BioMeld

Project:

BioMeld

Grant Agreement (GA) No. 101070328

"A MODULAR FRAMEWORK FOR DESIGNING AND PRODUCING BIOHYBRID MACHINES"

Call: HORIZON-CL4-2021-DIGITAL-EMERGING-01
Type of action: Research and Innovation action (RIA)
Start date of project: 01/10/2022 Duration: 36 months

D2.3: SYSTEM OBSERVABLES

DELIVERABLE FACTSHEET

Project title | Acronym | Number		A Modular Framework for Designing and Producing Biohybrid Machines | BioMeld | 101070328	
Due Date:	31/05/2023	**Date of submission:**	30/05/2023
Month of Project	08	**Month of submission:**	08
Title of deliverable:	D2.3 – System Observables	**Work Package:**	WP2 – Modelling and Simulation Framework
Dissemination level:	PU	**Version/Status**	1.0/Final
Deliverable leader (Name|Organisation)	Igor Balaz | UNSPF	**Editor(s)**	Milos Savic, Dusica Knezevic
Contribution of partners	UNSPF created tools, run simulations and wrote the report		
Final review and approval	UNSPF		
Keywords	System observables,		
Abstract	This deliverable focuses on the modeling and the identification of a minimum set of sensors for observing a complex system. The procedure is based on the observability of complex systems described by Liu et al. (2013). The steps involve defining equations to		

	describe the system's behavior, reconstructing an inference graph from these equations to capture information flow, decomposing the graph into strongly connected components (SCCs), and selecting sensor nodes from the root SCCs. For this purpose we developed the tool for estimation of observable parameters (EOP). which is implemented in Python and utilizes the NetworkX library. The EOP tool consists of three modules: eop_parser, eop_graph_analyzer, and main. The parser module reads and parses input files, the graph analyzer module determines SCCs and selects sensory nodes based on centrality metrics, and the main module controls the data flow between the two. Node centrality metrics, including node degree, in-degree, PageRank, and HITS hub and authority scores, are used to rank nodes and identify important nodes in the graph.

Document change history		
Date	**Authors**	**Description**
17.05.2023	Dusica Knezevic, Milos Savic	First version created with the description of the tool and test cases
30.05.2023	Dusica Knezevic, Milos Savic, Igor Balaz	Final version created with added introduction and conclusion.

CONSORTIUM

	Name	Short Name	Country
1.	UNIVERZITET U NOVOM SADU, POLJOPRIVREDNI FAKULTET NOVI SAD	UNSPF	Serbia
2.	SCUOLA SUPERIORE DI STUDI UNIVERSITARI E DI PERFEZIONAMENTO S ANNA	SSSA	Italy
3.	FUNDACIO INSTITUT DE BIOENGINYERIA DE CATALUNYA	IBEC-CERCA	Spain
4.	SMART SENSING S.R.L.	SMART SENSING	Italy
5.	UNIVERSITA DEGLI STUDI DI CAGLIARI	UNICA	Italy
6.	LEVERETTE LANCE	Lance Leverette	Belgium
7.	The University of the West of England	UWE Bristol	United Kingdom

EXECUTIVE SUMMARY

In this deliverable we developed a tool to analyze a mathematical model of a complex system and identify a minimum set of observables. The procedure is based on the observability of complex systems, as described in Liu et al., 2013.

In section 2 we describe the tool structure and implementation. The tool for the Estimation of Observable Parameters (EOP) has been developed in Python language and utilizes the NetworkX library for graph-based functionalities. The tool consists of three modules: eop_parser, eop_graph_analyzer, and main. Node centrality metrics are employed to rank nodes based on their importance in the graph, considering factors such

as node degree, in-degree, PageRank, and HITS hub and authority scores. The tool enables the selection of sensory nodes that ensure the observability of the entire system. By following this approach, the minimum set of sensors required for monitoring a complex system can be effectively determined.

In section 3 we tested the tool on three systems. The steps involve defining equations to describe the system's behavior, reconstructing an inference graph to capture information flow, decomposing the graph into Strongly Connected Components (SCCs), and selecting sensor nodes from the root SCCs. To test and verify the implementation of the EOP tool we use a system of equations from Liu et al. (2013) for which the root strongly connected components are known. The second and the third system are based on mathematical models developed for modelling catheter behavior.

In section 4 we give some concluding remarks and outlined the next steps.

LEGAL NOTICE
This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement number 101070328.
Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or European Commission. Neither the European Union nor the granting authority can be held responsible for them.

TABLE OF CONTENTS

LIST OF CODE LISTINGS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Abbreviation	Description
BHM	Bio-hybrid machine
EOP	Estimation of observable parameters
SCC	Strongly connected component

1 DESCRIPTION OF TASK

In control theory, system observables are the variables or quantities that can be measured or observed in a control system. These observables provide information about the system's behavior, state, or output. Their selection is crucial in control system design as they determine the information available for monitoring, analysis, and control. The specific system observables depend on the nature of the control system and the variables involved. They can be:

- Output variables - the ones that represent the system's output or response (e.g. physical quantities or electrical signals);
- State variables - they describe the internal state or condition of the system (e.g. position, velocity, acceleration);
- Sensor measurements - these are the variables that are directly measured by sensors or transducers in the control system, and provide feedback to the controller;
- Error or deviation variables - they represent the difference between the desired or reference value and the actual value of the system's output or state.

System observables can be determined through a combination of theoretical analysis and practical considerations. Therefore, we can point out two main approaches:

- System modeling - this approach depends on developed mathematical model of the system. This involves identifying the relevant variables, their relationships, and the dynamics of the system. In this approach, the choice of variables in the model will determine the potential observables;
- Controllability and observability analysis - Controllability refers to the ability to control the system's state from a given initial state to a desired final state. Observability, on the other hand, refers to the ability to estimate the system's state based on available measurements. Here, the choice of sensors or measurement devices plays a critical role. Their selection depends on factors such as the desired accuracy, range, resolution, and response time of the measurements.

Additional aspects in determining system observables that should be taken into account are:

- System requirements and objectives: The specific requirements, such as the desired performance criteria, control objectives, and operational constraints and objectives of the control system are the main factors that guide the selection of variables that need to be observed;
- Practical considerations: Practical constraints such as cost, availability, and feasibility of measurement may also impact the selection of observables. It is important to consider the practical limitations and trade-offs associated with measuring certain variables in the system.

