Coursework DES

Filippo Badalamenti (CID: 01998569)

February 23, 2021

The system that will be analysed is composed by a robot that can move in a specific environment (possible case of path planning, where control aspects are delegated to every single state in an hypotetic hybrid system). While the entire work done willl be presented along this coursework, all the Matlab code and file used will be also provided through the following link: `https://github.com/fil-bad/DES_coursework`.

1 Modeling the map

Since map and transition rules between rooms are provided, we can rapidly derive our finite deterministic automaton (FDA)[1] G_M, where $E = \{n, s, e, w\}$ and $X = \{Rm1, Rm2, Rm3, Rm4, Rm5, Rm6, Rm7\}$.

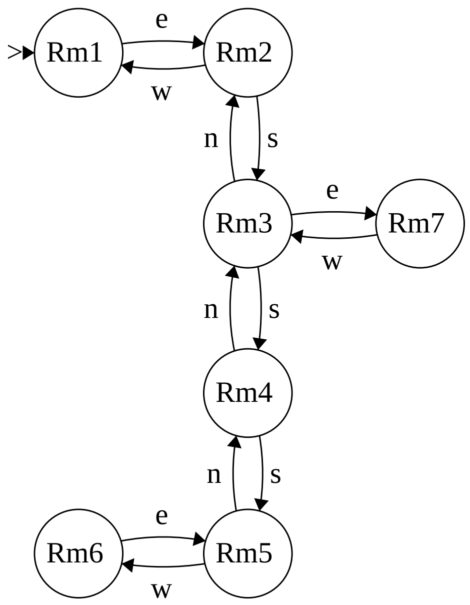


Figure 1: G_M automaton.

[1]We are going to use this tool `https://www.cs.unc.edu/~otternes/comp455/fsm_designer/` to rapidly generate an SVG file of the automaton; however, due to its intrinsic limitations, the initial state will be represented by a single arrow that doesn't start from any state.

Notice that in the following discussion the terminal state is not required since, as we can see from the next points, we will try to locate ourselves from an unknown starting state.

2 Modeling the robot

Is this case the automaton $G_R$is described by $E = \{r, n, s, e, w\}$, and $X = \{N, S, E, W\}$, where each state tells towards where the robot is facing.

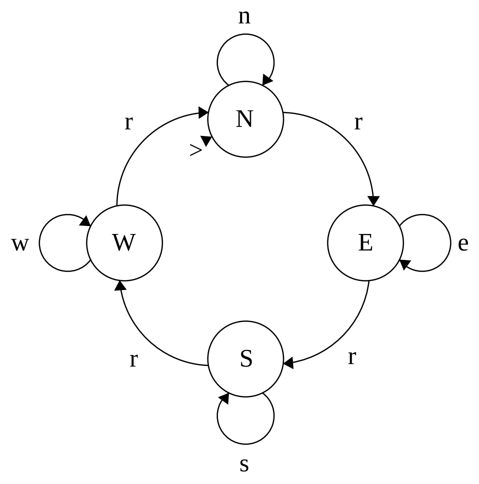


Figure 2: G_R automaton.

As we can see, both requests are satisfied – keeping track of robot heading, ensuring that only the movement in the front-facing direction is enabled –. Since the choice of the initial state does not matter, we choose N as initial one.

3 Modeling the robot inside the map

We can define the automata G_M and G_R in Matlab through the following structures:

```
% Map Automaton
G_M = struct("E",0, "X",0, "f",0, "x0",0);
G_M.E = ['n','s','e','w']';
G_M.X = ["Rm1","Rm2","Rm3","Rm4","Rm5","Rm6","Rm7"]';
G_M.f = [
        1,2,3;
        2,1,4;
        2,3,2;
        3,2,1;
        3,7,3;
        7,3,4;
        3,4,2;
        4,3,1;
        4,5,2;
        5,4,1;
        5,6,4;
        6,5,3
        ];
G_M.x0 = "Rm1";
```

```
% Robot Automaton
G_R = struct("E",0, "X",0, "f",0, "x0",0);
G_R.E = ['n','s','e','w','r']';
G_R.X = ["N","S","E","W"]';
G_R.f = [
        1,1,1;
        1,3,5;
        2,2,2;
        2,4,5;
        3,3,3;
        3,2,5;
        4,4,4;
        4,1,5
        ];
G_R.x0 = "N";
```

Remembering the definition of parallel composition, it ends up that in our case:

$$G_{Tot} = G_M \| G_R = \{E_M \cup E_R, X_M \times X_R, f, (X_{0,M}, X_{0,R})\}$$

and then we have to implement the new transition matrix, following the rules seen during class, that are:

$$f\left(\left(x_M, x_R\right), e\right) := \begin{cases} \left(f_1\left(x_M, e\right), x_R\right) & , e \in E_1 \setminus E_2 \wedge f_1\left(x_M, e\right)\ defined \\ \left(x_M, f_2\left(x_R, e\right)\right) & , e \in E_2 \setminus E_1 \wedge f_2\left(x_R, e\right)\ defined \\ \left(f_1\left(x_M, e\right), f_2\left(x_R, e\right)\right) & , e \in E_1 \cap E_2 \wedge f_{1,2}\left(x_{M,R}, e\right)\ defined \\ undefined & , otherwise \end{cases}$$

Now, we can define a Matlab function to obtain the parallel automaton. Due to the length of the code involved, it will be first conceptually analysed through bullet points, and then presented as the whole code:

1. The *union* of the two event set is carried out; to achieve this, the entire E_1 set is taken and then every other event in E_2 not already included is added.

2. The *cartesian product* of the two set of states is done generating each possible combination and rearranging it in a column vector. Although not fully efficient, the proposed method is conceptually easier.

3. For the *transition matrix*, we have to implement the previously presented possible cases, looking for each combination of states:

   (a) if we have a private event (for E_1 or E_2), then we check if it were an active event for that state, and if so, the opportune transition is computed and added to the list (taking in account that the update has to follow the new arrangement of the states).

   (b) if we have a shared event (that is $E_1 \cap E_2$), the transition in each automaton must be defined, so that the resulting one is given by their combination. In this case the triplet of $\begin{bmatrix} x & f\left(x, e\right) & e \end{bmatrix}$ is given by the rule:

   $$\begin{bmatrix} x1 + (x2 - 1) * length(G\_in1.X) & G\_in1.f(t1, 2) + (G\_in2.f(t2, 2) - 1) * length(G\_in1.X) & e \end{bmatrix}$$

   Notice that the offset $(x2 - 1) * length(G\_in1.X)$ is required in order to convert a matrix notation into a vector.

