

7. Aplicații

Stație meteo

Această secțiune își propune să prezinte mai multe aplicații practice care să exemplifice utilizarea cunoștințelor acumulate pe parcursul capitolelor anterioare.

7.1 Stație meteo

Această aplicație presupune utilizarea mai multor senzori, pentru a se realiza o stație meteo. Pentru aceasta se va folosi un senzor digital de temperatură și umiditate (DHT22) cât și un senzor analogic de ploaie. Acești senzori se vor conecta la o placă de dezvoltare Arduino Uno. Rezultatul se poate vizualiza pe orice dispozitiv de afișare prezentat în laboratoarele anterioare.

7.1.1 Utilizarea senzorului DHT22

Senzorul digital DHT22 este un senzor care permite măsurarea relativă a temperaturii și a umidității. Pentru a se realiza măsurarea umidității se utilizează un senzor capacitiv, iar pentru a măsura temperatura un termistor. Ambele componente sunt împachetate într-un chip cu 4 pini. Pinii corespunzători sunt cei din figura 7.1. Primul pin reprezintă sursa de alimentare. Aceasta trebuie să fie una continuă, cu valori între 3-5V. Pinul 2 este pinul de date. Senzorul comunică pe un singur fir de date ("One Wire Interface"). Pinul numărul 3 este neconectat. Ultimul pin este cel care se conectează la masă.

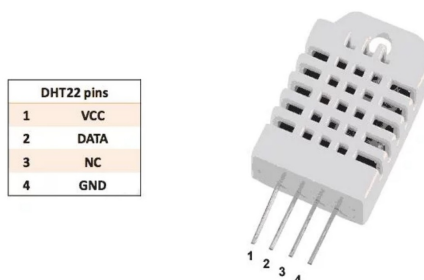


Fig. 7.1 Pinii senzorului de temperatură și umiditate DHT22.

Caracteristicile senzorului sunt:

- Tensiune de alimentare între 3-5V;
- Curent maxim: 2.5mA;
- Umiditatea măsurată: 0-100%, cu o acuratețe între 2-5%;
- Temperatura măsurată: de la -40°C până la 80°C , cu o acuratețe de $\pm 0.5^{\circ}\text{C}$.

Senzorul de temperatură utilizat în această aplicație poate fi înlocuit și cu DHT11 sau DHT21. Toți fac parte din aceeași clasă de senzori și utilizează același protocol de comunicație.

Citirea datelor de la senzorul DHT22 se va face utilizând-se librăria *DHT_sensor_library*. Pentru a putea fi utilizată aceasta trebuie inclusă în configurația proiectului. Accesul la funcțiile deja definite, de lucru cu librăria, se face prin includerea fișierului corespunzător librăriei dar și prin declararea clasei pentru lucrul cu senzorul. Următoarea secvență de cod exemplifică și celelalte setări de care mai este nevoie pentru a se utiliza librăria. În figura 7.2 este prezentată schema electrică de conectare a senzorului DHT la placa de dezvoltare Arduino Uno.

Algoritmul 7.1 Configurații necesare pentru utilizarea librăriei DHT.

```

1
2 #include "DHT.h"
3
4 //setare pin citire date de la senzor
5 #define DHT_PIN 2
6 //setare tip senzor
7 #define DHTTYPE DHT22
8
9 DHT dht(DHT_PIN, DHTTYPE);
10
11 void setup()
12 {
13     //Initializare comunicatie seriala
14     Serial.begin(9600);
15     //activarea senzorului DHT
16     dht.begin();
17     // Este necesara o intarziere pentru a se realiza
18     ↪ initializarea
19     delay(1000);
20 }
21 void loop()
22 {
23 }
```

Citirea temperaturii se realizează ușor, folosind clasa DHT din cadrul librăriei. Această

