|  |  |  |  |
| --- | --- | --- | --- |
| IST_logo | ***Industrial Processes Automation***  *MSc in Electrical and Computer Engineering*  *Scientific Area of Systems, Decision, and Control*  *Winter Semester 2016/2017* |  | *Group: A2*  *75167- Francisco Oliveira*  *75322-Catarina Gaspar*  *75637-Inês Lourenço*  *84993-Hugo Serieiro* |

***2nd Laboratory Assignment [[1]](#footnote-1)***

***Handling Faults in Keyboard Reading***

This laboratory assignment aims at studying Discrete Event Systems (DESs) in the aspects of modeling, analysis of properties and synthesis. Synthesis will be based on a recent methodology in the framework of supervised control. This assignment further develops the previous assignment in the keyboard reading component by introducing fault handling mechanisms.

The tools to be used in this work are MATLAB, including a supervisor design toolbox, and a Petri Net editor. In the last part of the work the Schneider PLCs will be used once more to validate the proposed methodologies.

# Part A - Petri Net

**DES modeling:**

Using as a base guideline the work done in the first laboratory assignment, it is now desired to model the process of reading one keyboard key as a DES. In particular is supposed to develop a Petri Net that describes the events and the state evolution of the key reading system. Complementing the formal definition of the Petri Net, it is desired also to obtain the corresponding incidence matrix.

Note: in order to obtain a Petri Net as simple as possible (less than 20 places), the rejection of multiple keys (first laboratory, part B) should not be included now; the rejection of multiple keys will be subject of formal analysis in part C of this assignment.

Note2: See in the course webpage tools helping this part of the assignment, namely the graphical editor "pmedit" (windows 32) which allows creating models to import with the Matlab toolbox "tpn5". Both tools are compressed in a single ZIP file "PN\_editor\_MATLAB\_sim\_and\_Manual.zip". See also Matlab functions simulating Petri nets, in particular the "5 Philosophers" demo and a template for simulating the keyboard "lab2\_sim\_kb\_v5.zip".

**Q1:** Write the list of events of the DES you are proposing. *(Fill the next table, adding as many lines as needed).*

|  |  |
| --- | --- |
| **Event Identifier** | **Description** |
| T1 | Fim do timer que mantém coluna 1 activa |
| T2 | Fim do timer que mantém coluna 2 activa |
| T3 | Fim do timer que mantém coluna 3 activa |
| T4 | Reconhece input da 1ª linha |
| T5 | Reconhece input da 2ª linha |
| T6 | Reconhece input da 3ª linha |
| T7 | Reconhece input da 4ª linha |
| T8 | Reconhece input da 1ª linha |
| T9 | Reconhece input da 2ª linha |
| T10 | Reconhece input da 3ª linha |
| T11 | Reconhece input da 4ª linha |
| T12 | Reconhece input da 1ª linha |
| T13 | Reconhece input da 2ª linha |
| T14 | Reconhece input da 3ª linha |
| T15 | Reconhece input da 4ª linha |
| T16 | Input da linha 1 desligado |
| T17 | Input da linha 2 desligado |
| T18 | Input da linha 3 desligado |
| T19 | Input da linha 4 desligado |
| T20 | Input da linha 1 desligado |
| T21 | Input da linha 2 desligado |
| T22 | Input da linha 3 desligado |
| T23 | Input da linha 4 desligado |
| T24 | Input da linha 1 desligado |
| T25 | Input da linha 2 desligado |
| T26 | Input da linha 3 desligado |
| T27 | Input da linha 4 desligado |

**Q2:** Write the list of conditions of the DES you are proposing. *(Fill the next table, adding as many lines as needed).*

|  |  |
| --- | --- |
| **Condition Identifier** | **Description** |
| P1 | Coluna 1 está activa |
| P2 | Coluna 2 está activa |
| P3 | Coluna 3 está active |
| P4 | Tecla ‘1’ está pressionada |
| P5 | Tecla ‘4’ está pressionada |
| P6 | Tecla ‘7’ está pressionada |
| P7 | Tecla ‘\*’ está pressionada |
| P8 | Tecla ‘2’ está pressionada |
| P9 | Tecla ‘5’ está pressionada |
| P10 | Tecla ‘8’ está pressionada |
| P11 | Tecla ‘0’ está pressionada |
| P12 | Tecla ‘3’ está pressionada |
| P13 | Tecla ‘6’ está pressionada |
| P14 | Tecla ‘9’ está pressionada |
| P15 | Tecla ‘#’ está pressionada |