   (c) In any other case, the step of the loop is skipped.

4. For the *initial state*, their sum is considered in the resulting automaton.

Now we are ready to read through the code, even if it is suggested to download it from GitHub and visualize it through a proper text editor:

```
function G_out = par_comp(G_in1, G_in2)
% This function will merge two automata through the parallel operator,
% outputting the resulting struct. Due to the several operation involved,
% we will separe each section to be computed.

G_out = struct("E",0, "X",0, "f",0, "x0",0);

% Part I: compute the set of events.

G_out.E = G_in1.E; % initial events set

for i = 1:length(G_in2.E)
    if ~(ismember( G_in2.E(i), G_in1.E ))
        G_out.E(end+1) =  G_in2.E(i);
    end
end
% Part II: compute the new states.

x_tmp = [];
for i = 1:length(G_in2.X)
    x_tmp = [x_tmp; G_in1.X + G_in2.X(i)];
end
% the resulting states are now in a column vector of X_1*X_2 length
G_out.X = x_tmp;

% Part III: compute the transition matrix.

f_tmp = [];
for x1 = 1:length(G_in1.X) % for each new state
    for x2 = 1:length(G_in2.X)
        for e = 1:length(G_out.E) % for each new event
            if (ismember( G_out.E(e),G_in1.E ) && ...
                ~ismember( G_out.E(e), G_in2.E )) % E1 private event

                event = find( G_in1.E == G_out.E(e));
                for t = 1:length(G_in1.f)
                    if (all((G_in1.f(t,:) == [x1 0 event]) == [1 0 1]))
                        % the transition is well defined
                        f_tmp = [f_tmp;
                                [x1+(x2-1)*length(G_in1.X) G_in1.f(t,2)+(x2-1)*length(G_in1.X) e]
                                ];
                    end
                end
            elseif (~ismember( G_out.E(e),G_in1.E ) && ...
                    ismember( G_out.E(e), G_in2.E )) % E2 private event

                event = find( G_in2.E == G_out.E(e) );
                for t = 1:length(G_in2.f)
                    if (all((G_in2.f(t,:) == [x2 0 event]) == [1 0 1]))
                        % the transition is well defined
                        f_tmp = [f_tmp;
                                [x1+(x2-1)*length(G_in1.X) x1+(G_in2.f(t,2)-1)*length(G_in1.X) e]
                                ];
                    end
                end
            elseif (ismember( G_out.E(e),G_in1.E ) && ...
                    ismember( G_out.E(e), G_in2.E )) % shared event

                event1 = find( G_in1.E == G_out.E(e) );
                event2 = find( G_in2.E == G_out.E(e) );
                for t1 = 1:length(G_in1.f)
                    if (all((G_in1.f(t1,:) == [x1 0 event1]) == [1 0 1]))
                        for t2 = 1:length(G_in2.f)
                            if (all((G_in2.f(t2,:) == [x2 0 event2]) == [1 0 1]))
                                % the transition is well defined for both states
                                f_tmp = [f_tmp;
                        [x1+(x2-1)*length(G_in1.X) G_in1.f(t1,2)+(G_in2.f(t2,2)-1)*length(G_in1.X) e]
                                        ];
                            end
                        end
                    end
                end
            else
                continue
            end
        end
    end
end
G_out.f = f_tmp; % finally, assigning the computation to the output automaton

% Part IV: compute the initial condition.

G_out.x0 = G_in1.x0 + G_in2.x0;

end
```

Finally, as from the assignment request, we can list states, events and transition matrix if the automaton $G_M \parallel G_R$:

$$\mathbf{E} = \begin{pmatrix} n \\ s \\ e \\ w \\ r \end{pmatrix} \quad \mathbf{X} = \begin{pmatrix} \text{Rm1N} \\ \text{Rm2N} \\ \text{Rm3N} \\ \text{Rm4N} \\ \text{Rm5N} \\ \text{Rm6N} \\ \text{Rm7N} \\ \text{Rm1S} \\ \text{Rm2S} \\ \text{Rm3S} \\ \text{Rm4S} \\ \text{Rm5S} \\ \text{Rm6S} \\ \text{Rm7S} \\ \text{Rm1E} \\ \text{Rm2E} \\ \text{Rm3E} \\ \text{Rm4E} \\ \text{Rm5E} \\ \text{Rm6E} \\ \text{Rm7E} \\ \text{Rm1W} \\ \text{Rm2W} \\ \text{Rm3W} \\ \text{Rm4W} \\ \text{Rm5W} \\ \text{Rm6W} \\ \text{Rm7W} \end{pmatrix} \quad \mathbf{f} = \begin{pmatrix} 1 & 15 & 5 \\ 8 & 22 & 5 \\ 15 & 16 & 3 \\ 15 & 8 & 5 \\ 22 & 1 & 5 \\ 2 & 16 & 5 \\ 9 & 10 & 2 \\ 9 & 23 & 5 \\ 16 & 9 & 5 \\ 23 & 22 & 4 \\ 23 & 2 & 5 \\ 3 & 2 & 1 \\ 3 & 17 & 5 \\ 10 & 11 & 2 \\ 10 & 24 & 5 \\ 17 & 21 & 3 \\ 17 & 10 & 5 \\ 24 & 3 & 5 \\ 4 & 3 & 1 \\ 4 & 18 & 5 \\ 11 & 12 & 2 \\ 11 & 25 & 5 \\ 18 & 11 & 5 \\ 25 & 4 & 5 \\ 5 & 4 & 1 \\ 5 & 19 & 5 \\ 12 & 26 & 5 \\ 19 & 12 & 5 \\ 26 & 27 & 4 \\ 26 & 5 & 5 \\ 6 & 20 & 5 \\ 13 & 27 & 5 \\ 20 & 19 & 3 \\ 20 & 13 & 5 \\ 27 & 6 & 5 \\ 7 & 21 & 5 \\ 14 & 28 & 5 \\ 21 & 14 & 5 \\ 28 & 24 & 4 \\ 28 & 7 & 5 \end{pmatrix}$$