In this deliverable we focused on the modeling stage and the question of identifying the minimum set of sensors through which we can observe a large complex system. The procedure is based on the work on the observability of complex systems described in **Liu et al., 2013**. The parameters estimation is based on the follow-ing steps: (1) Define a set of equations that describe the BHM behaviour; (2) Reconstruct inference graph from the set of equations by drawing a directed link xi → xj if xj appears in xi's equation, implying that one can collect information on xj by monitoring xi as a function of time. Such a graph captures the information flow in inferring the state of individual variables. Finally, by flipping the direction of each edge, the procedure recovers the system digraph encountered in structured systems theory; (3) Decompose the inference graph into a set of Strongly Connected Components (SCC) which are the largest subgraphs such that there is a directed path from each node to every other node in the subgraph. Each node in an SCC contains information pertaining to all other nodes within the SCC; (4) Select sensor nodes: from each SCCs that have no incoming edges (root SCC) select one node, which represents a potential sensor node. To ensure observability of the whole system we will choose at least one node from each root SCC.

2 TOOL FOR THE ESTIMATION OF OBSERVABLE PARAMETERS - DESCRIPTION AND IMPLEMENTATION

We developed the tool for estimation of observable parameters (EOP) which determines sensory nodes from a set of equation descriptions specified in a given textual file. The component is implemented in Python. It consists of three Python modules:

1. **eop_parser** – this module reads the provided input file, parses its content and forms the corresponding directed graph of dependencies between variables (also called the inference diagram).
2. **eop_graph_analyzer** – this module determines strongly connected components in the graph formed by eop_parser and selects sensory nodes from root strongly components according to various criteria based on centrality metrics for directed graphs.
3. **main** – this is a controller that regulates the dataflow between the two previously mentioned modules.

The EOP tool uses the NetworkX library **(Hagberg et al., 2008)** for graph-based functionalities (creating and analyzing graphs).

The parser module accepts textual files in the format specified by the following context-free grammar:

```
EQUATION_SET      -> (EQUATION "\n")+
EQUATION          -> SIMPLE_VARIABLE "|" DEPENDENCIES
DEPENDENCIES      -> VARIABLE | VARIABLE "," DEPENDENCIES
VARIABLE          -> SIMPLE_VARIABLE | INDEXED_VARIABLE
SIMPLE_VARIABLE   -> ["a" .. "z"| "A" .. "Z" | "0" .. "9"]+
INDEXED_VARIABLE -> "ARR" "[" SIMPLE_VARIABLE "," START_INDEX "," END_INDEX "]"
```

```
START_INDEX       -> INTEGER
END_INDEX         -> INTEGER
INTEGER           -> ["0" .. "9"]+
```

This means that one equation is specified in one line of the input text file ("\n" denotes new line). Each line specifies one variable (head variable) and its dependent variables. The head variable is separated by "|" from its dependent variables, while individual dependent variables are separated by comma. There are two types of variables: simple variables (one variable identified by its name, e.g. "X", "Y", "Z") and indexed variables (an array of variables determined by the same name and indexes). Names of variables are sequences of lowercase letters, uppercase letters and/or digits. The keyword "ARR'' is used to denote indexed variables. To specify such variables it is necessary to provide the name of the indexed variable, start index value and end index value (indices are natural numbers incremented from the start value to the end value by 1). For example, "ARR[X, 1, 5]" is an indexed variable X that represents five simple variables: X1, X2, X3, X4, and X5.

Since the above-context free grammar is relatively simple we implemented the parser for it from scratch without relying on any parser generator. The implementation is contained in two Python functions:

- **parse_line(line)** that parses one line and returns the list of variables given in that line where the first element of the list is the head variable (**Listing 1**)
- **parse(file_name)** that parses all lines in the specified file and forms the graph of dependencies between variables (**Listing 2**).

```
import re

def parse_line(line):
    param_arr = []

    line_split = line.split("|")

    # head variable
    param_arr.append(line_split[0].strip())

    # variables in the current equation linked to the head
    line_split = line_split[1].split(",")
    i = 0
    while i < len(line_split):
        part = line_split[i]
        if "ARR" in part:
            part = part + ", " +line_split[i+1] + ", " +  line_split[i+2]
            i = i + 3
            pattern = re.compile(r"ARR\[(.*?)\]")
            match_found = pattern.search(part).group(1)
            arr_parts = match_found.split(",")
            param = arr_parts[0].strip()
            start = int(arr_parts[1].strip())
            stop = int(arr_parts[2].strip())
            for j in range(start, stop + 1):
                param_arr.append(param + "_" + str(j))
        else:
            param_arr.append(part.strip())
            i = i + 1

    return param_arr
```

Listing 1. parse_line function

The parse_line function firstly crates the list of variables (param_arr) and then performs splitting by "|" to separate the head variable from its dependent variables. The string containing all dependent variables is then splitted by "," to form the list of dependent variables (line_split). This list is traversed element by element. If the current element is the keyword ARR it is followed by two additional parameters for the indexed variables. Needed information is easier to extract from the whole indexed variable hence merging three parts together into one string. Now, by using a grouping regular expression `r"ARR\[(.*?)\]"` group containing the name of the variable, start and end indices is extracted. By splitting this extracted group by "," individual parts are gained and used for addition of indexed variables.

```
import networkx as nx

def parse(file_name):
   file = open(file_name)
   ret_graph = nx.DiGraph()
   for line in file:
       param_arr = parse_line(line)

       src = param_arr[0]
       for i in range(1, len(param_arr)):
           dest = param_arr[i]
           ret_graph.add_edge(src, dest)

   return ret_graph
```

Listing 2. parse function

The parse function opens the input file (file_name) and forms an empty directed graph (ret_graph) by instantiating the DiGraph class from the NetworkX library. Then, it reads the input file line by line, parses the current line by calling parse_line function and forms directed links connecting the head variable to each dependent variable by calling add_edge method on ret_graph object.