**Q3:** Write the table of pre and post conditions for each of the events. *(Fill the next table, adding as many lines as needed).*

|  |  |  |
| --- | --- | --- |
| **Event** | **Pre-Conditions** | **Post-Conditions** |
| T1 | P1 | P2 |
| T2 | P2 | P3 |
| T3 | P3 | P1 |
| T4 | P1 | P4 |
| T5 | P1 | P5 |
| T6 | P1 | P6 |
| T7 | P1 | P7 |
| T8 | P2 | P8 |
| T9 | P2 | P9 |
| T10 | P2 | P10 |
| T11 | P2 | P11 |
| T12 | P3 | P12 |
| T13 | P3 | P13 |
| T14 | P3 | P14 |
| T15 | P3 | P15 |
| T16 | P4 | P1 |
| T17 | P5 | P1 |
| T18 | P6 | P1 |
| T19 | P7 | P1 |
| T20 | P8 | P2 |
| T21 | P9 | P2 |
| T22 | P10 | P2 |
| T23 | P11 | P2 |
| T24 | P12 | P3 |
| T25 | P13 | P3 |
| T26 | P14 | P3 |
| T27 | P15 | P3 |

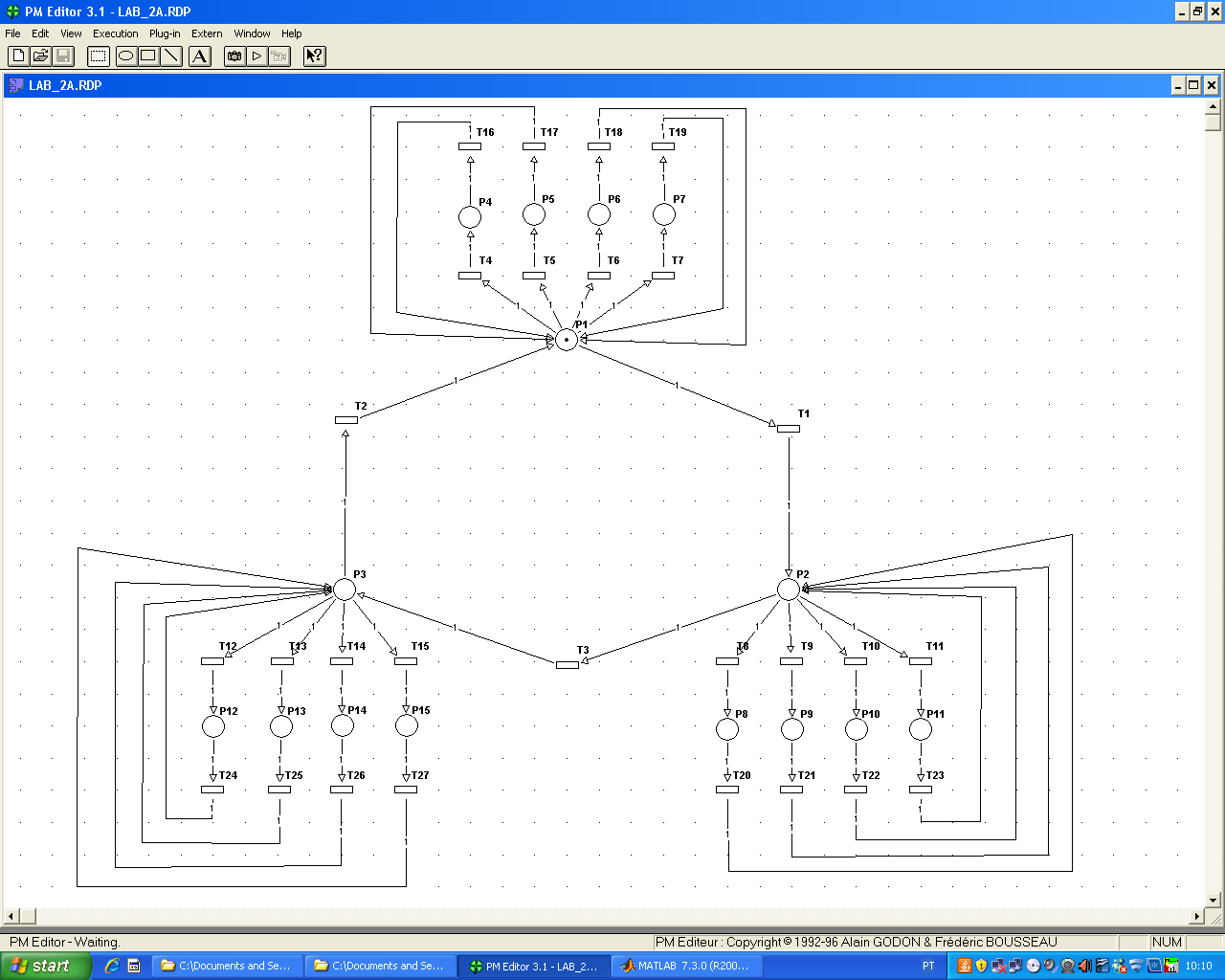
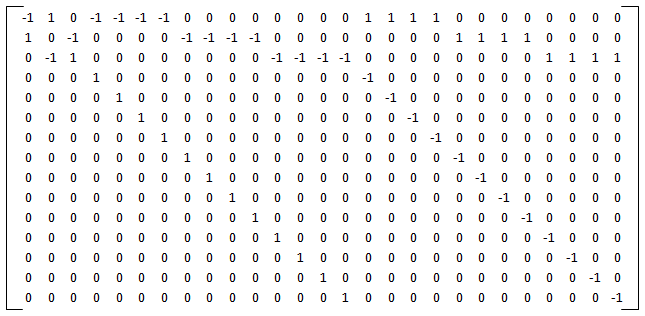
**Q4:** Draw the corresponding Petri Net. Comment about the temporizations implemented in the first assignment and which are not currently included in the proposed DES.