With some patience, it is even feasible to draw the entire automaton:

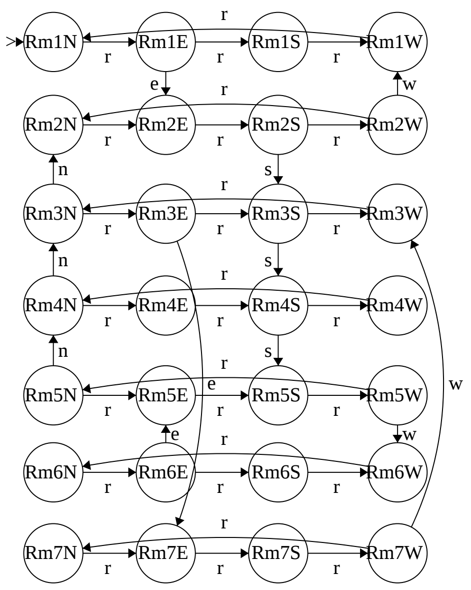


Figure 3: G_{Tot} automaton.

4 Modeling partial observability

As stated in the assessment, the robot is unaware of its heading; however, we can assume that its location it is still available. For this reason, our initial state is not completely known, as we could be in a room with any orientation, and if the next event is a rotation one, we'll continue to stay in the same one. Only when an event m happens, the robot moves in another room, and since the robot can move only in the looking-forward direction, from that moment we'll know the direction of the robot, and this information will not be lose.

For this reason, the matlab function that generates the new automaton has two main goals:

1. Substitute every transition in the parallel automaton with the new set of events $E = ['m', 'r']$
2. Add the new possible initial state for this automaton, as well as every transition that might occur.

The resulting automaton will be non-deterministic due to the uncertainty on the initial state.

Then, both resulting code and the list of events, states, and transitions, are presented in the next two pages:

```matlab
function G_out = partial_obs(G_in)
% function that change the automaton following the new partial
% observability condition.

G_out = struct("E",0, "X",G_in.X, "f",G_in.f, "x0",0);
new_e = ['m','r']';
G_out.E = new_e;

% now, we determine the position of events to be substituted
e2remove = ['n','s','e','w']';
e_index = zeros(4,1);
for e=1:length(e2remove)
    e_index(e) = find( G_in.E == e2remove(e));
end
r_old = find( G_in.E == 'r');

% change the indexes of events
for i=1:height(G_in.f)
    if ( ismember(G_in.f(i,3),e_index) )
        G_out.f(i,3) = 1;
    elseif ( ismember(G_in.f(i,3),r_old) )
        G_out.f(i,3) = 2;
    end
end

% finally , add the initial states
room = extractBefore(G_in.x0,4);

% build the state for x0
tmp_state = "{";
heading = ["N","S","E","W"];
for i=1:length(heading)
    tmp_state = tmp_state + room + heading(i) + ",";
end
tmp_state = char(tmp_state);
tmp_state(end) = '}';
G_in.x0 = string(tmp_state);

% add all the other possible initial state
add_state = [tmp_state];
for i=1:( (length(G_out.X)/length(heading))-1 ) % up to 6 in our case
    tmp_state = replace(tmp_state, num2str(i), num2str(i+1));
    add_state = [add_state; tmp_state];
end
orig_length = height(G_out.X);
G_out.X = [G_out.X; add_state];

% add the new possible transitions:
add_state = string(add_state);
new_len = height(G_out.X);

% those that are loops through 'r' event
for x=orig_length+1:new_len
    G_out.f = [G_out.f; [x x find(G_out.E == 'r')]];
end

% those that goes to a deterministic state
trans_f = find (G_out.f(:,3) == 1);
for i=1:length(trans_f)
    % find the initial state that contains the one involved in transition
    offset = find(contains(add_state, G_out.X(G_out.f(trans_f(i),1))));
    G_out.f = [G_out.f;
          [orig_length+offset G_out.f(trans_f(i),2) find(G_out.E == 'r')] ];
end

end
```

$$\mathbf{E} = \begin{pmatrix} m \\ r \end{pmatrix} \quad \mathbf{X} = \begin{pmatrix} \text{Rm1N} \\ \text{Rm2N} \\ \text{Rm3N} \\ \text{Rm4N} \\ \text{Rm5N} \\ \text{Rm6N} \\ \text{Rm7N} \\ \text{Rm1S} \\ \text{Rm2S} \\ \text{Rm3S} \\ \text{Rm4S} \\ \text{Rm5S} \\ \text{Rm6S} \\ \text{Rm7S} \\ \text{Rm1E} \\ \text{Rm2E} \\ \text{Rm3E} \\ \text{Rm4E} \\ \text{Rm5E} \\ \text{Rm6E} \\ \text{Rm7E} \\ \text{Rm1W} \\ \text{Rm2W} \\ \text{Rm3W} \\ \text{Rm4W} \\ \text{Rm5W} \\ \text{Rm6W} \\ \text{Rm7W} \\ \{Rm1N, Rm1S, Rm1E, Rm1W\} \\ \{Rm2N, Rm2S, Rm2E, Rm2W\} \\ \{Rm3N, Rm3S, Rm3E, Rm3W\} \\ \{Rm4N, Rm4S, Rm4E, Rm4W\} \\ \{Rm5N, Rm5S, Rm5E, Rm5W\} \\ \{Rm6N, Rm6S, Rm6E, Rm6W\} \\ \{Rm7N, Rm7S, Rm7E, Rm7W\} \end{pmatrix} \quad \mathbf{f} = \begin{pmatrix} 1 & 15 & 2 \\ 8 & 22 & 2 \\ 15 & 16 & 1 \\ 15 & 8 & 2 \\ 22 & 1 & 2 \\ 2 & 16 & 2 \\ 9 & 10 & 1 \\ 9 & 23 & 2 \\ 16 & 9 & 2 \\ 23 & 22 & 1 \\ 23 & 2 & 2 \\ 3 & 2 & 1 \\ 3 & 17 & 2 \\ 10 & 11 & 1 \\ 10 & 24 & 2 \\ 17 & 21 & 1 \\ 17 & 10 & 2 \\ 24 & 3 & 2 \\ 4 & 3 & 1 \\ 4 & 18 & 2 \\ 11 & 12 & 1 \\ 11 & 25 & 2 \\ 18 & 11 & 2 \\ 25 & 4 & 2 \\ 5 & 4 & 1 \\ 5 & 19 & 2 \\ 12 & 26 & 2 \\ 19 & 12 & 2 \\ 26 & 27 & 1 \\ 26 & 5 & 2 \\ 6 & 20 & 2 \\ 13 & 27 & 2 \\ 20 & 19 & 1 \\ 20 & 13 & 2 \\ 27 & 6 & 2 \\ 7 & 21 & 2 \\ 14 & 28 & 2 \\ 21 & 14 & 2 \\ 28 & 24 & 1 \\ 28 & 7 & 2 \\ 29 & 29 & 2 \\ 30 & 30 & 2 \\ 31 & 31 & 2 \\ 32 & 32 & 2 \\ 33 & 33 & 2 \\ 34 & 34 & 2 \\ 35 & 35 & 2 \\ 29 & 16 & 2 \\ 30 & 10 & 2 \\ 30 & 22 & 2 \\ 31 & 2 & 2 \\ 31 & 11 & 2 \\ 31 & 21 & 2 \\ 32 & 3 & 2 \\ 32 & 12 & 2 \\ 33 & 4 & 2 \\ 33 & 27 & 2 \\ 34 & 19 & 2 \\ 35 & 24 & 2 \end{pmatrix}$$

