Sisteme de Control

Andrei Bragarenco

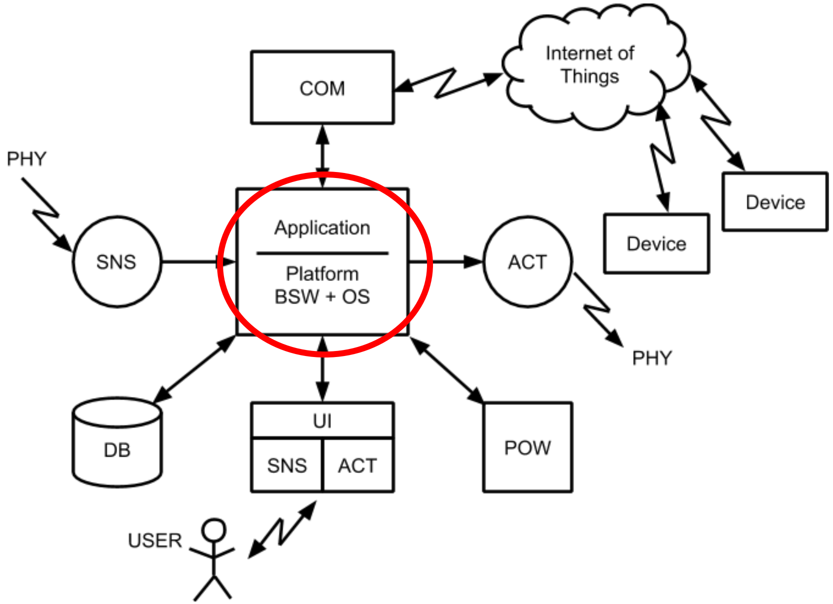
Clubul Ingineresc Micro Lab

IoT 2020 Bootcamp

Sisteme de control

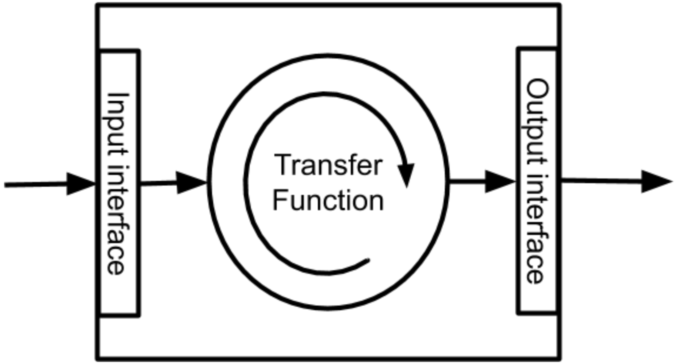
Control – o abstracție ce descrie o soluție a unei probleme.

- **Bucla Deschisa**
- **On/Off**
- **PID**
- **Automate Finite**
- *Control Fuzzy*
- *Retele Neuronale*
- *Machine Learning*



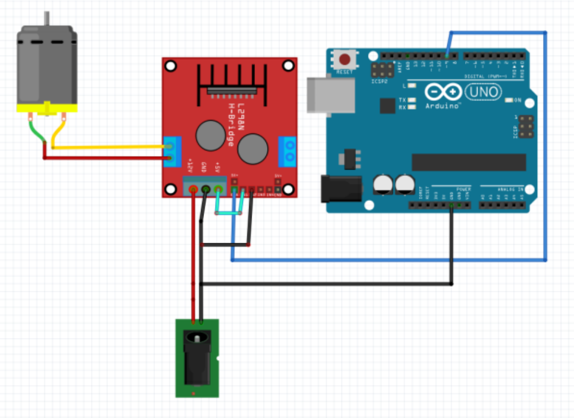
Control Functional

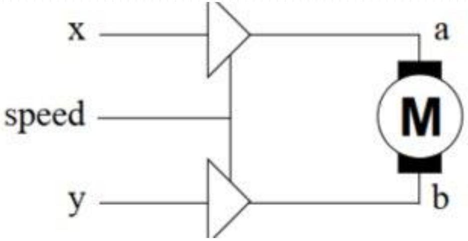
Definește comportamentul sistemului ca reacție la intrările sistemului si produce ieșiri corespunzătoare sub forma de functie

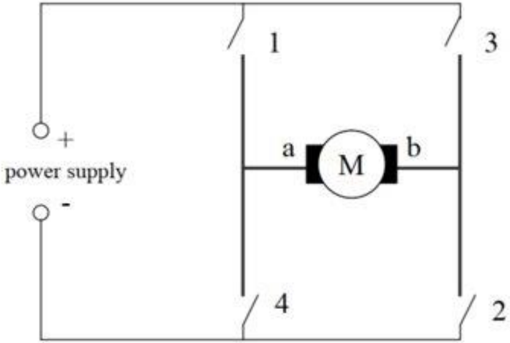


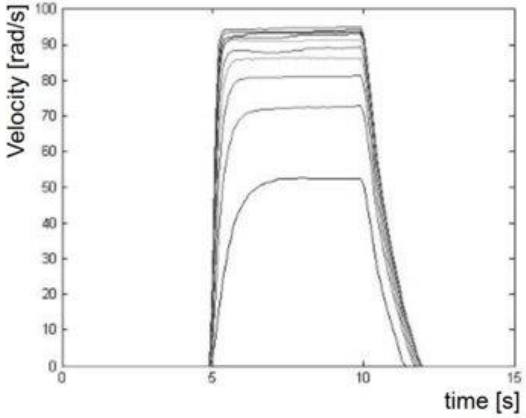
- Bucla Deschisa
- On/Off
- On/Off cu histereză
- PID