Figura 1 - Petri Net que implementa keyboard

**Q5:** Obtain the incidence matrix, D=D+-D-, describing the proposed Petri Net.



**Q6.1:** Run the default "5 philosophers dinner" simulation referred in Annex 1. Comment whether all *requests to eat* are satisfied or not. Modify function PN\_tfire.m to simulate random inputs, as detailed in Annex 1, and comment again on the *satisfied requests to eat* and, in particular, whether one philosopher can be *more satisfied* than others.

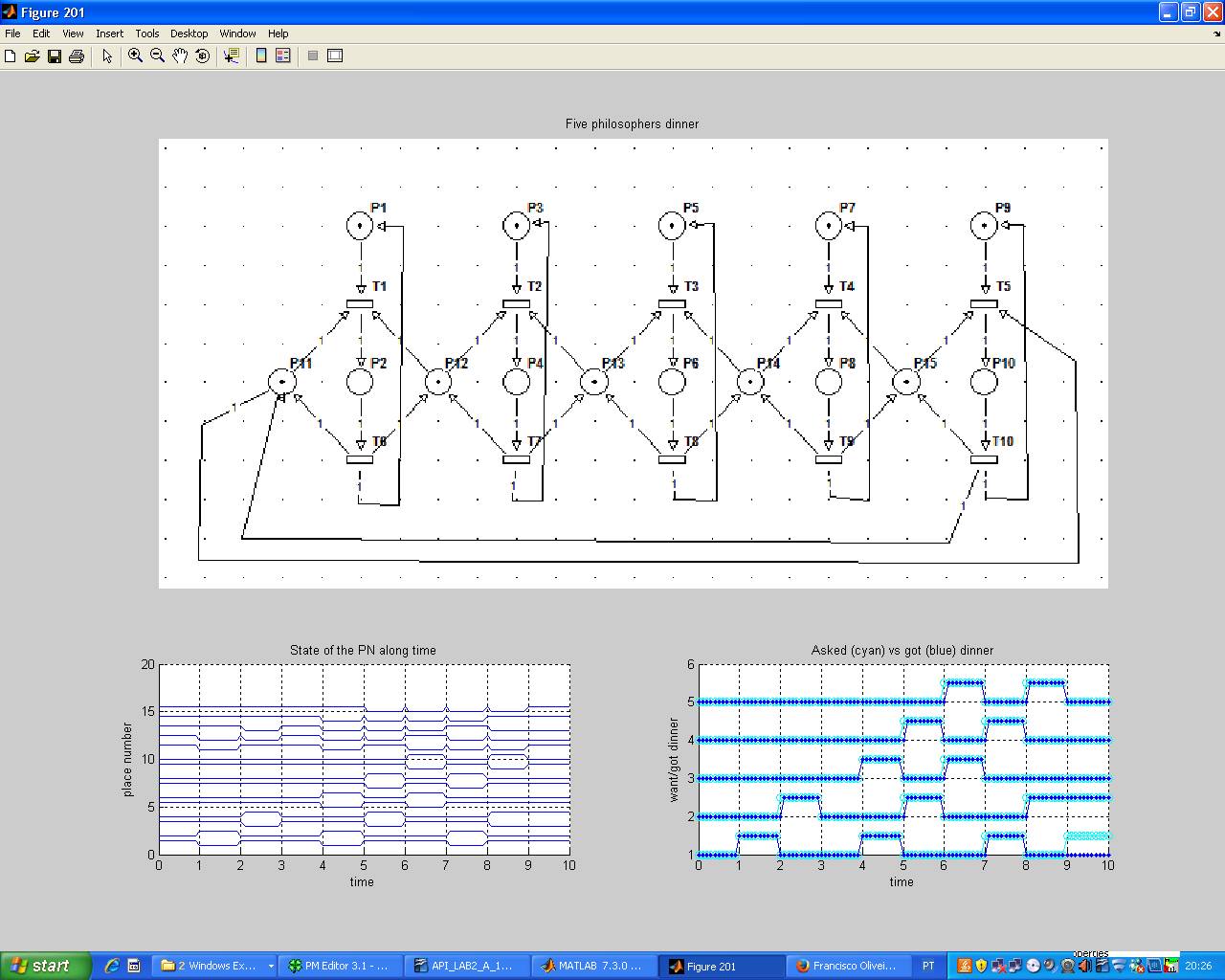


Figura 2 – Simulação “5 philosophers dinner”

# Nem todos os pedidos dos filósofos foram satisfeitos tal como pode ser visto na figura 2, em que o ultimo pedido do filósofo 1 não pôde ser satisfeito pois o filósofo 2 está a usar a faca. Em termos gerais, não é possível que dois filósofos que estejam sentados consecutivamente possam comer ao mesmo tempo.

# Após alterar a simulação para inputs aleatórios obteve-se os seguintes resultados:

# 15065032_1259520990785610_305546922_o.png

Figura 3 – Simulação “5 philosophers dinner” com inputs aleatórios

De novo como na simulação anterior houve vários filósofos que não viram os seus pedidos satisfeitos. Através dos resultados da simulação apresentados na figura 3, nota-se que tanto o primeiro fiósofo como o terceiro têm menos discrepância entre os seus pedidos (azul claro) e satisfação dos mesmos (azul escuro). Como resultado disto, tem-se que os filósofos 2, 4 e 5 automaticamente terão menos pedidos satisfeitos pois estão sentados directamente ao lado do 1 e do 3. Após a realização de várias experiências, conclui-se que esta tendência se repete levando a crer que a ordem de criação dos vários eventos influencie a presença de um token nesse evento.

**Q6.2:** Simulate the Petri net you proposed, as requested in this guide, and comment whether the simulated Petri net may have or not privileged transitions or places while receiving random inputs.

Ao simular no Matlab a Petri net implementada, testou-se em primeiro lugar o seu funcionamento para uma sequência predefinida de inputs das teclas, definida no ficheiro “PN\_device\_kb\_IO.m”:

0 mk\_keys([]) ; ...

1 mk\_keys(1) ; ...

2 mk\_keys([]) ; ...

3 mk\_keys(5) ; ...

4 mk\_keys([]) ; ...

5 mk\_keys(9) ; ...

6 mk\_keys([]) ; ...

7 mk\_keys([9 12]) ; ...

8 mk\_keys(12) ; ...

9 mk\_keys([]) ; ...

O funcionamento ideal do keyboard inclui impedir que duas teclas sejam pressionadas ao mesmo tempo, pelo que nesta sequência, em que se carrega em duas ao mesmo tempo e no instante seguinte se larga uma, o programa deveria inicialmente mostrar ou a tecla 9 ou a 12 activas, e em seguida apenas a 12.