5 Estimating position and heading

Unlike the previous point, neither position nor heading is available for localization. For this reason, the entire state space is a possible initial state, and following the hint provided by the assignment, we can represent the states as row vectors of zeros and ones (hence $x_0 = \begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix}$). Then, an algorithm is developed in order to compute every reachable state, for every enabled event, eliminating all the state that are no more reachable, and proceeding in a parallel in-depth search until we end up in single states (as we'll see later on).

Now, answering to all the point asked in the assignment:

1. There are 77 states for the observer automaton, which can be listed as (notice: every state is

made by a row:

$$
x = \begin{pmatrix}
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
0&1&1&1&0&0&0&0&0&1&1&1&0&0&0&1&0&0&1&0&1&1&0&1&0&0&1&0\\
0&1&1&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&1&0&0&1&0&0&1&0&0&1&0&1&0&1&1&1&0&0&0&0&0&1&1&1&0&0\\
0&1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&1&1&0&0\\
0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&1&0\\
0&0&1&1&1&0&0&0&1&1&1&0&0&0&1&0&1&0&0&1&0&0&1&0&0&1&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&1&0\\
0&0&0&1&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&1&0&0&0&0&0&0&0&1&0&0&0&0\\
0&1&0&0&1&0&1&1&0&1&0&0&1&0&0&0&1&1&1&0&0&0&1&1&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&1&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&1&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1\\
0&0&0&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&1&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&1&0&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&1&0&0&0&0&0&1&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&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&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&1&0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&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&1&0&0&0&0&0&0&1&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&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&1&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&1&0&0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0\\
0&0&0&0&0&0&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&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&0&0&0\\
0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0
\end{pmatrix}
$$

2. To provide a better verification tool, the sum of the row and column, as well the first 4 singular values are provided:
TODO: AGGIUNGERE UNA FORMA MIGLIORE (VETTORE RIGA SOPRA, GLI ALTRI DUE SOTTO), E FARE LA CONSIDERAZIONE CHE GLI ELEMENTI SULLA DIAGIONALE DI S SONO NON SINGOLARI (1 COME VALORE MINIMO), A DIFFERENZA

DEL RISULTATO DEL PUNTO 6.

$$Rows = \begin{pmatrix} 28 \\ 12 \\ 4 \\ 12 \\ 2 \\ 4 \\ 4 \\ 12 \\ 2 \\ 2 \\ 4 \\ 1 \\ 4 \\ 12 \\ 1 \\ 2 \\ 2 \\ 4 \\ 4 \\ 1 \\ 1 \\ 4 \\ 4 \\ 1 \\ 2 \\ 2 \\ 2 \\ 4 \\ 1 \\ 1 \\ 1 \\ 4 \\ 4 \\ 2 \\ 4 \\ 1 \\ 2 \\ 2 \\ 2 \\ 2 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 4 \\ 2 \\ 1 \\ 2 \\ 4 \\ 1 \\ 1 \\ 2 \\ 1 \\ 1 \\ 1 \\ 4 \end{pmatrix} \quad Cols = \left(\begin{matrix} 5 & 9 & 13 & 10 & 8 & 5 & 7 & 5 & 9 & 13 & 10 & 8 & 5 & 7 & 5 & 9 & 13 & 10 & 8 & 5 & 7 & 5 & 9 \end{matrix} \right.$$

```
function G_out = observer(G_in)

G_out = struct("E",0, "X",0, "f",0, "x0",0);
% we have to first convert the event and transitions lists
G_in = convert2parobs(G_in);
G_out.E = G_in.E;
% as suggested the whole initial state is given by the vector of all ones
% notice that the state will be expressed as a row, so that the "vector" of
% states will always be a column vector.

G_out.f = {};
G_out.x0 = ones(1,height(G_in.X));
G_out.X = G_out.x0; % the first defined state
x_new = G_out.x0;

while height(x_new) > 0
    x_til = x_new(1,:); % take the first state,
    x_new(1,:) = [];    % and remove it from the queue of states

        for e = 1:length(G_in.E) % compute the reachability
            state_index = find (G_in.f(:,3) == e);  % find all the states with
                                                     % the active event 'e'
            x_next = zeros(1,height(G_in.X));

            for x = 1:length(state_index)
                x_tmp = G_in.f(state_index(x),1); % starting x state
                if (x_til(x_tmp) == 1)
                    % if the Gamma is referred to a state we're considering
                    x_next(G_in.f(state_index(x),2)) = 1;
                    % then we consider the f(x,e) as a valid transition
                end
            end
            % once found the new state
            if any(x_next) % if it is a valid state
                % we define the transition map
                G_out.f(end+1,:) = {x_til, x_next, e};

                if ~ismember(x_next, G_out.X,'rows')  % if the state does not
                                                         % exist yet
                G_out.X = [G_out.X; x_next];     % add it to the list
                                                 % of all states
                x_new = [x_new; x_next];      % add at the end of the queue
            end
        end
    end
end

% implement transitions, based on the order of the new states
f_tmp = zeros(height(G_out.f),width(G_out.f));

for h=1:height(G_out.f)
    f_tmp(h,1) = find (ismember(G_out.X, cell2mat(G_out.f(h,1)),'rows'));
    f_tmp(h,2) = find (ismember(G_out.X, cell2mat(G_out.f(h,2)),'rows'));
    f_tmp(h,3) = cell2mat(G_out.f(h,3));
end
G_out.f = f_tmp;

end


function G_po = convert2parobs(G)
% In this sub-function, we convert the list of event and transition to the
% one that we can really observe ('m' and 'r').

G_po = G;
new_e = ['m','r']';
G_po.E = new_e;

% now, we determine the position of events to be substituted
e2remove = ['n','s','e','w']';
e_index = zeros(4,1);
for e=1:length(e2remove)
    e_index(e) = find( G.E == e2remove(e));
end
r_old = find( G.E == 'r');

% finally, change the indexes of events
for i=1:height(G.f)
    if ( ismember(G.f(i,3),e_index) )
        G_po.f(i,3) = 1;
    elseif ( ismember(G.f(i,3),r_old) )
        G_po.f(i,3) = 2;
    end
end

end
```