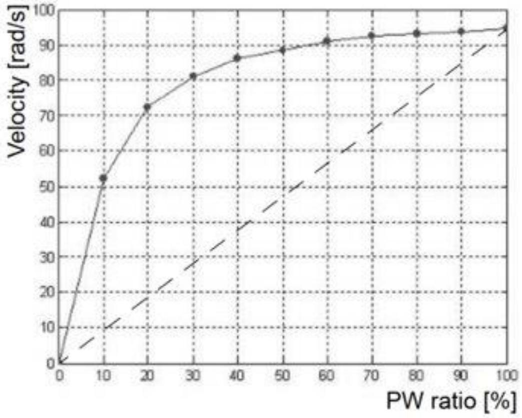
Control In bucla deschisa



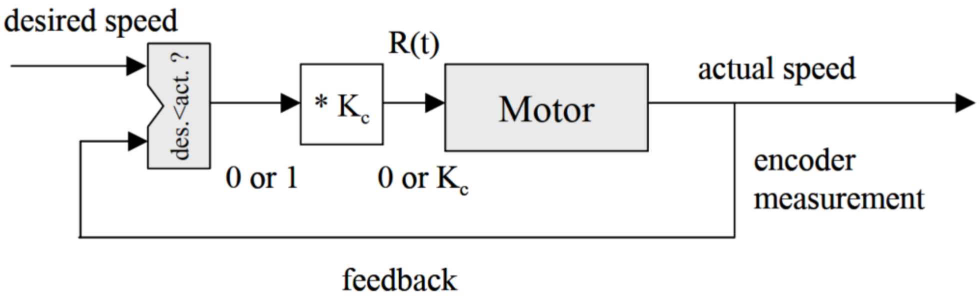


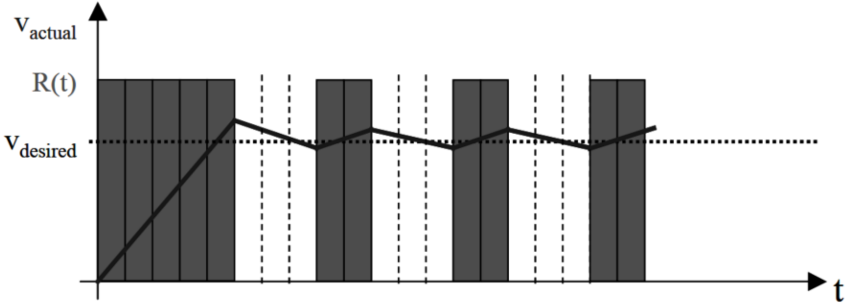




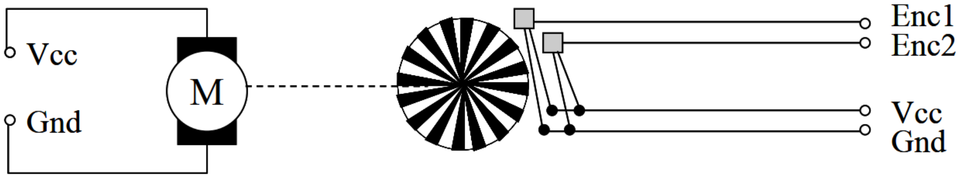


On-Off Control



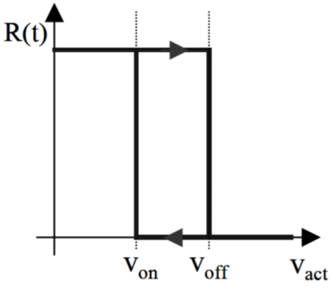


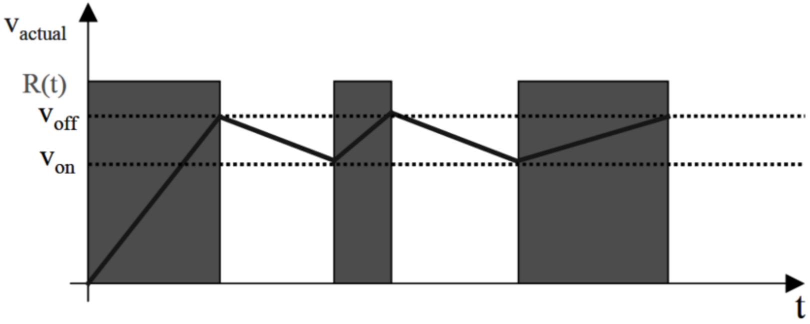
$$R(t) = \begin{cases} K_C & \text{if } v_{act}(t) < v_{des}(t) \\ 0 & \text{otherwise} \end{cases}$$

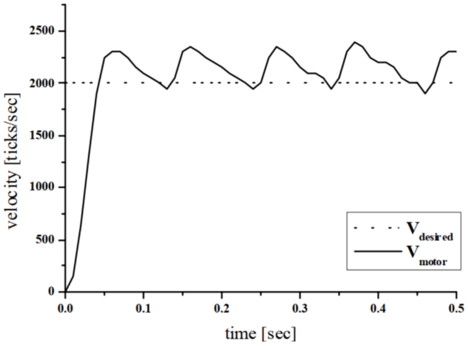


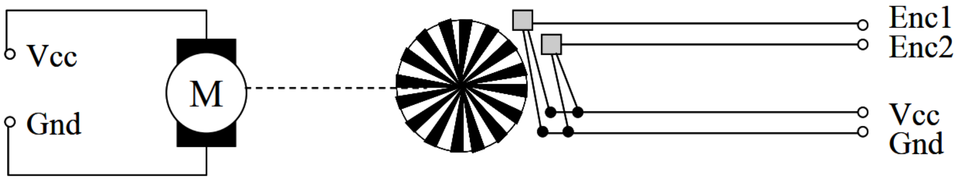
On-Off Control cu histereză

$$R(t+\Delta t) = \begin{cases} K_C & \text{if} \quad v_{act}(t) < v_{on}(t) \\ 0 & \text{if} \quad v_{act}(t) > v_{off}(t) \\ R(t) & \text{otherwise} \end{cases}$$