O resultado da simulação com o código implementado foi o seguinte:

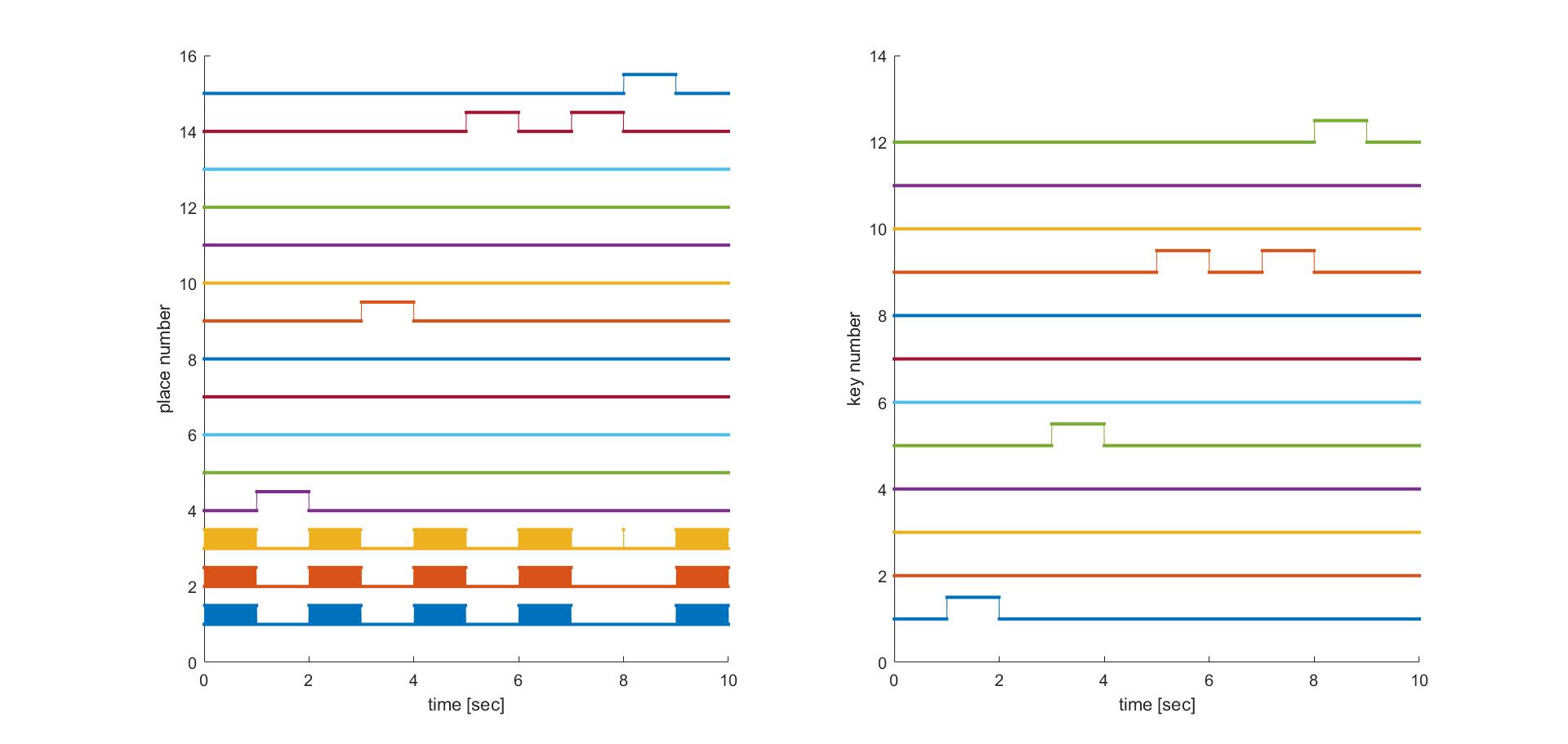


Figura 4 – Simulação do funcionamento do teclado com uma sequência pré-definida de teclas pressionadas

No gráfico da esquerda da figura 4 está representada a sequência de lugares activos, e no da direita a sequência das teclas correspondentes a cada estado. Por exemplo, no primeiro segundo os 3 lugares estão sucessivamente a alterar entre si na ordem predefinida, que equivale comutação das colunas, e passado o 1º segundo o *token* passa para o estado P4, que o gráfico da direita indica como sendo o estado correspondente à tecla número 1. O código foi implementado da seguinte maneira:

O vector *act* tem 3 posições, e cada uma está activa quando a coluna correspondente está activa sem nenhuma tecla pressionada, ou com qualquer tecla dessa coluna pressionada. Estando num lugar de coluna activa, quando uma tecla dessa coluna é pressionada a linha é guardada na variável lines. Se, por exemplo a segunda linha e a primeira coluna estiverem activas, a transição T5 vai activar para o token passar para o lugar P5 (tecla 4) e as transições de saída das outras teclas (T16, T18 e T19) vão activar também. Isto é feito para que se garanta que um token não possa estar nos lugares de teclas que não sejam a que foi pressionada. Quando não há nenhuma linha reconhecida então as transições de saída das teclas e das colunas são activas de forma a garantir que o token sai dos lugares das teclas, e caso já não esteja num destes lugares, para que transite para o lugar da próxima coluna.