6 OLD WORK

Rewriting the input according to state space representation:

$$u_f = \begin{cases} u_1 = -\dot{\theta} - (\theta - \theta_d) \\ u_2 = -\dot{r} - (r - r_d) \end{cases} \rightarrow \begin{cases} u_1 = -x_2 - (x_1 - \theta_d) \\ u_2 = -x_4 - (x_3 - r_d) \end{cases} \xrightarrow[\theta_d=0, r_d=0]{} \begin{cases} u_1 = -x_2 - x_1 \\ u_2 = -x_4 - x_3 \end{cases}$$

Leads the expression $\dot{x} = f(x, u_f)$ to be:

$$\dot{x} = \begin{bmatrix} x_2 \\ \frac{-x_2 - x_1 - 2x_3x_4x_2}{(x_3^2+1)} \\ x_4 \\ -x_4 - x_3 + x_3x_2^2 \end{bmatrix}$$

Again, as suggested by the assignment, we can modify the Lyapunov function adding the potential energy (that is expression of $\frac{1}{2}\theta^2$ and $\frac{1}{2}r^2$, function of position), so that it becomes:

$$\widetilde{V}(x) = \frac{1}{2}(x_3^2 + 1)x_2^2 + \frac{1}{2}x_4^2 + \frac{1}{2}x_1^2 + \frac{1}{2}x_3^2$$

Note that in this way, the $\widetilde{V}(x)$ is positive *definite* ($\widetilde{V}(x) = 0 \Leftrightarrow x = 0$), and radially unbounded. By calculating the function:

$$\dot{\widetilde{V}}(x) = \frac{\partial V(x)}{\partial x} \cdot f(x) = \begin{bmatrix} x_1 & x_2(x_3^2+1) & x_3(x_2^2+1) & x_4 \end{bmatrix} \begin{bmatrix} x_2 \\ \frac{-x_2 - x_1 - 2x_3x_4x_2}{(x_3^2+1)} \\ x_4 \\ -x_4 - x_3 + x_3x_2^2 \end{bmatrix} =$$

$$= x_1x_2 - x_2^2 - x_1x_2 - 2x_3x_4x_2^2 + x_3x_2^2x_4 + x_3x_4 - x_4^2 - x_4x_3 + x_4x_3x_2^2 =$$

$$= -x_2^2 - x_4^2 \leq -(\delta_1 + \epsilon_2) \begin{bmatrix} x_2 + x_1 & x_4 + x_3 \end{bmatrix} \begin{bmatrix} x_2 + x_1 \\ x_4 + x_3 \end{bmatrix} - (\delta_2 + \epsilon_1) \begin{bmatrix} x_2 \\ x_4 \end{bmatrix} \begin{bmatrix} x_2 & x_4 \end{bmatrix} \leq 0$$

At this point we can make some considerations about the feedback system; in fact it can be rewritten as a *MIMO linear system*, where:

$$A = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad C = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad D = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

however, since the variables are completely decoupled, it can be analyzed as two parallel SISO systems (with regard to θ and r respectively), with

$$A_1 = 0, B_1 = 1, C_1 = 1, D_1 = 1$$

as matrices and where each one is ***strictly input passive***, as their transfer function $G(s) = C(sI - A)^{-1}B + D$ verify the propriety $Re\{G(s)\} = 1 > 0, \forall s$.

For this reason, we can choose as coefficients:

$$\delta_1 = \epsilon_2 = \epsilon_1 = 0, \delta_2 \in (0, 1]$$

thus proving that $\dot{\tilde{V}}(x)$ is semi-definite negative (x_1 and x_3 could assume any value).
To prove Global Asymptotic Stability, we can use LaSalle invariance criterion, verifying that $\{0\}$ is the largest invariant set contained in $Ker\left\{\dot{\tilde{V}}\right\} := K_0$.

But $Ker\left\{\dot{\tilde{V}}\right\} = \left\{x : -x_2^2 - x_4^2 = 0\right\}$ can be seen as

$$-\begin{bmatrix} x_2 & x_4 \end{bmatrix}\begin{bmatrix} x_2 \\ x_4 \end{bmatrix} = -\left\|y_1\right\|^2 = 0$$

and, by the vector norm propriety, $\left\|y_1\right\|^2 = 0 \Leftrightarrow y_1 = 0$. For this reason, we can work with a simplified expression $K_0 = \left\{\begin{bmatrix} x_2 \\ x_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}\right\} = \{x_2 = 0 \wedge x_4 = 0\} = \mathbb{R}^2$ ($x_1 x_3$ plane). Using Lie derivatives, it is possible to calculate:

$$K_1 = \left\{ x : K_0 \wedge \mathcal{L}_f^1 \begin{bmatrix} x_2 \\ x_4 \end{bmatrix} = \frac{\partial \begin{bmatrix} x_2 \\ x_4 \end{bmatrix}}{\partial x} f(x) = 0 \right\} = \left\{ K_0 \wedge \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_2 \\ \frac{-x_2 - x_1 - 2x_3x_4x_2}{(x_3^2+1)} \\ x_4 \\ -x_4 - x_3 + x_3x_2^2 \end{bmatrix} = 0 \right\}$$
$$= \left\{ K_0 \wedge \begin{bmatrix} \frac{-x_2 - x_1 - 2x_3x_4x_2}{(x_3^2+1)} \\ -x_4 - x_3 + x_3x_2^2 \end{bmatrix} = 0 \right\}$$

and using the relations found in the previous iteration, we could simplify the expression:

$$\left\{ K_0 \wedge \begin{bmatrix} \frac{-x_1}{(x_3^2+1)} \\ -x_3 \end{bmatrix} = 0 \right\} = \left\{ K_0 \wedge \begin{bmatrix} x_1 \\ x_3 \end{bmatrix} = 0 \right\} = \{x_2 = 0 \wedge x_4 = 0 \wedge x_1 = 0 \wedge x_3 = 0\} = \{0\} = K_1$$