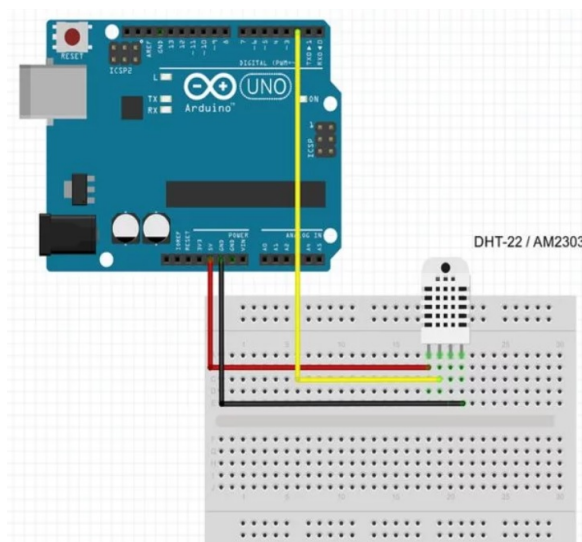


Fig. 7.2 Conectarea senzorului DHT22 la placa Arduino Uno.

clasă oferă funcții de acces direct la senzor și returnează valoarea temperaturii sau a umidității. Valoarea temperaturii poate fi returnată în grade Celsius sau în grade Fahrenheit. Un aspect important care trebuie luat în considerare este faptul că citirea de la senzor se poate face doar la un interval de maxim 0.5 secunde.

Următoarea secvență de cod demonstrează citirea de la senzor a valorii temperaturii/umidității și afișarea valorilor în consolă (pe linia de comunicații serială). Pe lângă citirea informației de la senzor mai este introdusă o linie de cod care verifică validitatea datelor citite de la senzor. În caz de eroare, aceasta este semnalată în consolă sub formă de mesaj.

Algoritmul 7.2 Configurații necesare pentru utilizarea librăriei DHT.

```

1 void loop() {
2     // Citirea temperaturii si a umiditatii mediului dureaza
3     // Citire umiditate
4     float h = dht.readHumidity();
5     //Citire temperatura in grade Celsius
6     float t = dht.readTemperature();
7     // Citire temperatura in grade Fahrenheit (isFahrenheit = true
8     // Citire temperatura in grade Fahrenheit (isFahrenheit = true
9     float f = dht.readTemperature(true);
10
11     // Verificam daca citirile au fost corecte
12     if (isnan(h) || isnan(t) || isnan(f))
13     {
14         Serial.println("Eroare la citirea de la senzorul DHT!");
15         return;
16     }

```

```

17     Serial.print("Humidity: ");
18     Serial.print(h);
19     Serial.print(" %\t");
20     Serial.print("Temperature: ");
21     Serial.print(t);
22     Serial.print(" *C ");
23     Serial.print(f);
24     Serial.print(" *F\t");
25
26     // Este necesar un delay intre citiri
27     delay(1000);
28 }

```

7.1.2 Utilizarea senzorului de ploaie

Senzorul de detectare a ploii este un mecanism foarte ușor de folosit și care permite detectarea ploii. El funcționează ca un switch, când picăturile de ploaie cad pe placa de citire a datelor. De asemenea, senzorul poate măsura și intensitatea căderii ploii. Sistemul senzorial este format din două componente: placa sensibilă la căderea ploii și o placă de control (separată de placa de citire).

Modulul de control este bazat pe circuitul LM393. Pe măsură ce picăturile de apă cad pe placa de colectare, ele crează trasee paralele care cresc rezistența circuitului. Aceste variații sunt măsurate de unitatea de control.

Senzorul, din punct de vedere funcțional, este un dipol rezistiv, adică va arăta o rezistență mai mică atunci când este umiditate pe el și returnează o rezistență mai mare când este uscat. Când picături de apă sunt prezente, ele reduc rezistența (apa este un bun conducător electric) prin conectarea liniilor de nichel de pe placa senzorială.

Unitatea de control este prezentată în figura 7.3. Semnificația elementelor componente este următoarea:

- Pinii A: pinii de conectare la placa senzorială;
- Potentiometrul B: permite ajustarea sensibilității citirii datelor de la placa senzorială;
- Pinul C: Pinul de ieșire analogică (A0). Valoarea pe pin variază între 0 și 5V. O tensiune mai mică reprezintă o umiditate mai mare;
- Pinul D: Pinul de ieșire analogică (D0) devine zero atunci când umiditatea depășește un prag presetat;
- Pinul E: Pin care este legat la masă (GND);
- Pinul F: Pin care se leagă la o sursă de alimentare (+5V).

