5160 & 18.2613 & -9.3122 & 6.0218 & -5.1092 \\ 1.6591 & 22.3543 & 0.3567 & 4.2947 & -5.6417 & 0.7259 & -25.3919 & 6.9683 & 3.4673 & -1.3361 \\ -8.2197 & -9.7848 & 10.2144 & -7.0982 & -4.7861 & 6.7580 & -0.8694 & -9.6561 & -0.4891 & 9.4454 \\ -7.0000 & 0 & 9.0000 & -1.0000 & 1.0000 & -1.0000 & -6.0000 & 4.0000 & 8.0000 & 9.0000 \\ 3.5126 & 11.4978 & -11.1670 & 0.5626 & -1.1657 & -0.1021 & -11.9715 & 5.8642 & -6.8587 & 4.2941 \\ 7.3409 & -18.3543 & 1.6433 & 6.7053 & 4.6417 & 14.2741 & 14.3919 & -13.9683 & 7.5327 & 7.3361 \\ 9.8535 & -14.8565 & 9.4763 & 2.2678 & 8.4759 & 7.1720 & 13.4204 & -10.1041 & -8.3261 & -3.3697 \\ 1.0732 & 1.9283 & 1.2619 & 4.3661 & 4.2620 & 3.4140 & -11.7102 & -6.4480 & 0.1630 & -8.8151 \end{bmatrix}$$

$$C = \begin{bmatrix} 8 & -2 & -8 & -4 & -5 & -6 & -7 & -6 & -9 & -1 \\ -2 & 7 & -5 & -2 & 6 & -10 & -8 & 0 & -1 & 10 \\ -8 & -2 & -9 & -8 & -9 & -7 & 2 & -3 & -7 & 6 \\ -1 & -2 & -1 & 0 & -2 & -8 & 8 & 0 & -10 & -10 \\ -4 & -3 & -5 & 4 & -10 & -5 & 9 & -5 & 10 & 4 \end{bmatrix}$$

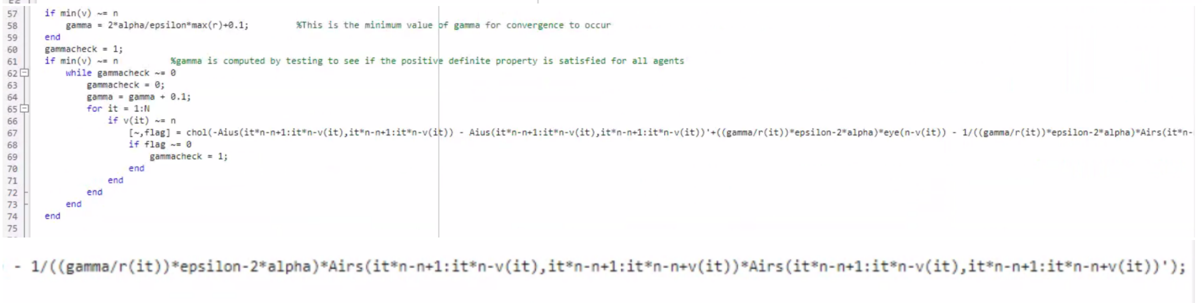
MATLAB code determining the necessary tools for the distributed observer