The graph analyzer module implements GraphAnalyzer class that analyzes the graph formed by the parser component in order to select sensory nodes. The constructor of the GraphAnalyzer class is given in Listing 3.

```
import networkx as nx

class GraphAnalyzer:
   def __init__(self, graph, filename, metric_selection="rand"):
       self.graph = graph
       self.filename = filename
       self.metric_selection =  metric_selection

       # compute node centrality metrics
       self.pagerank = nx.pagerank(self.graph, alpha=0.15) # dict node->value
       hits = nx.hits(self.graph)
       self.hubs = hits[0]                                 # dict node->value
       self.authorities = hits[1]                          # dict node->value

       #  determine  strongly  connected  components  by  the  Tarjan  algorithm
       # implemented in NetworkX
       self.strongly_connected_components =\
              sorted(nx.strongly_connected_components(self.graph))

       # determine root strongly connected components
       self.root_components = self.__roots_scc_graph()
```

```
        # select sensory nodes from root strongly connected components
        self.selected_nodes = self.__select_nodes(metric_selection)
```

Listing 3. The constructor of GraphAnalyzer class.

The constructor performs initialization of all relevant attributes, computes node centrality metrics for directed graphs (page rank and HITS hub and authority scores), determines strongly connected components, forms the graph of strongly connected components, determines the root strongly connected components and select sensory nodes from root strongly connected components. One sensory node is selected from each root strongly connected component according to the criteria defined by the value of metric_selection field.

The main purpose of node centrality metrics is to quantify the importance of nodes according to their position and connectedness in the graph **(Savić et al., 2019)**. Node centrality metrics are used to rank nodes according to some notion of structural importance in order to identify the most important nodes in the graph. Node degree (the number of links incident to a node, or, equivalently, the number of nearest neighbors of the node if the graph does not contain loops and parallel links) is the simplest measure of node importance. In directed graphs, in-degree (the number of incoming links) is commonly used as a local metric of node importance. On the other side, there are also so-called global metrics of node importance, i.e. metrics that consider the whole graph structure. Three commonly used global node centrality metrics for directed graphs are PageRank **(Brin and Page, 1998)** and HITS hub and authority scores **(Kleinberg, 1999)**.

PageRank was initially designed for ranking nodes in a web graph crawled by the Google search engine. Three main principles behind the measure are:

1. the importance of a node *z* depends on the importance of nodes which point to *z*,
2. *z* transfers its importance to referenced nodes (nodes receiving links from *z*) in equal shares, and
3. a fixed amount of importance is given to each node "for free".

More formally, PageRank of *z*, denoted by PR(*z*) can be defined by the following recurrence relation:

$$PR(z) = \frac{1-\alpha}{N} + \alpha \sum_{w \in V\,:\,w \to z} \frac{PR(w)}{k_{out}(w)}$$
$$= \frac{1-\alpha}{N} + \alpha \sum_{w \in V} A_{wz} \frac{PR(w)}{k_{out}(w)}$$

where, $k_{out}(w)$ is the out-degree (the number of out-going links) of node *w*, *V* is the set of nodes in the graph, *N* is the number of nodes in the graph, *A* denotes the adjacency matrix of the graph and α is a constant called the damping factor (usually α is set to 0.85). Also, PageRank has a nice probabilistic interpretation related to random walks in directed graphs. PR(*z*) actually represents the probability that the random walker visits node *z* where the random walk is defined by two rules:

1. with probability α the random walker moves from the current node to one of its out-neighbors, where each out-neighbor has an equal probability to be visited, and
2. with probability 1 - α the random walker jumps to a randomly selected node from the graph (this is the so-called teleportation rule which ensures that the random walker is able to continue the walk in the case that the graph is not strongly connected).

Since PageRank is recursively defined it can be computed by successive approximations starting from a vector in which all nodes have an equal PageRank value (1/N). The NetworkX library provides a function called pagerank that computes PageRank values for all nodes in the given graph (it should be noted that the alpha parameter of the function represents the teleportation probability 1 - α) by the previously mentioned method of successive approximations.

The HITS (Hyperlink-Induced Topic Search) hub and authority scores introduced by **Kleinberg (1999)** are also initially designed for ranking nodes in web graphs. In such graphs, authorities are nodes that contain information relevant to a topic of interest, while hubs are nodes that point to authorities. A node can be considered as a good hub if it points to many good authorities, while it is a good authority if it is being referenced by many good hubs. Therefore, the main idea of the HITS algorithm is to use two mutually recursive scores indicating whether a node is a good hub, a good authority or both. The authority score of a node *z*, denoted by *A*(*z*) is proportional to the sum of hub scores of nodes pointing to *z*. On the other side, the hub score of *z*, denoted by *H*(*z*) is proportional to the sum of authority scores of nodes to which *z* points:

$$H(z) = \alpha \sum_{w \in V\,:\,z \to w} A(w)$$

$$A(z) = \beta \sum_{w \in V\,:\,w \to z} H(w)$$

where α and β are constants. Similarly as PageRank, HITS scores can be computed by successive approximations starting from the position in which all nodes have equal values of *H* and *A*. Function hits from NetworkX implements this algorithm and returns hub and authority scores for all nodes in the graph.

Strongly connected components in directed graphs can be identified by the Tarjan algorithm **(Tarjan, 1972)**. The Tarjan algorithm contains two interleaved traversals of the graph. The main traversal is DFS (depth first search). DFS is typically recursively implemented: DFS(*x*) recursively calls DFS(*y*) for each unvisited neighbor *y* of current node *x* (DFS uses a set to store all visited nodes in order to avoid visiting previously visited nodes). To each node *z* two numbers are computed: (1) DFS-index(z) – the incrementing counter indicating when *z* is visited by DFS, and (2) LOW-index(z) – the value of the lowest DFS-index to which we can go from z by back edges (links pointing to nodes with unfinished DFS recursion) located in the DFS subtree of *z*. A root of a strongly connected component is a node for which DFS-index is equal to LOW-index. Once a root is found, the algorithm performs the second traversal in which all descendants of the root node are marked as the elements of the corresponding strongly connected component. The second traversal is implemented by using a stack to which each node is pushed when visited by DFS. When the DFS of the root node is finished all nodes in the stack up to the root node are removed from the stack to form the strongly connected component recovered from the root node. Function strongly_connected_components from NetworkX implements the Tarjan algorithm. This function returns a list of sets of nodes, one for each strongly connected component of the graph.