So the largest invariant set is $\Omega \subset K_1 = \{0\} \Rightarrow \{0\}$ is GAS.

7 Simulate the system for different constant values of r_d and θ_d

In order to show the results achieved in a way conceptually similar to reality, the simulation has been carried out in Simulink. The built model is the following one, where:

- the System block is described by the following Matlab function:

```
function [xDot,y] = System(x,u)
    xDot=zeros(4,1); % defining size of xDot
                     % (required for simulation)
    xDot(1) = x(2);
    xDot(2) = (u(1)-2*x(3)*x(4)*x(2))/((x(3)^2)+1);
    xDot(3) = x(4);
    xDot(4) = u(2)+x(3)*(x(2)^2);

    y=zeros(2,1); % defining size of y
    y(1)=x(2);
    y(2)=x(4);
end
```

- the IC block allows to set a specific initial condition (in our case, it's sufficient to not be the origin, since it is an equilibrium).
- the feedback loop is built from the output y and accordingly to the equation given by the assignment.
- the scope is attached to the state, showing that all its components converge to a specific value.

Another way to describe the system is through a Matlab function that would be integrated through ode45:

```
function [xDot] = sys(t,x)
    global theta_d r_d;

    u(1)=-x(2)-( x(1) - theta_d );
    u(2)=-x(4)-( x(3) - r_d );

    xDot(1,1) = x(2);
    xDot(2,1) = (u(1)-2*x(3)*x(4)*x(2))/((x(3)^2)+1);
    xDot(3,1) = x(4);
    xDot(4,1) = u(2)+x(3)*(x(2)^2);
end
```

where the global variables are set into an external script to be easily accessible and editable.

he simulations - we refer to the first plot due to its smaller scale and better resolution -, once the θ_d and r_d value are fixed and the system begins to evolve, as we can see the x_1 and x_3 components of the state (θ and r respectively) converge to the reference values, while x_2 and x_4 (that are $\dot{\theta}$ and $\dot{r}$) reduce their amplitude towards zero as the state approaches the asymptotic value.

h higher reference value, the overshoot of both the components of the state increases, but eventually, after a long transient, the state converges to:

$$x_\infty = \begin{bmatrix}\theta_d & 0 & r_d & 0\end{bmatrix}^T$$

In conclusion, we can claim that the controller achieves *asymptotic tracking for constant values*, with the special case of $\theta_d = 0, r_d = 0$, where the converging point is the origin.

8 Prove that the closed-loop system is not ISS

8.1 Build an appropriate input signal

Unfortunately, this point was not really clear for me. Due to the hint to use the Lyapunov function and the form of $k(x(t)) : \mathbb{R}^4 \longrightarrow \mathbb{R}$ for the disturbance, I initially thought that the function was exactly $\widetilde{V}(x)$, or a similar function; however, this cannot be possible, since a sign definite function would simple lead to a constant value tracking.

On the other hand, since the idea is to maximize the power, we could touch the variables that are involved with velocity (x_2 or x_4), and since $\theta_d$appears in x_2, and according to the simulation for constant values, the best approach would be to invert the input signal every time the x_2 crosses zero. As we would expect, x_1 has an oscillatory unbounded solution (it is directly influenced by x_2, so that each time the solution changes direction, it is accelerated by the new value of derivative). Moreover, the disturbance should have a rather large constant compared to initial condition, being the only positive term for $\dot{x_2}$.

Another consideration could be done using the derivative of Lyapunov function with this specific disturbance. In fact, doing some calculations, for $\theta_d = c \cdot sign\left(k\left(x\left(t\right)\right)\right), r_d = 0$, we obtain:

$$\dot{\widetilde{V}}(x, d) = -x_2^2 - x_4^2 + x_2 \cdot c \cdot sign\left(k\left(x\left(t\right)\right)\right)$$

and choosing $k\left(x\left(t\right)\right) = x_2$, we would get:

$$\dot{\widetilde{V}}(x, d) = -x_2^2 - x_4^2 + |x_2| \cdot c$$

that not only is not definite negative, it is also not sign defined for $c > 0$. Even with some manipulations (rewriting $\dot{\widetilde{V}}(x, d)$ as $-\frac{x_2^2}{2} - x_4^2 + \frac{\theta_d^2}{2}$, or find $\chi\left(|\theta_d|\right) = 2\left|\theta_d\right| = 2\left|c \cdot sign\left(k\left(x\left(t\right)\right)\right)\right| = 2\left|c\right|$) we cannot use this Lyapunov function as an ISS one of either types; another proof is given by Theorem 19 on Lecture Notes.

8.2 Show the resulting simulation

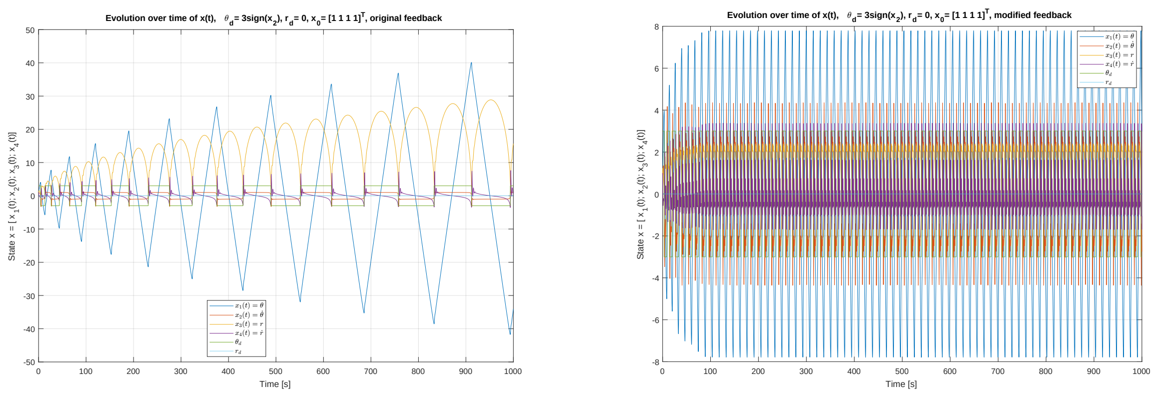


Figure 4: Simulation with $\theta_d = 3sign\,(x_2)\,, r_d = 0$ and $x_0 = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}^T$ for both original and modified feedback.