Conectarea senzorului la placa de dezvoltare Arduino Uno se realizează conform cu schema din figura 7.4. Citirea de la senzor se face pe intrarea analogică A0. Ceilalți doi pini utilizați sunt alimentarea și semnalul de masă.

Următoarea secvență de cod prezintă modul de citire și interpretare a datelor de la senzorul

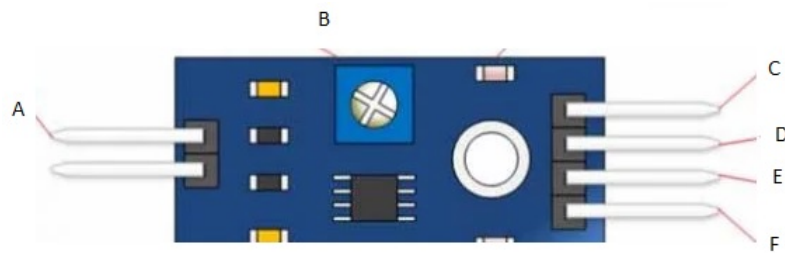


Fig. 7.3 Unitatea de control a senzorului de ploaie.

de ploaie. Informația primită este afișată în consolă, utilizându-se comunicația serială.

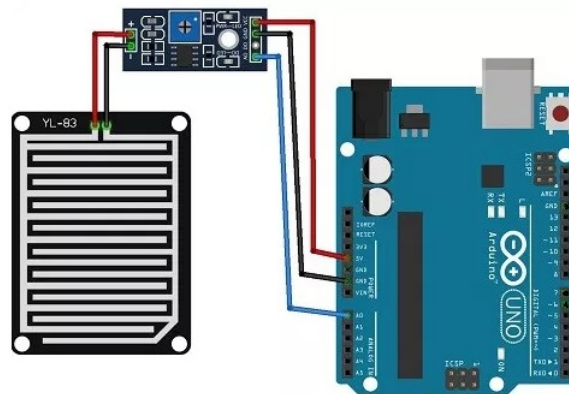


Fig. 7.4 Schema electrică pentru conectarea senzorului de ploaie la Arduino Uno.

Algoritmul 7.3 Citirea și afișarea datelor de la senzorul de ploaie.

```

1  const int sensorMinVal = 0;
2  const int sensorMaxVal = 1024;
3
4  void setup()
5  {
6      Serial.begin(9600);
7  }
8
9  void loop()
10 {
11     //Citire de la senzor, pe pinul A0
12     int citireSenzor = analogRead(A0);
13     int range = map(citireSenzor, sensorMinVal, sensorMaxVal, 0,
14                     ↪ 3);
15
16     switch (range)
17     {
18         case 0:
19             Serial.println("Ploaie puternica");

```

```
19         break;
20     case 1:
21         Serial.println("Ploaie usoara");
22         break;
23     case 2:
24         Serial.println("Fara ploaie");
25         break;
26     default:
27         Serial.println("Stare necunoscuta");
28         break;
29     }
30     delay(1000);
31
32 }
```

În exemplul anterior a fost făcută o mapare (categorisire) a datelor citite de la senzorul de ploaie, astfel încât să putem interpreta valorile citite. Maparea este una statică, fără a se ține cont de experimente anterioare.

7.1.3 Aplicație propusă

Pornind de la cele două module prezentate anterior, să se creeze o aplicație care să citească datele de la cei doi senzori (temperatura/umiditate și ploaie) și să afișeze pe un LCD 16x2 datele colectate de la senzori.

7.2 Controlul unui proces utilizându-se un regulator PID

Într-un sistem de reglare, rolul unui regulator este unul de decizie. El primește la intrare un semnal de referință și un semnal măsurat. După procesarea celor două mărimi, regulatorul elaborează un semnal de comandă pentru elementul de execuție (prin intermediul căruia se intervine direct asupra procesului reglat).