After determining strongly connected components, the graph analyzer forms the graph of strongly connected components (**Listing 4**). Each node in this graph represents one strongly connected component. Two strongly connected components *A* and *B* are connected by a directed link *A* → *B* if *A* contains a node *x* and *B* contains a node *y* such that the *x* points to *y* in the graph (i.e., the graph contains link x → y). Method __components_connected checks whether two strongly connected components (denoted as comp1 and comp2) are connected. This method is used by getSCCGraph which performs the following operations: (1) creates an empty directed graph of strongly connected components (denoted as comp_graph in the code), (2) adds all strongly connected components in comp_graph as nodes (the first for loop) and (3) creates links in comp_graph by checking whether each pair of strongly connected components is connected (nested for loops).

```
def __components_connected(self, comp1, comp2):
    for node1 in comp1:
        for node2 in comp2:
            if (node1, node2) in self.graph.edges:
                return True
    return False
```

```
def getSCCGraph(self):
    comp_graph = nx.DiGraph()
    for i in range(0, len(self.strongly_connected_components)):
        comp_graph.add_node(i)

    for i in range(0, len(self.strongly_connected_components)):
        for j in range(0, len(self.strongly_connected_components)):
            comp1 = self.strongly_connected_components[i]
            comp2 = self.strongly_connected_components[j]
            if i != j and self.__components_connected(comp1, comp2):
                comp_graph.add_edge(i, j)

    return comp_graph
```

Listing 4. Methods of GraphAnalyzer class that form the graph of strongly connected components.

After forming the graph of strongly connected components, Graph Analyzer determines root strongly connected components (Listing 5). Root strongly connected components are represented by zero in-degree nodes in the graph of strongly connected components.

```
def __roots_scc_graph(self):
    #list of all SCCs with indegree 0
    all_zero = []
    sccGraph = self.getSCCGraph()
    for node in sccGraph.nodes:
        if sccGraph.in_degree(node) == 0:
            all_zero.append(node)

    return all_zero
```

Listing 5. The method for determining root strongly connected components.

The core functionality of Graph Analyzer is the method for selecting sensory nodes (**Listing 6**). Method __select_nodes performing this functionality has one parameter (denoted as metric in the code) which indicates how sensory nodes are selected from root strongly connected components. One sensory node from each root component is selected. Five different strategies are implemented:

1. rand – sensory nodes are selected randomly (the first node in the component is selected as sensory node)
2. pr – the node with the highest value of PageRank is selected,
3. hubs – the node with the highest value of HITS hub score is selected,
4. auth – the node with the highest value of HITS authority score is selected,
5. in – the node with the highest value of in-degree is selected (this strategy is also used when __select_nodes is called with some unknown value for metric parameter).

```
def __select_nodes(self, metric):
    best = []
    if metric == "rand":
        for comp_ind in self.root_components:
            best.append(list(self.strongly_connected_components[comp_ind])[0])
    else:
        for comp_ind in self.root_components:
            values = dict()
            if metric == "pr":
                for node in self.strongly_connected_components[comp_ind]:
                    values[node] = self.pagerank[node]
            elif metric == "hubs":
                for node in self.strongly_connected_components[comp_ind]:
```

```
                    values[node] = self.hubs[node]
                elif metric == "auth":
                    for node in self.strongly_connected_components[comp_ind]:
                        values[node] = self.authorities[node]
                else:
                    #in-degree
                    for node in self.strongly_connected_components[comp_ind]:
                        values[node] = self.graph.in_degree(node)

                values = dict(sorted(values.items(), key=lambda x:x[1], reverse=True))
                best.append(list(values.keys())[0])

        return best
```

Listing 6. The method for selecting sensory nodes.

Graph Analyzer also implements two methods that form reports about identified strongly connected components and nodes in comma separated value (CSV) file format (**Listing 6**). The first report (scc_report) lists all strongly connected components indicating which of them are root components. For each strongly connected component this report gives its size (the number of nodes) and all encompassed nodes. The second report (nodes_report) lists information about individual nodes (variables): their identifiers (variable names), a flag indicating whether a node is selected as sensory node, the identifier of the strongly connected component in which the node is located and values of all computed node centrality metrics.

```
    def scc_report(self):
        roots = self.root_components
        file = open(self.filename + "_scc_report.csv", "w")
        file.write("SCC_ID, is_root, number_of_nodes, nodes\n")

        for i in range(0, len(self.strongly_connected_components)):
            is_root = i in roots
            output = str(i) + ", " +\
              str(is_root) + ", " +\
              str(len(self.strongly_connected_components[i]))

        for node in self.strongly_connected_components[i]:
             output = output + ", " + str(node)
            file.write(output + "\n")
        file.close()



    def nodes_report(self):
        file = open(self.filename + "_nodes_report_" +\
              self.metric_selection + ".csv", "w")

        file.write("ID, is_sensor, SCC_ID, pagerank, authority, hub, indegree\n")
        for node in self.graph.nodes:
            is_sensor = node in self.selected_nodes
            scc_id = None
            for i in range(0, len(self.strongly_connected_components)):
                if node in self.strongly_connected_components[i]:
                    scc_id = i
                    break
            pr = self.pagerank[node]
            auth = self.authorities[node]
            hub = self.hubs[node]
            indeg = self.graph.in_degree(node)

            line = str(node) + ", " + str(is_sensor) + ", " +\
                     str(scc_id) + ", " + str(pr) + ", " +\
```

```
                    str(auth) + ", " + str(hub) + ", " + str(indeg) + "\n"
            file.write(line)
        file.close()
```

Listing 7. Methods for generating output reports in the CSV file format.

The last functionality implemented by Graph Analyzer is related to graph export in the GraphML format **(Brandes et al., 2014)**. This export functionality **(Listing 7)** enables us to load analyzed graphs into Gephi – a software tool for graph visualization **(Bastian et al, 2009)**. The export_graph method relies on NetworkX write_graphml function for graph exporting. Prior physical export to the output file, an attribute indicating whether a node is selected as a sensory node or not is formed. By using this attribute sensory nodes can be distinguished in Gephi (e.g., we can color sensory nodes in one color and non-sensory nodes in some other color).

```
def export_graph(self):
    sensor_dict = {}
    for node in self.graph.nodes:
        is_sensor = node in self.selected_nodes
        if is_sensor:
            sensor_dict[node] = "sensor-node"
        else:
            sensor_dict[node] = "non-sensor-node"

    nx.set_node_attributes(self.graph, sensor_dict, 'sensor')
    nx.write_graphml(self.graph, self.filename + '.graphml')
```

Listing 8. The method for graph exporting in the GraphML format.