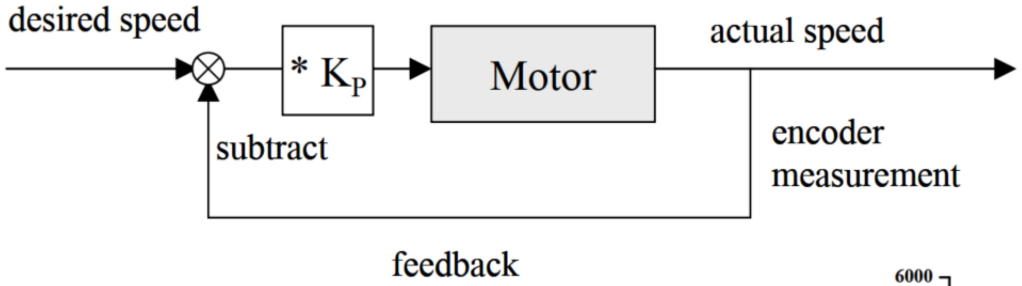








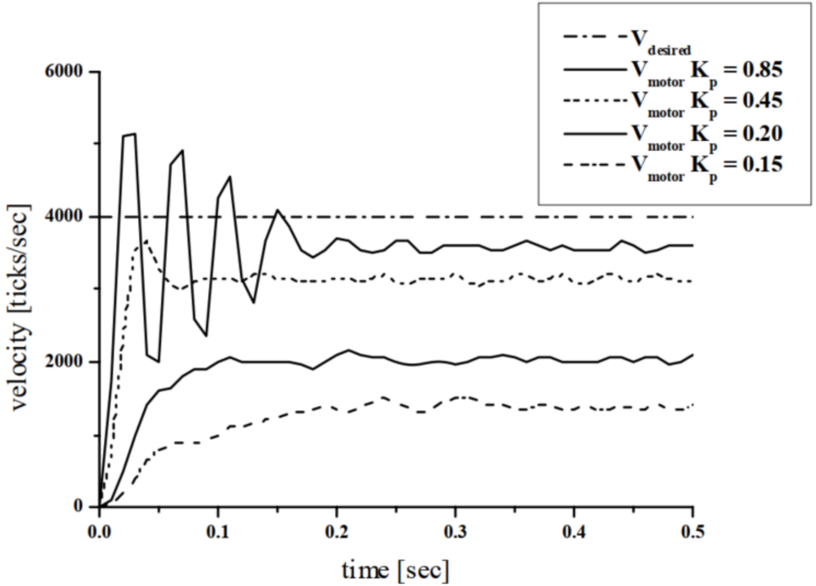
PID Control – Proporțional



P - Proporțional

$$e = (V_{des} - V_{act})$$

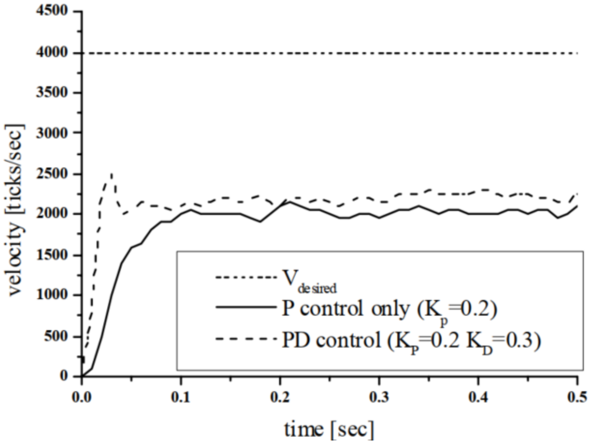
$$R_P(t) = K_P \cdot e(\mathbf{t})$$



PID Control

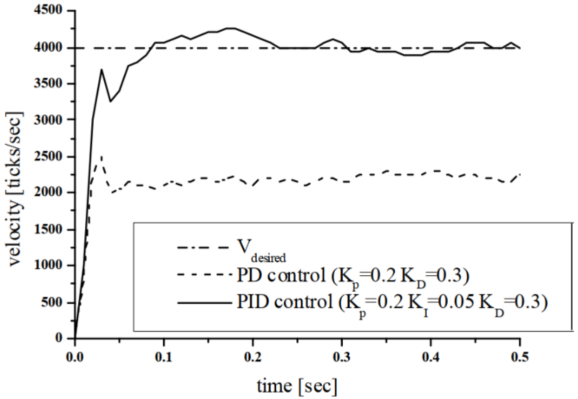
D - Diferențial

$$R_D(t) = K_D \cdot \frac{de}{dt} = K_D \cdot \frac{e(\mathrm{t}) - e(\mathrm{t} - \Delta t)}{\Delta t}$$