În practică, cele mai utilizate regulatoare sunt cele de tip Proporțional (P), Proporțional-Integrativ (PI), Proporțional-Derivativ (PD) și Proporțional-Integrativ-Derivativ (PID). Pentru exemplul propus, o să utilizăm un regulator de tip PID. Rolul fiecărei componente, din cadrul regulatorului, în control este următorul:

- componenta proporțională: permite o ajustare a mărimii de ieșire, proporțională cu valoarea erorii. Doar această componentă folosită ca și regulator asigură foarte rar eroare staționară nulă;
- componenta integrativă: pentru a elimina eroarea staționară, este introdusă componenta integrativă. Un astfel de regulator produce ajustări care au la bază eroarea acumulată pe parcursul timpului de evoluție a procesului;
- componenta derivativă: această componentă este responsabilă cu rata de variație (schimbare) a erorii. Această componentă este responsabilă cu predicția erorii. În

principiu, cu cât răspunsul sistemului se apropie mai rapid de referință, cu atât componenta derivativă va încetini evoluția răspunsului sistemului, astfel încât să nu se depășească mărimea de referință. Se folosește de obicei la procese lente.

Schema unui sistem în buclă închisă, care utilizează un regulator PID pentru controlul unui proces este prezentată în figura 7.5.

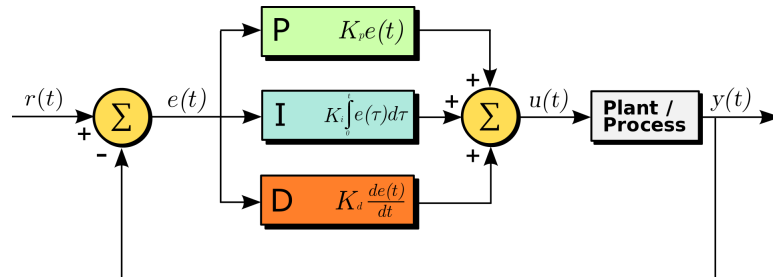


Fig. 7.5 Controlul unui proces utilizându-se un regulator PID.

În schema de control cu regulator PID sunt implicate următoarele mărimi:

- semnalul $r(t)$ reprezintă mărimea de intrare de referință;
- mărimea $y(t)$ reprezintă semnalul de ieșire;
- mărimea controlată este dată de $u(t)$;
- mărimea de eroare este $e(t)$.

Pentru a se obține performanțele dorite, parametrii unui regulator trebuie acordați. Acest proces presupune determinarea constantelor $\mathbf{K}$ din figura 7.5. De asemenea, calculele constantelor PID trebuie să se desfășoare la intervale cunoscute și exacte de timp.

Pentru implementarea unui regulator PID în cod Arduino este nevoie să se cunoască cinci parametri: constantele de tip proporțional, integrativ și derivativ, mărimea de referință și mărimea de intrare. Calculele pentru PID trebuie să se desfășoare la intervale cunoscute de timp. Pentru aceasta se recomandă utilizarea de întreruperi de timer. Pentru aplicația propusă o să utilizăm valoarea de 100ms pentru apariția unei întreruperi.

Pentru a se determina eroarea se folosește expresia:

```
1   err = marimeReferinta - marimeIntrare;
```

Componenta integrativă face referire la eroarea acumulată, astfel avem:

```
1   errAcumulata += err;
```

Componenta derivativă a erorii reprezintă rata de variație a erorii:

```
1   errRata = err - errAnterioara;
```

Avându-se în vedere mărimile anterior amintite se poate determina valoarea mărimii de ieșire, astfel:

```
1   iesire = Kp * err + Ki * errAcumulata + Kd * errRata;
```

Valorile constantelor K_p , K_i , K_d sunt predefinite. După determinarea mărimii de ieșire se calculează o valoare temporară a erorii, care o să fie utilizată în iterația următoare:

```
1    errAnterioara = err;
```

Aplicația propusă presupune controlul unui motor electric de curent continuu, astfel încât acesta să ajungă la o poziție predefinită. Această poziție se obține de la un encoder rezistiv. Valoarea acestuia poate fi determinată cu funcția de citire a unei mărimi analogice. Controlul motorului se face cu ajutorul unui semnal PWM, generat cu ajutorul unui timer. Constantele necesare în program au fost alese empiric.