3 RESULTS

3.1 SYSTEM 1 – TEST SYSTEM

To test and verify the implementation of the EOP tool we use a system of equations from Liu et al. (2013) for which the root strongly connected components are known (Figure 1). The system describes balance equations of chemical reactions for 11 species that are derived using the mass-action kinetics.

A **Reactions**

$$A + B + C \longrightarrow D + F + J$$
$$D \longleftrightarrow E$$
$$H + I \longleftrightarrow G$$
$$J + K \longrightarrow G + H$$

B **Balance equations**

$$\begin{aligned}
\dot{x}_1 &= -k_1 x_1 x_2 x_3 \\
\dot{x}_2 &= -k_1 x_1 x_2 x_3 \\
\dot{x}_3 &= -k_1 x_1 x_2 x_3 \\
\dot{x}_4 &= +k_1 x_1 x_2 x_3 - k_2 x_4 + k_3 x_5 \\
\dot{x}_5 &= +k_2 x_4 - k_3 x_5 \\
\dot{x}_6 &= +k_1 x_1 x_2 x_3 \\
\dot{x}_7 &= +k_4 x_8 x_9 - k_5 x_7 + k_6 x_{10} x_{11} \\
\dot{x}_8 &= -k_4 x_8 x_9 + k_5 x_7 + k_6 x_{10} x_{11} \\
\dot{x}_9 &= -k_4 x_8 x_9 + k_5 x_7 \\
\dot{x}_{10} &= +k_1 x_1 x_2 x_3 - k_6 x_{10} x_{11} \\
\dot{x}_{11} &= -k_6 x_{10} x_{11}
\end{aligned}$$

Figure 1. Structure of the Test 1 system. (A) Chemical reaction system with 11 species (A,B,. . .,J,K) involved in four reactions. (B) Balance equations of the chemical reaction system shown in A. Source: Liu et al. (2013)

In the input file format of our EOP tool, dependencies between variables are specified as follows:

x1 | x1, x2, x3
x2 | x1, x2, x3
x3 | x1, x2, x3
x4 | x1, x2, x3, x4, x5
x5 | x4, x5
x6 | x1, x2, x3
x7 | x8, x9, x7, x10, x11
x8 | x8, x9, x7, x10, x11
x9 | x8, x9, x7
x10 | x1, x2, x3, x10, x11
x11 | x10, x11

By parsing and analyzing this system we obtained a graph containing 5 strongly connected components. Table 1 shows which of the components are flagged as the root components. Selected root components by our EOP tool are the same as in the referenced paper.

Table 1. Strongly connected components for system 1

SCC_ID	**is_root**	**nodes**
0	false	x3, x2, x1
1	true	x5, x4
2	true	x6
3	false	x10, x11
4	true	x7, x8, x9

From the root components EOP selected sensory nodes by multiple criteria: random selection, selecting nodes by the highest PageRank, by the highest hub score, by the highest authority score and by the highest in-degree. Table 2 shows selected nodes by each criterion.

Table 2. Sensory nodes for system 1 by each criterion

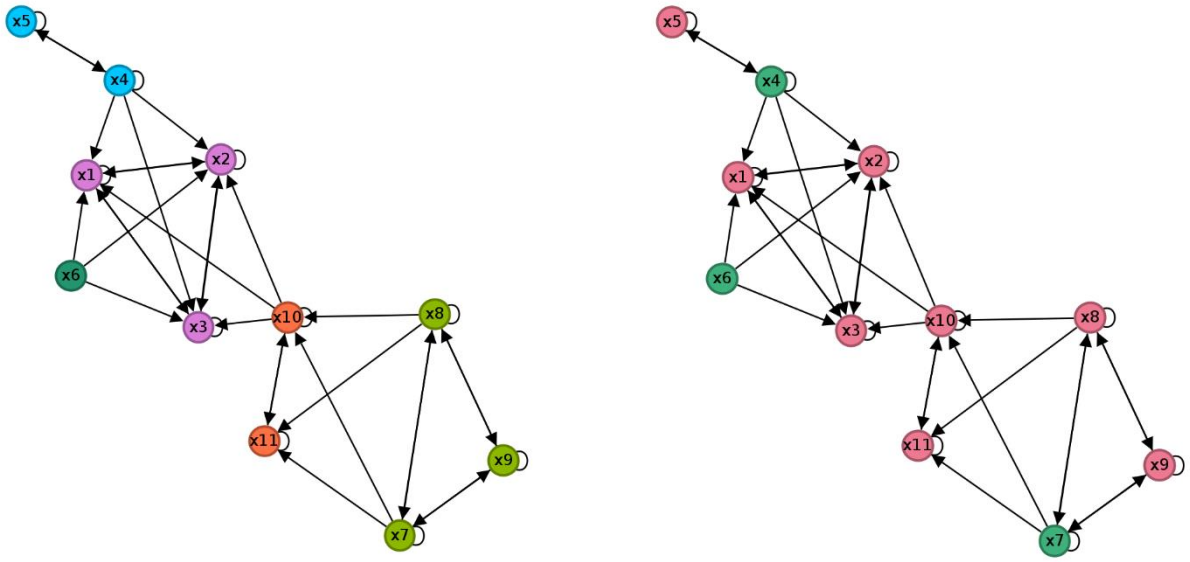
NODE_ID	**PageRank score criterion**	**Hub score criterion**	**Authority score criterion**	**In-degree score criterion**	**Random selection**
x1	false	false	False	false	false
x2	false	false	False	false	false
x3	false	false	False	false	false
x4	**true**	**true**	False	false	**true**

x5	false	false	**True**	**true**	false
x6	**true**	**true**	**True**	**true**	**true**
x7	false	false	False	false	**true**
x8	false	false	False	false	false
x9	**true**	**true**	**True**	**true**	false
x10	false	false	False	false	false
x11	false	false	False	false	false

Values for each node centrality metric are shown in Table 3. Figure 1 shows visually selected sensory nodes from System 1.