I - Integral

$$R_I(t) = K_I \cdot \int_0^t e(t)\,dt = K_I \cdot \sum_{k=0}^{n} e_k \cdot \Delta t$$



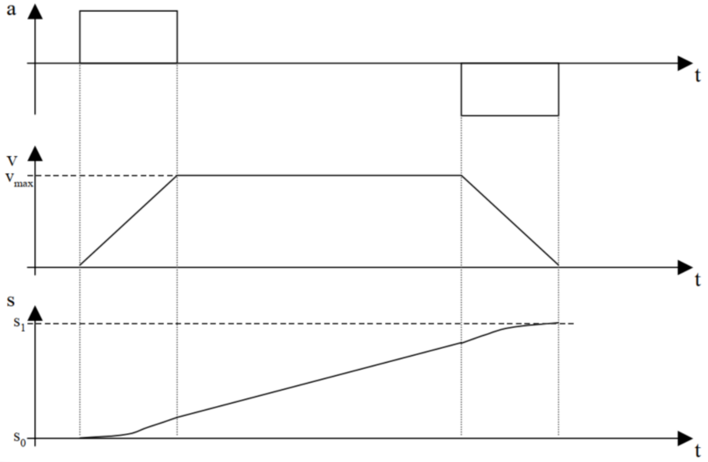
$$R_{PID}(t) = R_P(t) + R_I(t) + R_D(t) = K_P \cdot e(\mathrm{t}) + K_I \cdot \sum_{k=0}^{n} e_k \cdot \Delta t + K_D \cdot \frac{e(\mathrm{t}) - e(\mathrm{t} - \Delta t)}{\Delta t}$$

PID - Tuning

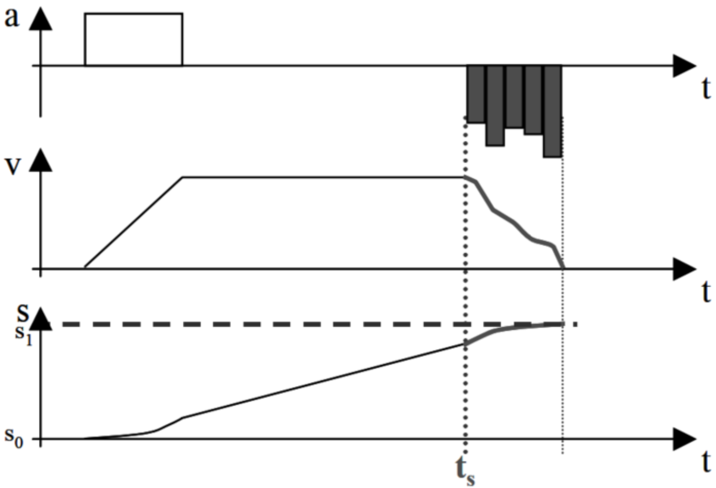
1. Setare *Vd* dorita , *Kp = 0, Ki = 0, Kd = 0*
2. Mărire *Kp* pana la oscilare. împărțire *Kp/2*
3. Mărire Kd pana se observa creștere cu 5-10%
4. Mărire *Ki* pana la oscilare, împărțire *Ki/2* sau *Ki /3*
5. Verificare cu diverse valori *Vd*

Control evoluție si poziție

pornire / oprire lină



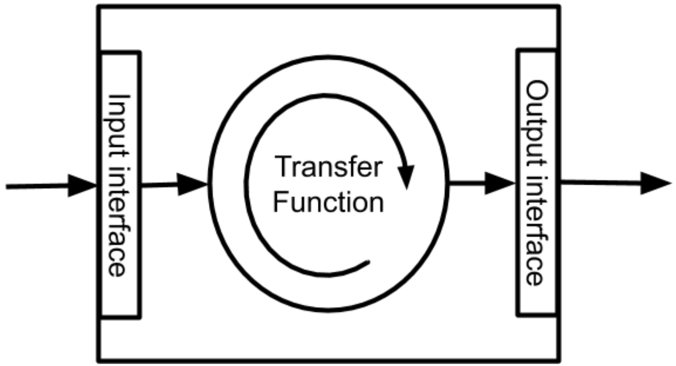
adaptare frânare



Control cu Automate Finite - FSM

Automat Finit este o abstracție ce descrie o soluție a unei probleme.

Definește comportamentul sistemului sub forma de mecanism care își schimba stările ca reacție la intrările sistemului si produce ieșiri corespunzătoare

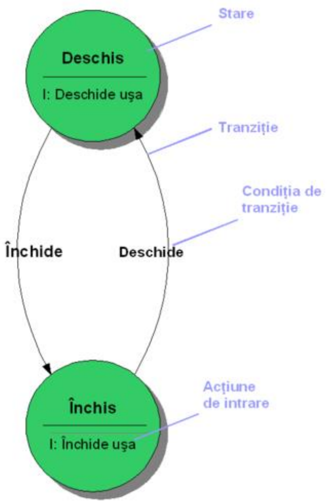


En: Finite State Mashine

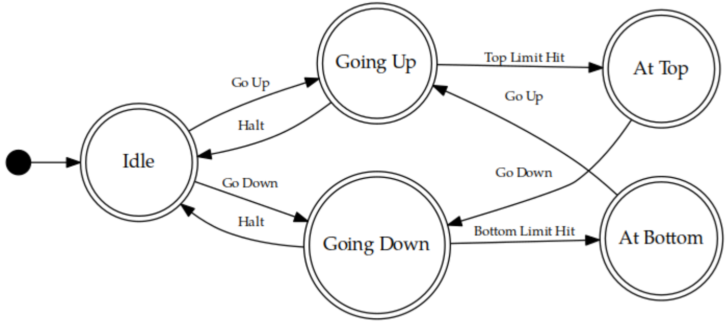
1. Număr finit de stări, *una este definita ca si inițiala*
2. Număr finit de intrări in sistem
3. Număr finit de ieșiri generate.
4. Funcție de transfer pentru tranzițiile dintre stări.
5. Funcție de definire a ieșirilor