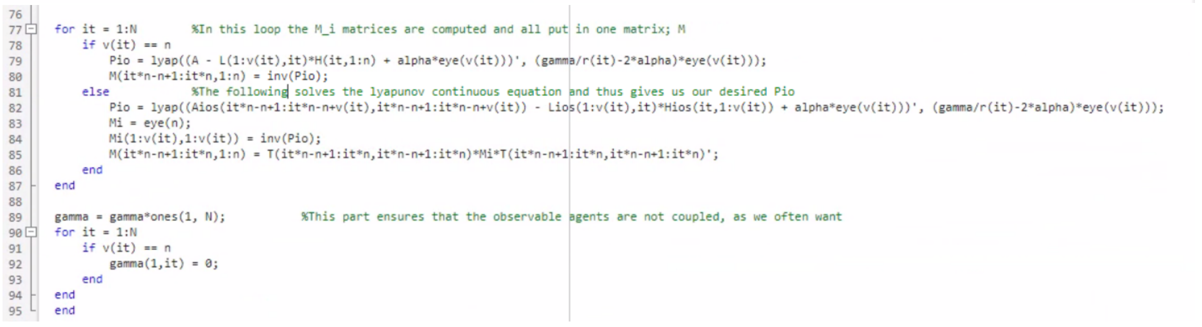
```matlab
function [L, M, gamma, v, r, epsilon] = generalcase(A,H,alpha,LM, ev, gamma)
n = size(A,1);
N = size(H,1);
T = zeros(n*N);                 %Matrix containing all the transformation matrices
G = zeros(N*n);
v = [];                         %This vector contains the ranks of the observability matrices
L = zeros(n,N);                 %This matrix contains all the Li matrices for the agents in columns
M = zeros(N*n,n);               %This matrix contains all the Mi matrices for the agents above each other

[VD, ~] = eig(LM);              %The r vector is computed according to theory
for it = 1:N
    if VD(:,it) == real(VD(:,it))
        if all(VD(:,it) >= 0) | all(VD(:,it) <= 0)
            r = abs(VD(:,it));
        end
    end
end

R = diag(r);
```



```
R = diag(r);

for it = 1:N                    %The following computations are needed for the computation of the epsilon, which in turn is needed to determine the gamma values for the agents
    v(it) = rank(obsv(A,H(it,:)));
    G(it*n-n+1:it*n-n+v(it),it*n-n+1:it*n-n+v(it)) = eye(v(it));
    if v(it) == n
        T(it*n-n+1:it*n, it*n-n+1:it*n) = eye(n);                   %These are the transformation matrices in the case that the agent is observable
    else
        [~,~,~,Ti,k] = obsvf(rot90(A,2),0,rot90(H(it,:),2));
        T(it*n-n+1:it*n, it*n-n+1:it*n) = rot90(Ti,2)';            %These are the transformation matrices in the case that the agents are not observable
    end
end
epsilon = abs(min(eig(T'*kron(R*LM+LM'*R,eye(n))*T+G))/2);

Lios  = zeros(n, N);                %These matrices will contain the stored values of the observable and unobservable parts of the Luenberger observer
Aios = zeros(n, N*n);
Aius = zeros(n, N*n);
Airs = zeros(n, N*n);
Hios = zeros(N, n);

for it = 1:N
    if v(it) == n
        L(1:n,it) = acker(A',H(it,:)',-alpha-ev:-1:-alpha - v(it)+1-ev)';      %This matrix contains all the L_i matrices for the observer
    else
        Abar = T(it*n-n+1:it*n, it*n-n+1:it*n)'*A*T(it*n-n+1:it*n, it*n-n+1:it*n);  %The following matrices are needed to compute the gammas and the Li and Mi matrices
        Hbar = H(it,:)*T(it*n-n+1:it*n,it*n-n+1:it*n);
        Aius(it*n-n+1:it*n-v(it),it*n-n+1:it*n-v(it)) = Abar(v(it)+1:n,v(it)+1:n);   %We save the following matrices for the computation of the gammas, and subsequently also the M matrices
        Aios(it*n-n+1:it*n-n+v(it),it*n-n+1:it*n-n+v(it)) = Abar(1:v(it),1:v(it));
        Airs(it*n-n+1:it*n-v(it),it*n-n+1:it*n-n+v(it)) = Abar(v(it)+1:n,1:v(it));
        Hios(it,1:v(it)) = Hbar(1:v(it));
        Lios(1:v(it),it) = acker(Aios(it*n-n+1:it*n-n+v(it),it*n-n+1:it*n-n+v(it))',Hios(it,1:v(it))',-alpha-ev:-1:-alpha - v(it)+1-ev)';

        %The eigenvalues are set to less than -alpha to ensure the desired convergence rate

        L(1:n,it) = T(it*n-n+1:it*n,it*n-n+1:it*n)*[Lios(1:v(it),it);zeros(n-v(it),1)];
    end
end
```