As noticeable by the figure and by their description, we have chosen the same exact conditions, while using different feedback, where the second one is the same used in the next point analysis. In the first one, both x_1 and x_3 are diverging and, while x_2 becomes somewhat the same at each cycle of its waveform, x_4 continues to increase its peak, pushing higher x_3 and consequently x_1.

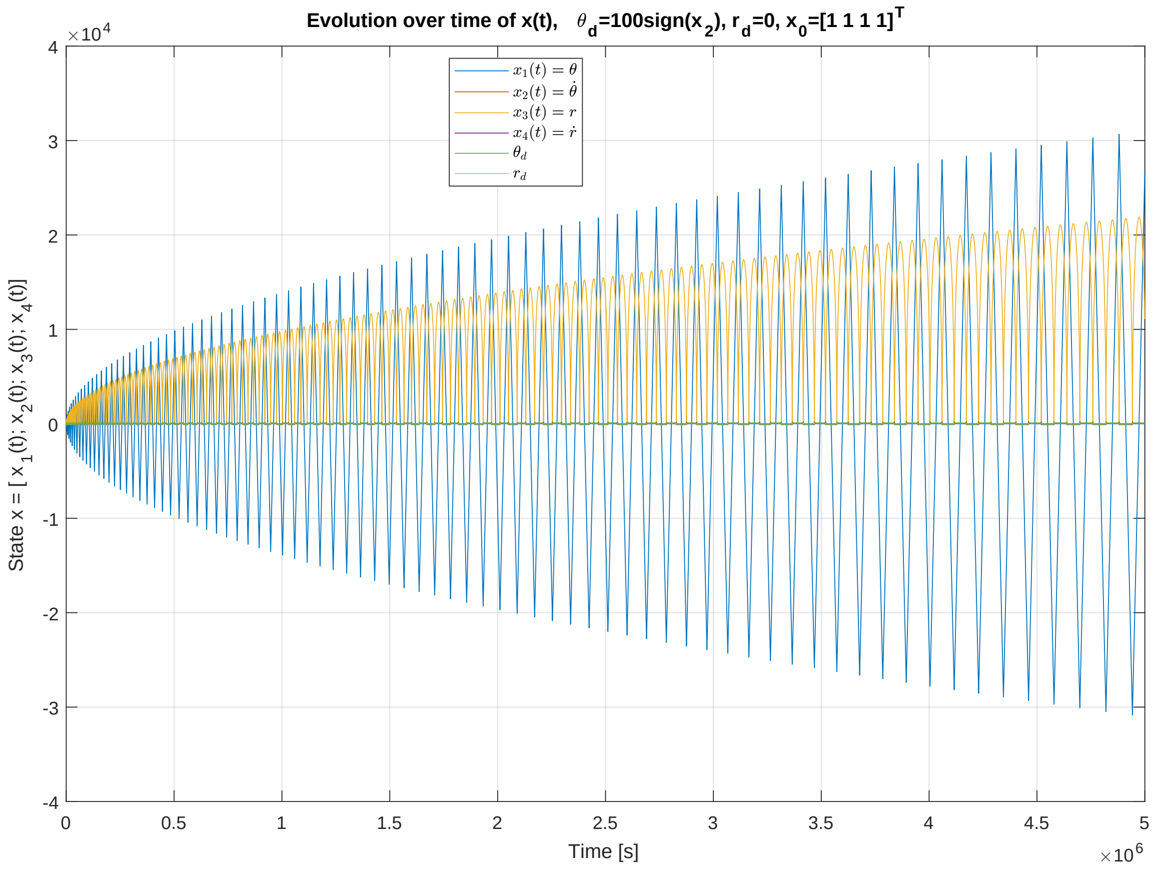


Figure 5: Simulation with $\theta_d = 100sign\left(x_2\right), r_d = 0$ and $x_0 = \begin{bmatrix}1 & 1 & 1 & 1\end{bmatrix}^T$, original feedback.

Even for larger times, the state continues to increase, following an $\sqrt{x}$ form and thus being unbounded, while having a bounded input.

9 Show that the modified feedback fulfills the ISS properties

With the new pair of inputs:

$$u_N = \begin{cases} u_1 = -\dot{\theta} - (\theta - \theta_d) \\ u_2 = -\dot{r} - (r - r_d) - (r^3 - r_d^3) \end{cases} \rightarrow \begin{cases} u_1 = -x_2 - (x_1 - \theta_d) \\ u_2 = -x_4 - (x_3 - r_d) - (x_3^3 - r_d^3) \end{cases}$$

the state space representation becomes:

$$\dot{x} = \begin{bmatrix} x_2 \\ \frac{-x_2 - (x_1 - \theta_d) - 2x_3x_4x_2}{(x_3^2+1)} \\ x_4 \\ -x_4 - (x_3 - r_d) - (x_3^3 - r_d^3) + x_3x_2^2 \end{bmatrix}$$

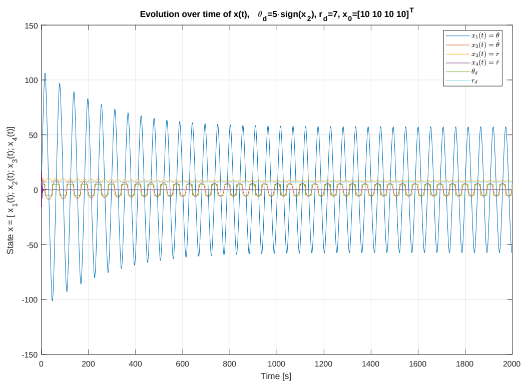
Then, doing some computations, we can show how this system fulfills the typical ISS properties.