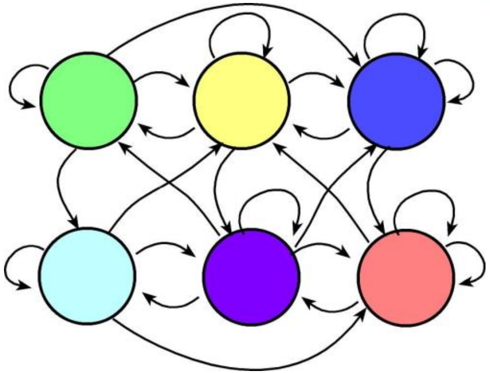
Automate Finite - Diagrame

"mașină cu un număr finit de stări" este un model de comportament compus din stări, tranziții și acțiuni.



- ***O stare*** stochează informații despre trecut, adică reflectă schimbările intrării de la inițializarea sistemului până în momentul de față.
- ***O tranziție*** indică o schimbare de stare și este descrisă de **o condiție** care este nevoie să fie îndeplinită pentru a declanșa tranziția.
- ***O acțiune*** este o descriere a unei activități ce urmează a fi executată la un anumit moment.

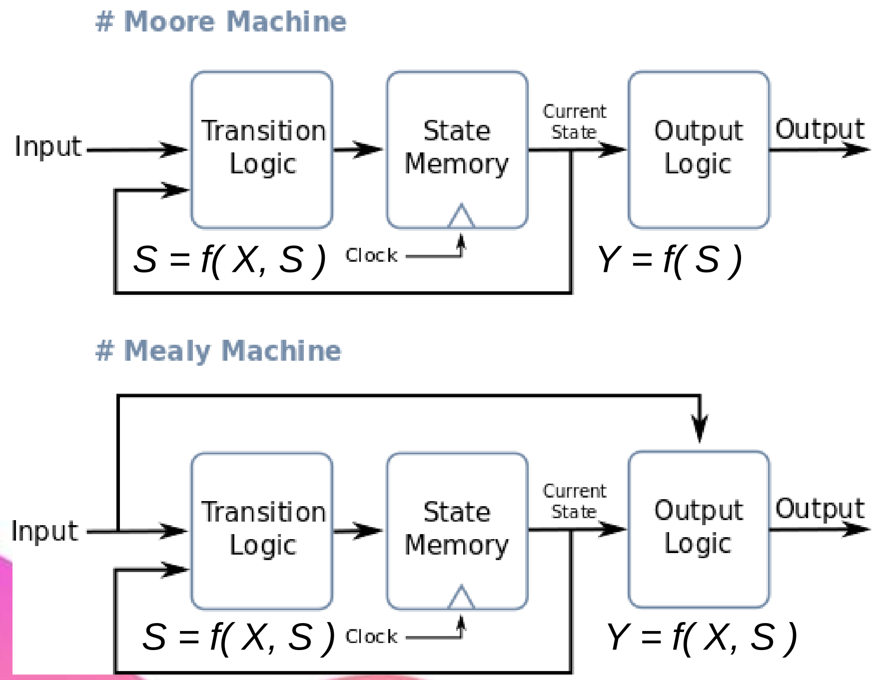


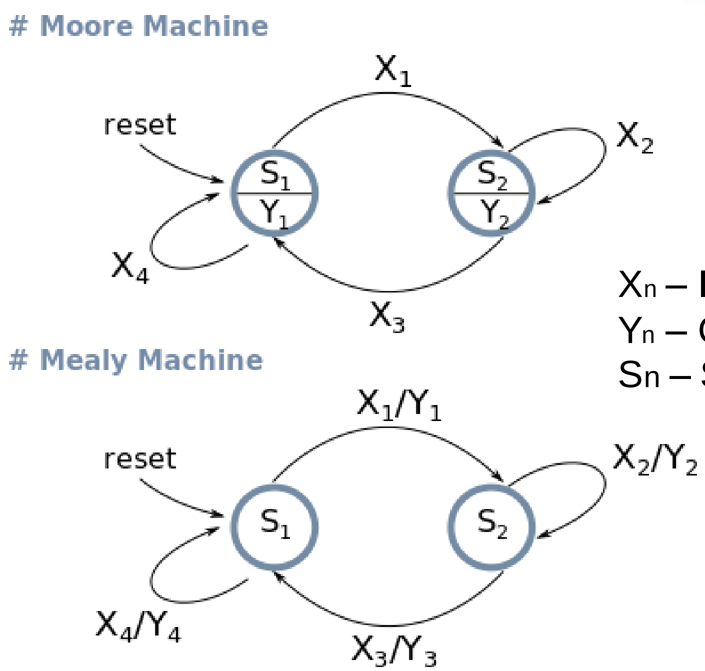


https://ro.wikipedia.org/wiki/Automat_finit

Google Search: "FSM Diagramm example"

Automate Finite – Mealy vs Moore





X_n – Inputs
Y_n – Outputs
S_n – States

http://calangorobo.blogspot.com/2018/

Automate Finite - Evaluare

Automat Moore

NextState = f(Input, CurrentState)