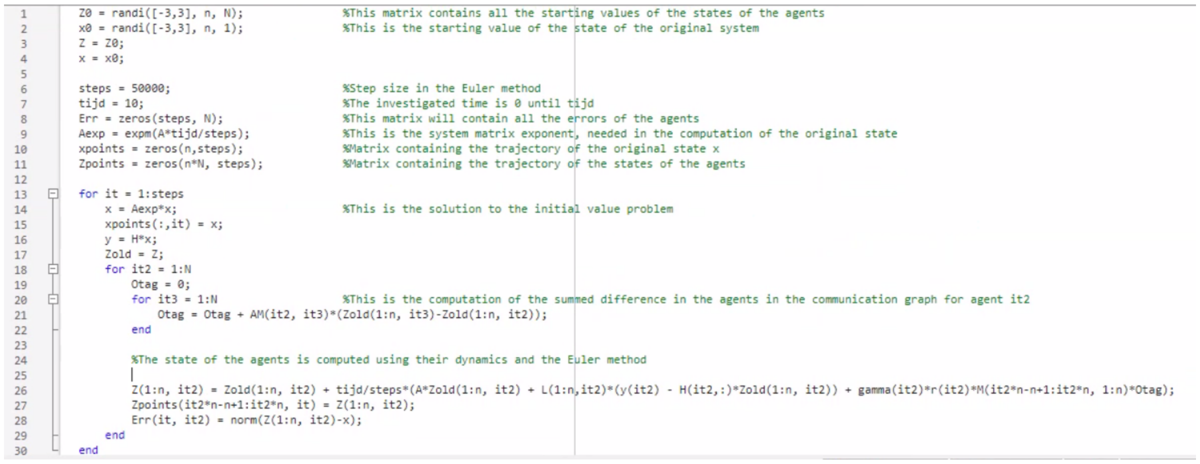


```
if min(v) ~= n
    gamma = 2*alpha/epsilon*max(r)+0.1;         %This is the minimum value of gamma for convergence to occur
end
gammacheck = 1;
if min(v) ~= n          %gamma is computed by testing to see if the positive definite property is satisfied for all agents
    while gammacheck ~= 0
        gammacheck = 0;
        gamma = gamma + 0.1;
        for it = 1:N
            if v(it) ~= n
                [~,flag] = chol(-Aius(it*n-n+1:it*n-v(it),it*n-n+1:it*n-v(it)) - Aius(it*n-n+1:it*n-v(it),it*n-n+1:it*n-v(it))'+((gamma/r(it))*epsilon-2*alpha)*eye(n-v(it)) - 1/((gamma/r(it))*epsilon-2*alpha)*Airs(it*n-
                if flag ~= 0
                    gammacheck = 1;
                end
            end
        end
    end
end

 - 1/((gamma/r(it))*epsilon-2*alpha)*Airs(it*n-n+1:it*n-v(it),it*n-n+1:it*n-n+v(it))*Airs(it*n-n+1:it*n-v(it),it*n-n+1:it*n-n+v(it))');
```



```

for it = 1:N        %In this loop the M_i matrices are computed and all put in one matrix; M
    if v(it) == n
        Pio = lyap((A - L(1:v(it),it)*H(it,1:n) + alpha*eye(v(it)))', (gamma/r(it)-2*alpha)*eye(v(it)));
        M(it*n-n+1:it*n,1:n) = inv(Pio);
    else            %The following solves the lyapunov continuous equation and thus gives us our desired Pio
        Pio = lyap((Aios(it*n-n+1:it*n-n+v(it),it*n-n+1:it*n-n+v(it)) - Lios(1:v(it),it)*Hios(it,1:v(it)) + alpha*eye(v(it)))', (gamma/r(it)-2*alpha)*eye(v(it)));
        Mi = eye(n);
        Mi(1:v(it),1:v(it)) = inv(Pio);
        M(it*n-n+1:it*n,1:n) = T(it*n-n+1:it*n,it*n-n+1:it*n)*Mi*T(it*n-n+1:it*n,it*n-n+1:it*n)';
    end
end

gamma = gamma*ones(1, N);           %This part ensures that the observable agents are not coupled, as we often want
for it = 1:N
    if v(it) == n
        gamma(1,it) = 0;
    end
end
end
```

MATLAB code that determines the trajectory and error of the agent states



```matlab
Z0 = randi([-3,3], n, N);               %This matrix contains all the starting values of the states of the agents
x0 = randi([-3,3], n, 1);               %This is the starting value of the state of the original system
Z = Z0;
x = x0;

steps = 50000;                          %Step size in the Euler method
tijd = 10;                              %The investigated time is 0 until tijd
Err = zeros(steps, N);                  %This matrix will contain all the errors of the agents
Aexp = expm(A*tijd/steps);              %This is the system matrix exponent, needed in the computation of the original state
xpoints = zeros(n,steps);               %Matrix containing the trajectory of the original state x
Zpoints = zeros(n*N, steps);            %Matrix containing the trajectory of the states of the agents

for it = 1:steps
    x = Aexp*x;                         %This is the solution to the initial value problem
    xpoints(:,it) = x;
    y = H*x;
    Zold = Z;
    for it2 = 1:N
        Otag = 0;
        for it3 = 1:N                   %This is the computation of the summed difference in the agents in the communication graph for agent it2
            Otag = Otag + AM(it2, it3)*(Zold(1:n, it3)-Zold(1:n, it2));
        end

        %The state of the agents is computed using their dynamics and the Euler method

        Z(1:n, it2) = Zold(1:n, it2) + tijd/steps*(A*Zold(1:n, it2) + L(1:n,it2)*(y(it2) - H(it2,:)*Zold(1:n, it2)) + gamma(it2)*r(it2)*M(it2*n-n+1:it2*n, 1:n)*Otag);
        Zpoints(it2*n-n+1:it2*n, it) = Z(1:n, it2);
        Err(it, it2) = norm(Z(1:n, it2)-x);
    end
end
```