Sendo assim, verifica-se que de facto a Petri Net funciona e corre o programa exactamente como seria de esperar, tendo a correcta sequência de teclas sido efectuada. De notar que no caso em que tanto a tecla 9 e 12 são pressionadas o token vai para o lugar P14 que está associado à tecla 9. Isto acontece pois a Petri Net dá prioridade a transições criadas primeiro, logo como a transição T14 foi criada antes da T15 este escolhe o T14 que leva ao lugar P14.

Por outro lado, de modo a, em vez de se atribuir uma sequência pre-definida de inputs estes forem escolhidos aleatoriamente, introduz-se uma sequência aleatória de transições, que simulam o acto de carregar numa sequência de teclas aleatórias, tal como foi feito para o exemplo dos filósofos. O comando dado foi:

qk= [round(rand(1,27))];

Note-se que, para além disto e de modo a garantir que o funcionamento da Petri net representa exactamente o funcionamento do keyboard, em vez de pôr todas as transições aleatórias resolveu-se garantir que as transições entre colunas são sempre activas quando nenhuma teclas nessa coluna é pressionada. A diferença é que sem este pormenor poderia acontecer que, por vezes, uma coluna estaria activa durante dois instantes seguidos, o que não é suposto e foi então corrigido.

O funcionamento passa a ser o seguinte:

