Coursework DES

Filippo Badalamenti (CID: 01998569)

March 1, 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 $X = \{Rm1, Rm2, Rm3, Rm4, Rm5, Rm6, Rm7\}$.

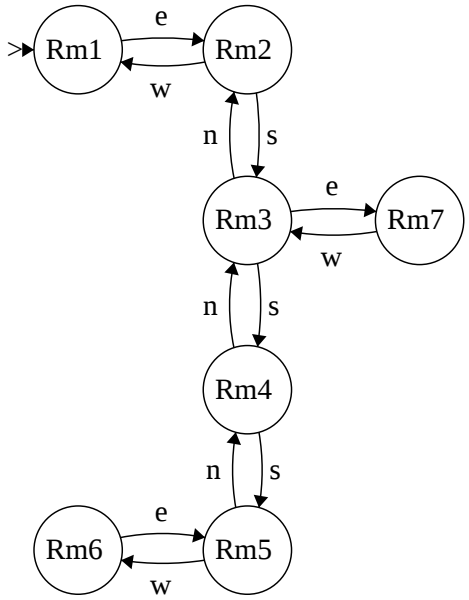

Figure 1: G_M automaton.

[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.

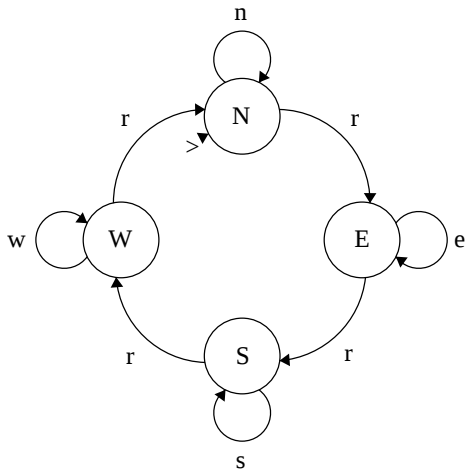

Figure 2: G_R automaton.

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 the following points, and then presented as the whole code:

1. The *union* of the two event sets 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 sets of states is done generating each possible combination and rearranging them in a column vector. Although not fully efficient, the proposed method is conceptually easier to be implemented.

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 there 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, although it is suggested to be downloaded from GitHub and being visualized 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 $G_M \parallel G_R$:

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

Notice that with some patience, it is still feasible to draw the entire automaton:

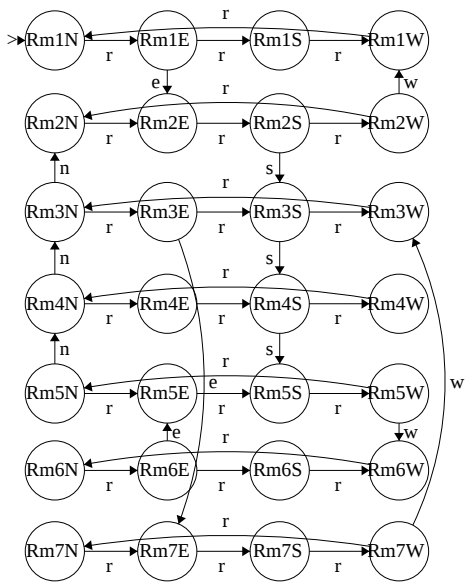

Figure 3: G_{Tot} automaton.

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 specific room but with any orientation, and if the next event is a rotation one, we'll continue to stay in the same uncertain state. 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 lost.

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 states for this automaton, as well as all the transitions that might occur from them.

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:

```
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 longer reachable, and proceeding in a parallel in-depth search until we end up in single states (as we'll see later on).