Output = g(CurrentState)

TaskMooreFSM(){

1. ***Evaluare Ieșiri***, dependente **doar** de ***Starea curenta***
2. ***Reținere*** perioada definita de stare
3. ***Colectare Intrări***
4. ***Evaluare Stare următoare*** dependenta de ***Intrare*** si ***Stare curentă***

}

Automat Mealy

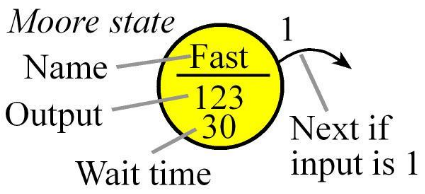
NextState = f(Input, CurrentState)

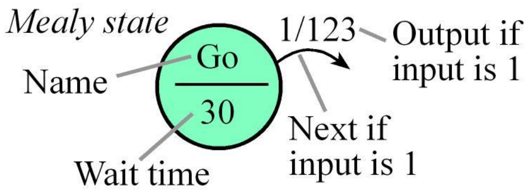
Output = h(Input, CurrentState)

TaskMealyFSM()

1. ***Reținere*** perioada definita de stare
2. ***Colectare Intrări***
3. ***Evaluare Ieșiri***, dependente de ***Intrare*** si ***Starea curentă***
4. ***Evaluare Stare următoare*** dependenta de ***Intrare*** si ***Stare curentă***

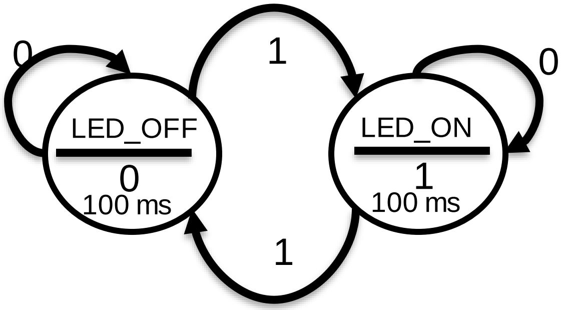
}

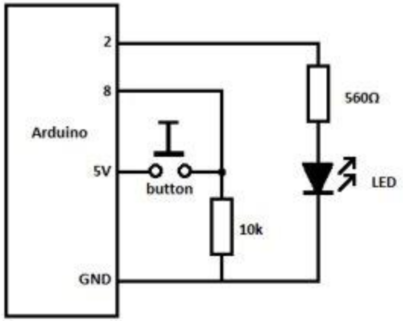




https://courses.edx.org/courses/course-v1:UTAustinX+UT.6.03x+1T2016

Automate Finite – Buton/Led : FSM design





Num	Name	Out	Delay	In = 0	In = 1
0	LED_OFF	0	100 ms	LED_OFF	LED_ON
1	LED_ON	1	100 ms	LED_ON	LED_OFF

```
#define LED_OFF_STATE 0
#define LED_ON_STATE 1

struct State {
    unsigned long Out;     // Led State
    unsigned long Time;    // delay in 10ms units
    unsigned long Next[2]; // next state for inputs 0,1
};
typedef const struct State STyp;

STyp FSM[2]={
    {0,10,{LED_OFF_STATE, LED_ON_STATE }},
    {1,10,{LED_ON_STATE,  LED_OFF_STATE }}
};
```

https://courses.edx.org/courses/course-v1:UTAustinX+UT.6.03x+1T2016

Automate Finite – Button/Led : FSM Controller

```c
#define LED_PIN 2
#define BUTTON_PIN 8

#define LED_OFF_STATE  0
#define LED_ON_STATE   1

struct State {
    unsigned long Out;    // Led State
    unsigned long Time;   // delay in 10ms units
    unsigned long Next[2];// next state for inputs 0,1
};
typedef const struct State STyp;

STyp FSM[2]={
        {0,10,{LED_OFF_STATE, LED_ON_STATE  }},
        {1,10,{LED_ON_STATE,  LED_OFF_STATE }}
};

int FSM_State = LED_OFF_STATE;

void setup()
{
    // Init Button
    pinMode(BUTTON_PIN, INPUT);
    // Init LED
    pinMode(LED_PIN, OUTPUT);
    // Init Initial State
    FSM_State = LED_OFF_STATE;
}
```

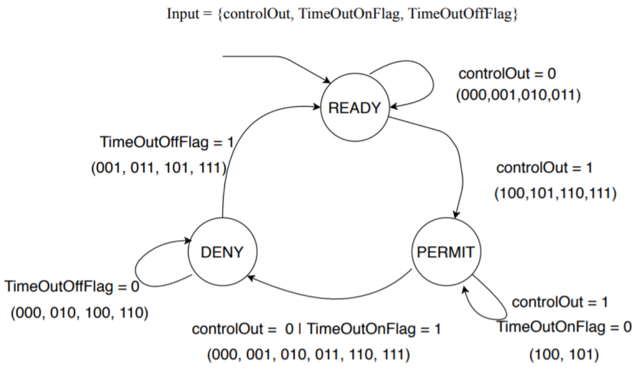
TaskMooreFSM(){

1. ***Evaluare ieșiri***, dependente **doar** de ***Starea curenta***
2. ***Reținere*** perioada definita de stare
3. ***Colectare Intrări***
4. ***Evaluare Stare următoare*** dependenta de ***Intrare*** si ***Stare curentă***