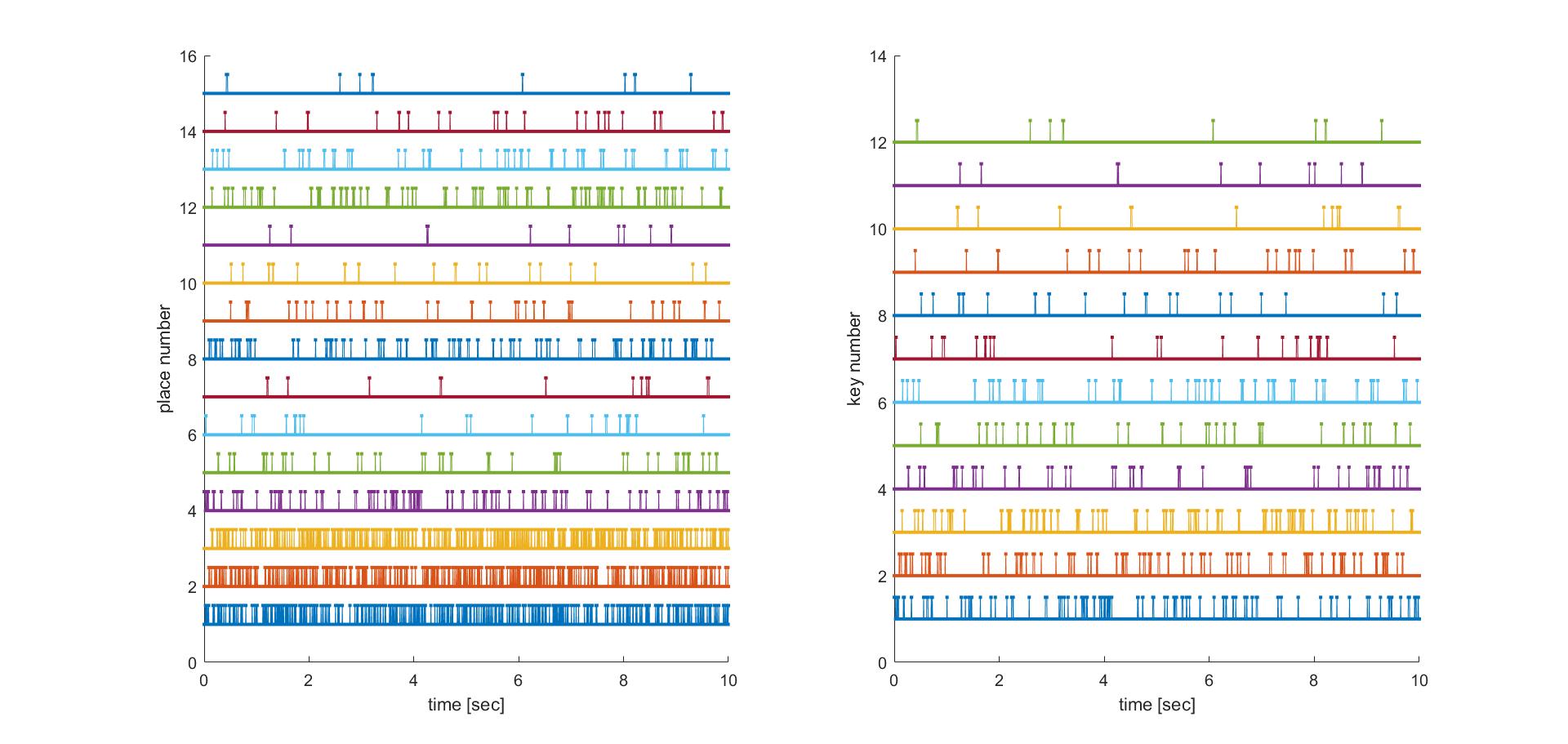


Figura 5 – Simulação do funcionamento do teclado com inputs aleatórios

Analisando o gráfico do “place number” pode-se concluir que, ao simular a petri net com inputs aleatórios, como era esperado as colunas são os lugares mais frequentemente activados (mais visitados pelo token). Isto acontece porque, em primeiro lugar, são os 3 lugares que têm o maior número de transições a chegar a eles, e, em segundo lugar, pelo facto das transições entre estes lugar terem sido as primeiras a serem criadas: isto é, caso todas as transições estejam activas, o token de um lugar de uma coluna irá sempre para o lugar da coluna seguinte, em vez de ir para uma tecla dessa coluna.

Isto justifica também, que nos restantes lugares haja uma prioridade pelas primeiras transições a serem criadas. Por exemplo, do lugar da primeira coluna caso a transição T4 esteja activa o token passará sempre para o lugar P4 (1ª linha), independentemente do estado das outras transições para as outras teclas. Só passará para outro lugar de outra tecla se esta transição estiver desligada. Isto é verificado em todas as colunas, significando que, para cada coluna activa, a ordem de prioridade das teclas é: A tecla da 1ª linha, depois a da 2ª, e assim sucessivamente.

Isto pode ser comprovado analisando o gráfico “key pressed” da figura 5, em que as teclas premidas mais frequentemente são respectivamente a 1,2,3, seguidas das 4,5,6, e assim sucessivamente.