```
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
```

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

1. There are 77 states for the observer automaton, which can be listed as a matrix where each 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&1&1&1\\
0&1&1&1&0&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&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\\
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&1&1&0&0&0&0&0&0&1&1&0&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&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&0&1&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&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&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&0&0&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&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&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&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&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&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&1&0&0&0\\
0&0&1&0&0&0&0&0&1&0&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&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&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&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&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&1&1&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&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&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&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&0&1&0&0&0&0&0&1&0&0&0&0&1&0&0&0&0&0&0&1&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\\
1&0&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&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&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&0&1&0&0&0&0&0&1&0&0&0&0&1&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&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&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&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&1&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&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&1&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&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&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&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&1&1&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&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&0&0&1&0&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&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&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&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&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&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&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&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&1&0&0&0&0&0
\end{pmatrix}$$

2. To provide a more readable verification tool, the sum of the row and column, as well the first 4 singular values are provided. As we can see, two important results can be obtained by their values:

(a) the matrix S is a tall matrix with all the values on the main diagonal greater than zero (actually ≥ 1).

(b) Thanks to the *rows* vector, we know for each state how many of them have precise information about robot position and heading; as we can see, there are 28 ones in this vector, that means that if an appropriate word happens, it is possible to end up in a deterministic state (it is obvious that just spinning would not improve our knowledge on position).

It is important to notice that both results will be different from the ones obtained in the next point of the coursework.

$$Cols = \begin{pmatrix} 5 & 9 & 13 & 10 & 8 & 5 & 7 & 5 & 9 & 13 & 10 & 8 & 5 & 7 & 5 & 9 & 13 & 10 & 8 & 5 & 7 & 5 & 9 & 13 & 10 & 8 & 5 & 7 \end{pmatrix}$$

$$Rows = \begin{pmatrix} 28 \\ 12 \\ 4 \\ 12 \\ 2 \\ 4 \\ 4 \\ 12 \\ 2 \\ 2 \\ 4 \\ 1 \\ 4 \\ 12 \\ 1 \\ 2 \\ 2 \\ 4 \\ 4 \\ 1 \\ 1 \\ 4 \\ 4 \\ 1 \\ 2 \\ 2 \\ 2 \\ 4 \\ 1 \\ 1 \\ 1 \\ 4 \\ 4 \\ 2 \\ 4 \\ 1 \\ 2 \\ 2 \\ 2 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 4 \\ 2 \\ 1 \\ 2 \\ 4 \\ 1 \\ 1 \\ 2 \\ 1 \\ 1 \\ 1 \\ 4 \\ 2 \\ 1 \\ 2 \\ 4 \\ 1 \\ 1 \\ 1 \\ 4 \\ 2 \\ 1 \\ 2 \\ 1 \\ 1 \\ 2 \\ 2 \\ 1 \\ 2 \\ 2 \\ 2 \\ 2 \end{pmatrix} \qquad Sing\_vals = \begin{pmatrix} 8.2219 \\ 4.7713 \\ 4.2310 \\ 4.2310 \end{pmatrix}$$

3. In order to compute every possible transition given by a word, a new Matlab function has been implemented:

```
function x_final = explore_obs(G,w)
% Given any word and an observer automaton from an unknown state, we find
% out in which state we end up, starting from the completely unknown state.

if isempty(w)
    x_final = G.x0;
    return;
end
x_final = find(ismember(G.X,G.x0,'rows')); % initial state position.

for e=1:length(w)
    e_num = find (G.E == w(e));
    f_row = find(ismember(G.f(:,[1 3]), [x_final e_num], 'rows'));

    if isempty(f_row) % our word has led us to an illegal state
        x_final = NaN;
        return;
    end
    x_final = G.f(f_row,2);
end

% once found the final state of the word, we have to convert to the actual
% value of the state.

x_final = G.X(x_final,:);

end
```

In this way, we can compute where we could possibly end up. In our case, since the state is certain after the word $w = "mrrmrmrrmrm"$– that means only one 1 is present in the row state vector of the observer –, we can match it in the parallel automaton, and the resulting state is $x = f(x_0, w) = "Rm2N"$.

4. As shown in the code above, in order to find the final state, it is sufficient use the extension of f to words, such that $f(x, se) := f(f(x, s), e)$, looking to the column of the transition matrix of the observer automaton.

6 Question: Deterministic Interpretation

As analysed above, due to the fact that there are 28 single states, we can affirm that we are able to reconstruct the exact position of the robot; this is due to the fact that the map has no symmetry. In fact, once the map it is modified, if we compute the same tools used before (row and column sum vectors, singular values), we can see that there are no single value states; instead, we have $|X|/2$ double state because of the symmetry of the map.

For this reason, as we are only able to observe a movement and a rotation without knowing towards where we are moving, we have a specular behaviour along the horizontal median line of the map, and it will be impossible to reconstruct our position and heading (we will always have uncertainty between at least two states). One of the signs of this behaviour can be found in the singular matrix S, where the last half of the diagonal values are really close to zero, and thus it is ill-conditioned.

$$Cols = \begin{pmatrix} 3 & 5 & 6 & 6 & 5 & 3 & 3 & 5 & 6 & 6 & 5 & 3 & 3 & 5 & 6 & 6 & 5 & 3 & 3 & 5 & 6 & 6 & 5 & 3 \end{pmatrix}$$

$$Rows = \begin{pmatrix} 24 \\ 10 \\ 4 \\ 10 \\ 2 \\ 4 \\ 2 \\ 10 \\ 2 \\ 4 \\ 2 \\ 2 \\ 10 \\ 2 \\ 4 \\ 4 \\ 2 \\ 2 \\ 2 \\ 4 \\ 2 \\ 2 \\ 2 \end{pmatrix} \qquad Sing\_vals = \begin{pmatrix} 6.9336 \\ 4.1813 \\ 3.1623 \\ 3.1623 \end{pmatrix}$$