9.0.1 Input-to-State Stable Definition

A system is ISS if and only if:

$$|\phi(t, x_0, d)| \leq \max\left\{\beta\left(|x_0|, t\right), \gamma\left(\|d\|_\infty\right)\right\}, \beta \in \mathcal{KL}, \gamma \in \mathcal{K}_\infty$$

Due to this definition, we have a shape in which our solution is bounded, and for large time, it almost solely depends on the second term.



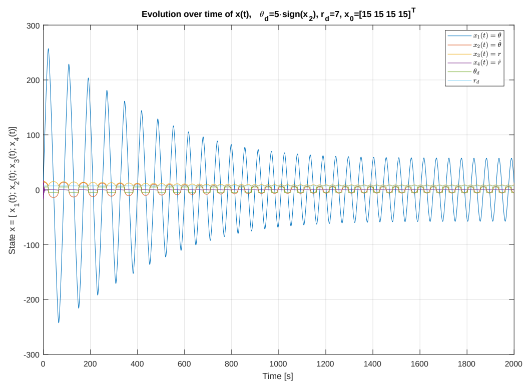


Figure 6: Simulation for $\theta_d = 5sign(x_2), r_d = 7$, and $x_{0_1} = \begin{bmatrix}10 & 10 & 10 & 10\end{bmatrix}^T$,$x_{0_2} = \begin{bmatrix}15 & 15 & 15 & 15\end{bmatrix}^T$respectively.

As we can notice from the two figures above, when x_0 increases the transient does the same, while the disturbance, being the same, leaves the second part of the bound (given by γ for large time) unaltered.

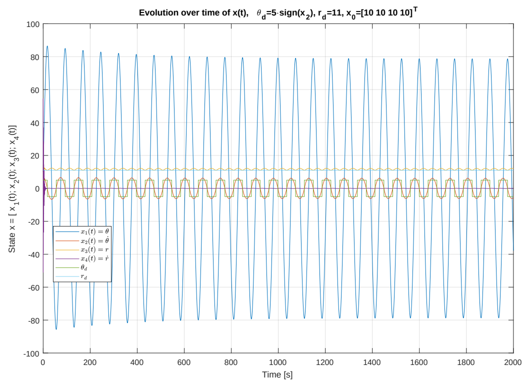


Figure 7: Simulation for $\theta_d = 5sign(x_2), r_d = 11$, and $x_0 = \begin{bmatrix}10 & 10 & 10 & 10\end{bmatrix}^T$.

When the disturbance increases (only one component in our example for simplicity of analysis), we can note how the γ part increases, leading to a steady state value bigger according to $\mathcal{K}_\infty$ function definition.

9.0.2 Converging-Input Converging-State (CICS)

To prove this property, we could choose a suitable disturbance that goes to zero for $t \to +\infty$. For simplicity of analysis $r_d$is fixed to zero, while $\theta_d = 3e^{-\frac{t}{10}}$, an $\mathcal{L}$-type function (again, one of the simplest possible).

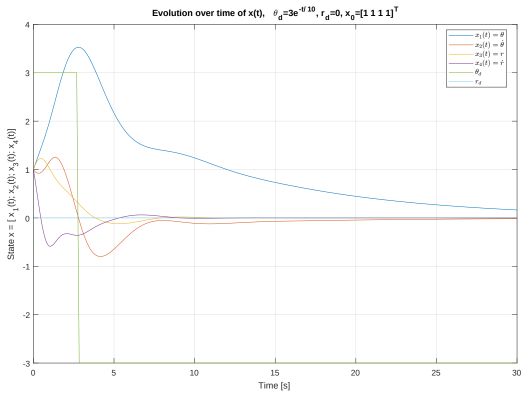


Figure 8: Simulation for $\theta_d = 3e^{-\frac{t}{10}}, r_d = 0$, and $x_0 = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}^T$.

As we can notice from the figure, after a transient somewhat "chaotic", the system is bounded by the γ part of the ISS definition, thus the solution goes down exponentially to zero.

9.0.3 0-GAS

The Global Asymptotic Stability in absence of disturbance ($d = 0$), means that the solution is bounded by the β part of the ISS definition, an $\mathcal{L}$-type function that eventually goes to zero for $t \to +\infty$.

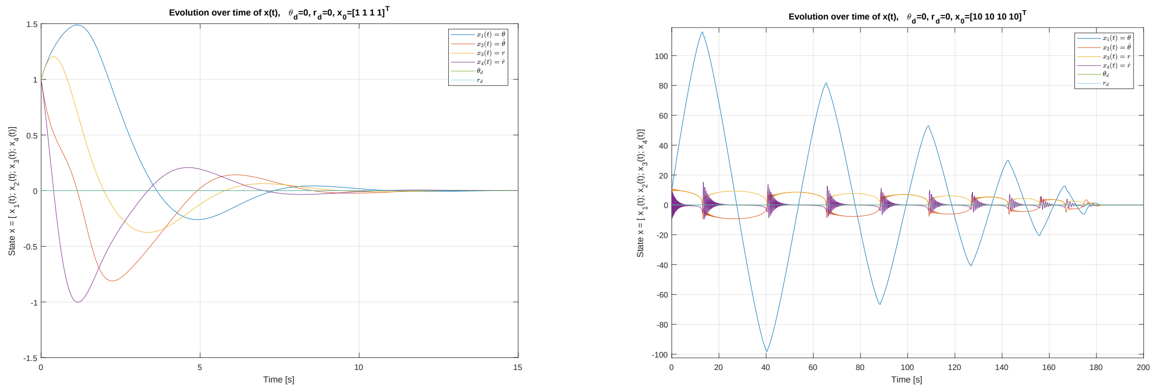


Figure 9: Simulation for $\theta_d = 0, r_d = 0$, and $x_{0_1} = \begin{bmatrix}1 & 1 & 1 & 1\end{bmatrix}^T$,$x_{0_2} = \begin{bmatrix}10 & 10 & 10 & 10\end{bmatrix}^T$respectively.

As we can see by the figures, the system goes to zero regardless of initial condition and, after an eventually long transient, it converges to the origin.