}

```c
// The loop function is called in an endless loop
void loop()
{
    // 1. Output Based on current state
    int output = FSM[FSM_State].Out;
    digitalWrite(LED_PIN, output);

    // 2. wait for time relevant to state
    delay(FSM[FSM_State].Time * 10);

    // 3. Get Input
    int input = digitalRead(BUTTON_PIN);

    // 4. Change state based on input and current state
    FSM_State = FSM[FSM_State].Next[input];

}
```

Automate Finite – Motor Control



Input = {controlOut, TimeOutOnFlag, TimeOutOffFlag}

Nr	Name	Out	Delay	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
1	Ready	0	100ms	Ready	Ready	Ready	Ready	Permit	Permit	Permit	Permit
2	Permit	1	100ms	Deny	Deny	Deny	Deny	Permit	Permit	Deny	Deny
3	Deny	2	100ms	Deny	Ready	Deny	Ready	Deny	Ready	Deny	Ready

```
#define READY 0
#define PERMIT 1
#define DENY 2

struct State {
    unsigned int Out;          // Automata state
    unsigned int Time;         // delay in 10ms Units
    unsigned int Next[8];      // next sate for inputs 0..7
};

typedef struct State STyp;

STyp FSM[3] = {
  {0, 10, {READY,  READY, READY, READY, PERMIT, PERMIT, PERMIT, PERMIT}},
  {1, 10, {DENY,   DENY,  DENY,  DENY,  PERMIT, PERMIT, DENY,   DENY}},
  {2, 10, {DENY,   READY, DENY,  READY, DENY,   READY,  DENY,   READY}}
};

int FSM_State = READY;
```

Clubul Ingineresc Micro Lab
IoT 2020 Bootcamp

Automate Finite – Motor Control

```c
#define READY 0
#define PERMIT 1
#define DENY 2

struct State {
    unsigned int Out;       // Automata state
    unsigned int Time;      // delay in 10ms Units
    unsigned int Next[8];   // next sate for inputs 0..7
};

typedef struct State STyp;

STyp FSM[3] = {
  {0, 10, {READY,  READY, READY, READY, PERMIT, PERMIT, PERMIT, PERMIT}},
  {1, 10, {DENY,   DENY,  DENY,  DENY,  PERMIT, PERMIT, DENY,   DENY}},
  {2, 10, {DENY,   READY, DENY,  READY, DENY,   READY,  DENY,   READY}}
};

int FSM_State = READY;

int automatInput(int controlOut, int TimeOutOnFlag, int TimeOutOffFlag) {
    int result;

    if (controlOut == 0 && TimeOutOnFlag == 0 && TimeOutOffFlag == 0) {
        result = 0b000;
    } else if (controlOut == 0 && TimeOutOnFlag == 0 && TimeOutOffFlag == 1) {
        result = 0b001;
    } else if (controlOut == 0 && TimeOutOnFlag == 1 && TimeOutOffFlag == 0) {
        result = 0b010;
    } else if (controlOut == 0 && TimeOutOnFlag == 1 && TimeOutOffFlag == 1) {
        result = 0b011;
    } else if (controlOut == 1 && TimeOutOnFlag == 0 && TimeOutOffFlag == 0) {
        result = 0b100;
    } else if (controlOut == 1 && TimeOutOnFlag == 0 && TimeOutOffFlag == 1) {
        result = 0b101;
    } else if (controlOut == 1 && TimeOutOnFlag == 1 && TimeOutOffFlag == 0) {
        result = 0b110;
    } else if (controlOut == 1 && TimeOutOnFlag == 1 && TimeOutOffFlag == 1) {
        result = 0b111;
    }
    return result;
}
```

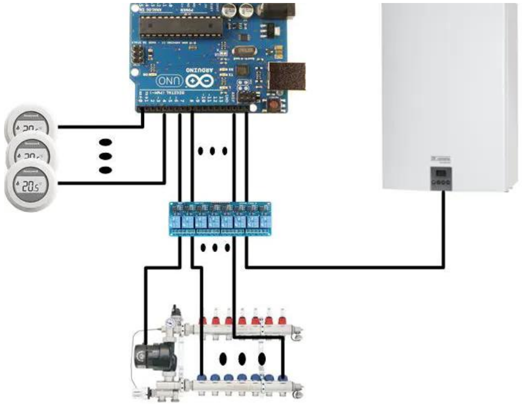
TaskMooreFSM(){

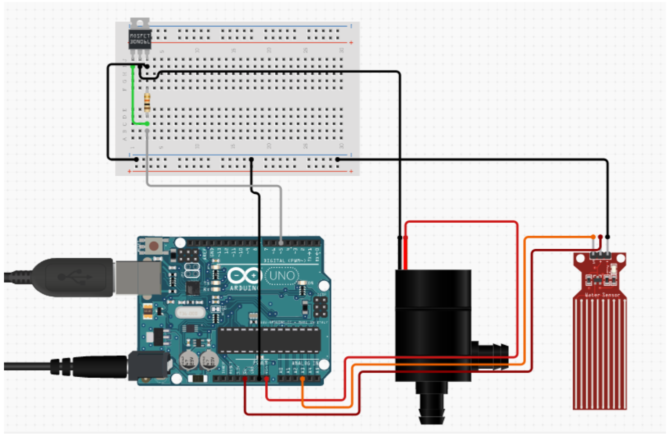
1. ***Evaluare ieșiri***, dependente **doar** de ***Starea curenta***
2. ***Reținere*** perioada definita de stare
3. ***Colectare Intrări***
4. ***Evaluare Stare următoare*** dependenta de ***Intrare*** si ***Stare curentă***