Table 3. Node centrality metric scores for System 1

NODE_ID	PageRank score	Authority score	Hub score	In-degree
x1	0.101754238	0.222387681	0.131686813	6
x2	0.101754238	0.222387681	0.131686813	6
x3	0.101754238	0.222387681	0.131686813	6
x4	0.086338301	0.044221735	0.149144073	2
x5	0.086338301	0.044221735	0.01745726	2
x6	0.077272727	0	0.131686813	0
x7	0.086823355	0.030384412	0.048239182	3
x8	0.086823355	0.030384412	0.048239182	3
x9	0.086823355	0.030384412	0.017992123	3
x10	0.092158946	0.076620125	0.161933872	4
x11	0.092158946	0.076620125	0.030247059	4



a) Detected SCCs **b)** Sensory nodes

Figure 2. Detected strongly connected components (a; nodes in the same color belong to the same connected component) and selected sensory nodes (b; sensory nodes are colored in green) for System 1 (random selection was used)

By comparing our results to the results given in Liu et al. (2013) it can be seen that our EOP tool gives the same three root components from which sensory nodes are selected. Thus, it can be concluded that the EOP tool is successfully implemented.

3.2 SYSTEM 2

As the first applicative system of equations, we used simple model developed by Salvatori et al. (2023). They analyzed the catheter mechanical properties in terms of flexural rigidity (*Fl*). For a catheter, such parameter can be calculated as follows:

$$Fl = E * I_0$$

where *E* is the Young's modulus of the material used to build the catheter, and I_0 is the moment of inertia that can be expressed for a hollow structure as:

$$I_0 = \frac{\pi(D^4 - d^4)}{64}$$

where D is the external diameter and d is the internal diameter. They analyzed the variation of the external diameter D (from 2 mm to 10 mm) and, for a specific D, the variation of the Young's modu-

lus up to 1.5 MPa. Then, they estimated the deflection of the catheter by assuming the force generated by the biohybrid actuator equal to 100 µN. According to the Euler-Bernoulli's beam theory, the deflection (δ) can be estimated as:

$$\delta = \frac{M * L^2}{2 * E * I_0}$$

where L is the catheter tip length, while M is the torque generated by the biohybrid actuator, calculated as follows:

$$M = F * \frac{d}{2}$$

Where F is the force produced by the biohybrid actuator.

The dependencies between variables are:

Fl | E, I_0
I_0 | pi, D, D_0
b | M, L, E, I_0
M | F, d

EOP analyzer detected 10 strongly connected components meaning this graph does not have a non-trivial strongly connected component (strongly connected components containing more than one node). Table 3 shows connected components for System 2.

Table 4. Detected strongly connected components for System 2

SCC_ID	**is_root**	**nodes**
1	false	pi
2	false	D
3	false	D_0
4	false	I_0
5	**true**	Fl
6	false	F
7	false	d
8	false	M

9	false	L
0	false	E
10	**true**	b

EOP analyzer marked two components as root components. These components contain only one node, so those nodes will be selected as sensory nodes for each criterion. Table 5 shows node centrality metric values for each node and which node is selected as sensory.

Table 5. Value of node metric scores and selection of sensory nodes for system 2

ID	**is_sensory**	**PageRank score**	**Authority score**	**Hub score**	**In-degree**
Fl	**true**	0.086039994	0	0.381966009	0
E	false	0.095719547	0.309016992	0	2
I_0	false	0.095719547	0.309016992	5.50E-09	2
pi	false	0.090825964	2.97E-09	0	1
D	false	0.090825964	2.97E-09	0	1
D_0	false	0.090825964	2.97E-09	0	1
b	**true**	0.086039994	0	0.618033985	0
M	false	0.089266511	0.190983004	8.50E-15	1
L	false	0.089266511	0.190983004	0	1
F	false	0.092735002	6.88E-15	0	1
d	false	0.092735002	6.88E-15	0	1

Visualization of the graph of System 2 with selected sensory nodes is depicted in Figure 2.

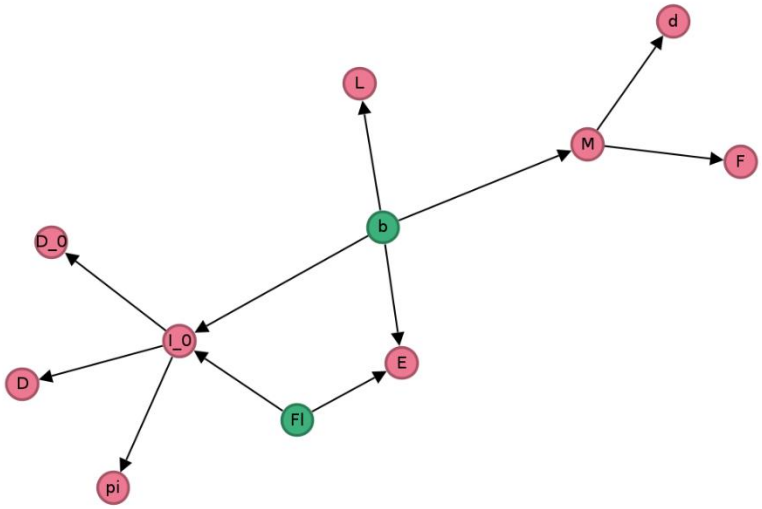


Figure 3. Selected sensory nodes for System 2

3.3 SYSTEM 3

The System 3 is based on the model developed in Ghoreishi et al (2022) whose goal is to model pre-operative selection of the catheter with proper maneuverability, capable of tracking a desired trajectory. Their focus is to represent soft catheter robots with multiple actuators and to provide a time-dependent model for characterizing the dynamics of multi-actuator soft catheter robots.