# Annex 1 - Philosophers Dinner Simulation

Please find in the webpage of the course the "5 philosophers dinner" simulation files as a zip file. Decompress the zip file and run pdinner\_tst.m. Note that you need also the RDP.M function found in the tpn5 toolbox that you have to install beforehand.

The pdinner\_tst.m demonstration script loads a Petri net model shown in figure1 and built using PMEDIT. The Petri net model is read using the referred RDP.M function.

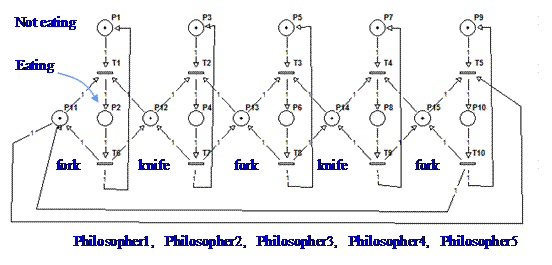


Figure 1: Petri net model representing the 5 philosophers dinner problem.

The default simulation has the transitions scheduled, driven by a time table. In order to run a simulation where the transitions are driven randomly, you can edit the file PN\_tfire.m and replace the line:

qk= pdinner\_IO(act, t);

by the line:

qk= round(rand(1,5)); qk= [qk not(qk)];

Note that only transitions 1 till 5 are computed randomly. Transitions 6 till 10 are defined simply as the negation of the transitions 1 till 5.

To have more information on the Petri net simulator and the specific simulation of the "5 philosophers dinner" see:

http://users.isr.ist.utl.pt/~jag/course\_utils/13\_pn\_sim/PN\_sim.html

1. Original guide by Prof. Paulo J. Oliveira. Revised by Prof. José Gaspar (2016). [↑](#footnote-ref-1)