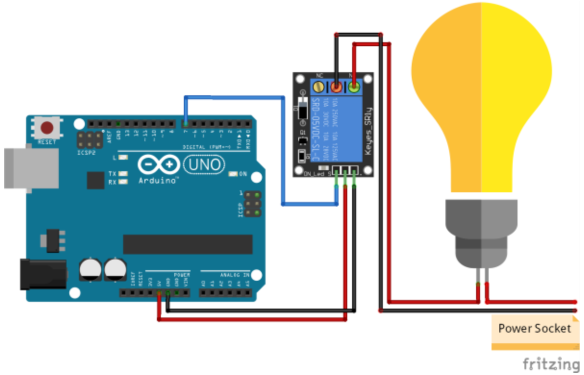
}

```c
int automatProcess(int controlOut, int TimeOutOnFlag, int TimeOutOffFlag) {

    // 1. Output Based on current State
    int output = FSM[FSM_State].Out;

    // 2. Wait for time relevant to state
    delay(FSM[FSM_State].Time * 10);

    // 3. Get Input
    int input = automatInput(controlOut, TimeOutOnFlag, TimeOutOffFlag);

    // 4. Change State based in input and current state
    FSM_State = FSM[FSM_State].Next[input];

    return output;
}
```

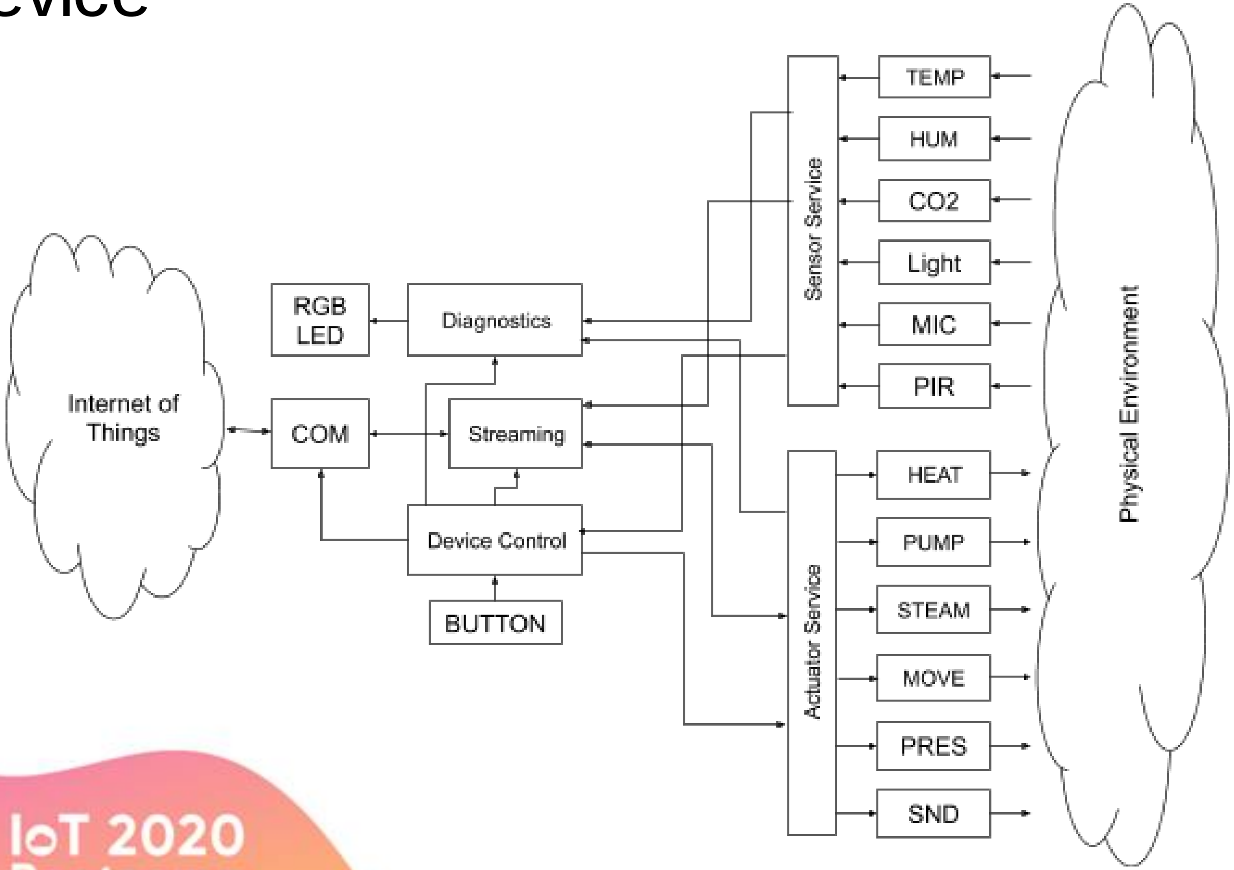
Acțiune - Temperatura







IoT Device



Internet of Things
RGB LED
Diagnostics
COM
Streaming
Device Control
BUTTON
Sensor Service
TEMP
HUM
CO2
Light
MIC
PIR
Actuator Service
HEAT
PUMP
STEAM
MOVE
PRES
SND
Physical Environment

Mulțumesc pentru atentie

Întrebari?

Clubul Ingineresc Micro Lab

IoT 2020 Bootcamp