They define the state vector $\mathbf{s}_t = [\mathbf{M}_t, x_t^{init}, y_t^{init}, \theta_t^{init}]$ as the sufficient information to obtain the coordinates of any point on the catheter. Here, moment $\mathbf{M}_t = [M_{1,t}, ..., M_{n,t}]$, x_t^{init}, y_t^{init} are initial coordinates, while θ_t^{init} is the angle of the initial point on the catheter with respect to the horizontal line at step *t*. Therefore, by having the knowledge about the state of catheter at step *t* the coordinates of any point with distance l^s from the initial point of the catheter at step *t + 1* after applying $\mathbf{u}_t = [\Delta\mathbf{M}_t, \Delta d_t]$, where $\mathbf{\Delta M}_t = [\Delta M_{1,t}, ..., \Delta M_{n,t}]$ with $\Delta M_{i,t}$ being the additional moment applied by actuator *i* at step *t*, for *i* = 1,…,*n*, can be obtained as

$$x_{t+1}(\mathbf{u}_t, l^s) = f(s_t, \mathbf{u}_t, l^s),$$
$$y_{t+1}(\mathbf{u}_t, l^s) = g(s_t, \mathbf{u}_t, l^s)$$

where

$$f(\mathbf{s}_t, \mathbf{u}_t, l^s) = x_t^{init} + \Delta d_t \cos(\theta_t^{init}) + (\sum_{j=1}^{i-1}[(\frac{EI}{M_{j,t} + \Delta M_{j,t}} - \frac{EI}{M_{j+1,t} + \Delta M_{j+1,t}}) \sin(\sum_{c=1}^{j} \frac{(M_{c,t} + \Delta M_{c,t}) l_c}{EI})] +$$
$$\frac{EI}{M_{i,t} + \Delta M_{i,t}} \sin(\frac{(M_{i,t} + \Delta M_{i,t}) l_t^s}{EI} + \sum_{j=1}^{i-1} \frac{(M_{j,t} + \Delta M_{j,t}) l_j}{EI})) \times \cos(\theta_t^{init}) -$$
$$(\frac{EI}{M_{1,t} + \Delta M_{1,t}} + \sum_{j=1}^{i-1}[(\frac{EI}{M_{j+1,t} + \Delta M_{j+1,t}} - \frac{EI}{M_{j,t} + \Delta M_{j,t}}) \cos(\sum_{c=1}^{j} \frac{(M_{c,t} + \Delta M_{c,t}) l_c}{EI})] -$$
$$\frac{EI}{M_{i,t} + \Delta M_{i,t}} \times \cos(\frac{(M_{i,t} + \Delta M_{i,t}) l_i^s}{EI} + \sum_{j=1}^{i-1} \frac{(M_{j,t} + \Delta M_{j,t}) l_j}{EI})) \sin(\theta_t^{init})$$

and

$$g(\mathbf{s}_t, \mathbf{u}_t, l^s) = y_t^{init} + \Delta d_t \sin(\theta_t^{init}) + (\sum_{j=1}^{i-1}[(\frac{EI}{M_{j,t} + \Delta M_{j,t}} - \frac{EI}{M_{j+1,t} + \Delta M_{j+1,t}}) \sin(\sum_{c=1}^{j} \frac{(M_{c,t} + \Delta M_{c,t}) l_c}{EI})] +$$
$$\frac{EI}{M_{i,t} + \Delta M_{i,t}} \sin(\frac{(M_{i,t} + \Delta M_{i,t}) l_t^s}{EI} + \sum_{j=1}^{i-1} \frac{(M_{j,t} + \Delta M_{j,t}) l_j}{EI})) \times \sin(\theta_t^{init}) +$$
$$(\frac{EI}{M_{1,t} + \Delta M_{1,t}} + \sum_{j=1}^{i-1}[(\frac{EI}{M_{j+1,t} + \Delta M_{j+1,t}} - \frac{EI}{M_{j,t} + \Delta M_{j,t}}) \cos(\sum_{c=1}^{j} \frac{(M_{c,t} + \Delta M_{c,t}) l_c}{EI})] -$$
$$\frac{EI}{M_{i,t} + \Delta M_{i,t}} \times \cos(\frac{(M_{i,t} + \Delta M_{i,t}) l_i^s}{EI} + \sum_{j=1}^{i-1} \frac{(M_{j,t} + \Delta M_{j,t}) l_j}{EI})) \cos(\theta_t^{init}).$$

This system considers *n* actuators with length l_i along the catheter, the position of any point with distance l^s from the initial point of the catheter which is with distance l_i^s from the initial point of the *i*th actuator, i.e. $l^s = \sum_{j=1}^{i-1} l_j + l_i^s$. Here, Δd_t is the insertion depth.

For the system composed of 10 actuators, this system can be described as:
F | X, D, Teta, E, I, ARR[M, 1, 11], ARR[DM, 1, 11], ARR[LC, 1, 11]
G | Y, D, Teta, E, I, ARR[M, 1, 11], ARR[DM, 1, 11], ARR[LC, 1, 11]

EOP analyzer detected 41 trivial strongly connected components. Table 6 shows detected SCCs and selected root components.

Table 6. Detected strongly connected components and selected root components for System 3

SCC_ID	is_root	nodes
0	false	X
1	false	D
2	false	Teta

3	false	E
4	false	I
5	false	M_1
6	false	M_2
7	false	M_3
8	false	M_4
9	false	M_5
10	false	M_6
11	false	M_7
12	false	M_8
13	false	M_9
14	false	M_10
15	false	M_11
16	false	DM_1
17	false	DM_2
18	false	DM_3
19	false	DM_4
20	false	DM_5
21	false	DM_6
22	false	DM_7
23	false	DM_8
24	false	DM_9

25	false	DM_10
26	false	DM_11
27	false	LC_1
28	false	LC_2
29	false	LC_3
30	false	LC_4
31	false	LC_5
32	false	LC_6
33	false	LC_7
34	false	LC_8
35	false	LC_9
36	false	LC_10
37	false	LC_11
38	**true**	F
39	false	Y
40	**true**	G

Since the graph contains only trivial strongly connected components, each selected root component has only one candidate for a sensory node so this candidate will be selected for each criterion. Table 7 shows selected sensory nodes and value of each metric for nodes in System 3. Figure 3 displays sensory nodes for System 3.

Table 7. Value of node metric scores and selection of sensory nodes for system 3

ID	**is_sensory**	**PageRank score**	**Authority score**	**Hub score**	**Indegree**
F	**true**	0.024213085	0	0.5	0

X	false	0.024308657	0.013157895	0	1
D	false	0.02440423	0.026315789	0	2
Teta	false	0.02440423	0.026315789	0	2
E	false	0.02440423	0.026315789	0	2
I	false	0.02440423	0.026315789	0	2
M_1	false	0.02440423	0.026315789	0	2
M_2	false	0.02440423	0.026315789	0	2
M_3	false	0.02440423	0.026315789	0	2
M_4	false	0.02440423	0.026315789	0	2
M_5	false	0.02440423	0.026315789	0	2
M_6	false	0.02440423	0.026315789	0	2
M_7	false	0.02440423	0.026315789	0	2
M_8	false	0.02440423	0.026315789	0	2
M_9	false	0.02440423	0.026315789	0	2
M_10	false	0.02440423	0.026315789	0	2
M_11	false	0.02440423	0.026315789	0	2
DM_1	false	0.02440423	0.026315789	0	2
DM_2	false	0.02440423	0.026315789	0	2
DM_3	false	0.02440423	0.026315789	0	2
DM_4	false	0.02440423	0.026315789	0	2
DM_5	false	0.02440423	0.026315789	0	2
DM_6	false	0.02440423	0.026315789	0	2

DM_7	false	0.02440423	0.026315789	0	2
DM_8	false	0.02440423	0.026315789	0	2
DM_9	false	0.02440423	0.026315789	0	2
DM_10	false	0.02440423	0.026315789	0	2
DM_11	false	0.02440423	0.026315789	0	2
LC_1	false	0.02440423	0.026315789	0	2
LC_2	false	0.02440423	0.026315789	0	2
LC_3	false	0.02440423	0.026315789	0	2
LC_4	false	0.02440423	0.026315789	0	2
LC_5	false	0.02440423	0.026315789	0	2
LC_6	false	0.02440423	0.026315789	0	2
LC_7	false	0.02440423	0.026315789	0	2
LC_8	false	0.02440423	0.026315789	0	2
LC_9	false	0.02440423	0.026315789	0	2
LC_10	false	0.02440423	0.026315789	0	2
LC_11	false	0.02440423	0.026315789	0	2
G	**true**	0.024213085	0	0.5	0
Y	false	0.024308657	0.013157895	0	1

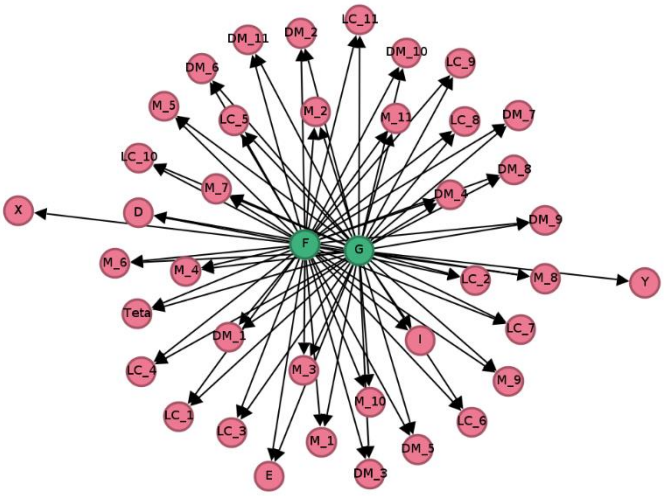


Figure 4. Selected sensory nodes for System 3

4 CONCLUSIONS

One of the main goals of this project is to develop a modeling and simulation framework that will streamline the processes of design, and testing of BHMs. The approach described here is focused on simplifying the process of defining system observables that should be measured or observed in a control system. With that we can bypass the need to know the system's kinetic constants and focus only on analyzing the topology of the inference diagram.

Crucial properties of interest in the inference diagram are root SCCs. For root SCCs of size larger than 1 any node could be chosen as a sensor. Given that some root SCCs can be quite large, and we need only one node from each root SCC, this could considerably reduce the number of sensor nodes. The minimal number of sensor nodes that are necessary to observe a system equals the number of root SCCs of its inference diagram (Liu et al., 2013).

Examples presented in this deliverable show that the modeling choices have a significant influence on determining system observables. The choice of variables and their relationships in the mathematical model directly impact the set of observables that can be derived or estimated from available measurements. Since both, model structure and the choice of variables should capture the relevant aspects of the system's behavior and dynamics, we should be very careful in the model design process.

In the next steps, we will continue working on the appropriate model development for simulating catheter behavior and controllability and iteratively test the applicability of the EOP tool.

5 REFERENCES

Bastian, M., Heymann, S., Jacomy, M. (2009). Gephi: An Open Source Software for Exploring and Manipulating Networks. Proceedings of the International AAAI Conference on Web and Social Media, 3(1), 361-362. doi: 10.1609/icwsm.v3i1.13937

Brandes, U., Eiglsperger, M., Jürgen, L., Christian, P. (2014). Graph Markup Language (GraphML). In Tamassia, Roberto (ed.). Handbook of Graph Drawing and Visualization. CRC Press. pp. 517–541.

Brin, S., Page, L. (1998). The anatomy of a large-scale hypertextual Web search engine. Computer Networks and ISDN Systems 30(1–7), pp. 107–117, doi: 10.1016/S0169-7552(98)00110-X

Ghoreishi SF, Sochol RD, Gandhi D, Krieger A and Fuge M (2022) Physics-Informed Modeling and Control of Multi-Actuator Soft Catheter Robots. Front. Robot. AI 8:772628. doi: 10.3389/frobt.2021.772628

Hagberg, A., Swart, P., & S Chult, D. (2008). Exploring network structure, dynamics, and function using NetworkX, in Proceedings of the 7th Python in Science Conference (SciPy2008), Gäel Varoquaux, Travis Vaught, and Jarrod Millman (Eds), (Pasadena, CA USA), pp. 11–15, Aug 2008

Kleinberg, J.M. (1999). Authoritative sources in a hyperlinked environment. Journal of the ACM 46(5), 604–632, doi: 10.1145/324133.324140

Liu, Y. Y., Slotine, J. J., & Barabási, A. L. (2013). Observability of complex systems. PNAS 110(7):2460-2465. doi: 10.1073/pnas.1215508110

Salvatori, C., Ricotti, L., Vannozzi, L. (2023) A novel concept of steerable catheters actuated by muscle cells: the BioMeld project. GNB2023 - VIII Congress of the National Group of Bioengineering (GNB), June 21st-23rd 2023, Padova, Italy

Savić, M., Ivanović, M., & Jain, L.C. (2019). Complex Networks in Software, Knowledge, and Social Systems, Springer Cham, doi: 10.1007/978-3-319-91196-0

Tarjan, R. E. (1972). Depth-first search and linear graph algorithms, SIAM Journal on Computing, 1 (2): 146–160, doi:10.1137/0201010