Algoritmul 7.4 Implementare regulator PID pentru controlul unui motor de curent continuu.

```
1
2  long encoder_ticks = 0;
3  const float wheelPerimeter = 0.31416f;
4  const long ticksPerRevolution = 2240;
5  float speed = 0;
6  float distance = 0;
7  const float invTe = 10.f; // 1/Te
8
9  struct PIDstruct
10 {
11     double Kp;
12     double Ki;
13     double Kd;
14
15     double referinta;
16
17     double errAcumulata;
18     double errAnterioara;
19 } pidMotor;
20
21 // se poate si cu referinte, in locul pointerilor
22 void initializarePID(PIDstruct* pid, double Kp, double Ki, double
    ↪ Kd)
23 {
24     // setare constante regulator
25     pid->Kp = Kp;
26     pid->Ki = Ki;
27     pid->Kd = Kd;
28
29     pid->referinta = 0;
30
31     // setare valori initiale pentru erori
```

```
32     pid->errAcumulata = 0;
33     pid->errAnterioara = 0;
34 }
35
36 void setareReferintaPID(PIDstruct* pid, double ref)
37 {
38     pid->referinta = ref;
39 }
40
41 double calculareIesirePID(PIDstruct* pid, double marimeIesire)
42 {
43     // calculare eroare curenta
44     const double err = pid->referinta - marimeIesire;
45     // aproximare derivata eroare
46     const double errRata = (err - pid->errAnterioara);
47     // aproximare integrare eroare
48     pid->errAcumulata += err; //
49
50     // aplicarea formulei de calcul pentru PID
51     return (pid->Kp * err) + (pid->Ki * pid->errAcumulata) + (pid
        ↪ ->Kd * errRata);
52 }
53
54 void configurare_timer()
55 {
56     // Configurare Timer 1 pentru intrerupere la 10 Hz:
57     // oprire intreruperi
58     cli();
59
60     // initializare registru de control TCCR1A
61     TCCR1A = 0;
62     // initializare registru de control TCCR1B
63     TCCR1B = 0;
64     // initializare contor la 0
65     TCNT1 = 0;
66
67     // setare OCR pentru intrerupere la 10 Hz
68     // frecventa = 16000000 / (64 * 10) - 1 (trebuie sa fie <65536)
69     OCR1A = 24999;
70
71     // porneste CTC mode
```

```

72  TCCR1B |= (1 << WGM12);
73  // Setare CS12, CS11 si CS10 bits pentru presacalar pe 64 biti
74  TCCR1B |= (0 << CS12) | (1 << CS11) | (1 << CS10);
75  // activare intrerupere timer compare
76  TIMSK1 |= (1 << OCIE1A);
77
78  // activare intreruperi globale
79  sei();
80 }
81
82 void configurare_intrerupere()
83 {
84  // dezactivare intreruperi globale
85  SREG &= ~(1 << SREG_I);
86
87  // configurare intreruperi externe
88  // Aparitie intrerupere pe orice front al semnalului de intrare
89  EICRA = (0 << ISC11) | (0 << ISC10) | (0 << ISC01) | (1 << ISC00
    ↪ );
90  // Activare masca intrerupere externa
91  EIMSK = (0 << INT1) | (1 << INT0);
92  // Intializare biti marcare aparitie intrerupere
93  EIFR = (0 << INTF1) | (0 << INTF0);
94  // Permite activarea intreruperilor externe
95  PCICR = (0 << PCIE2) | (0 << PCIE1) | (0 << PCIE0);
96
97  // activare intreruperi globale
98  SREG |= 1 << SREG_I;
99 }
100
101 void setup()
102 {
103  // Configurare timer 1 pentru generare intrerupere la 100ms
104  configurare_timer();
105
106  // Configurare intrerupere externa pe int0 si int1
107  configurare_intrerupere();
108
109  initializarePID(&pidMotor, 2, 5, 1);
110  setareReferintaPID(&pidMotor, 1.5); // ex: 1.5m/s
111 }

```

```
112
113 void loop()
114 {
115 }
116
117 ISR(TIMER1_OVF_vect)
118 {
119     //dezactivare intreruperi globale
120     SREG &= ~(1 << SREG_I) ;
121
122     //calculam distanta avand in vedere constantele legate de
123     ↪ encoderul de pe motor
124     const float dist = encoder_ticks * wheelPerimeter /
125     ↪ ticksPerRevolution;
126     //determinam viteza motorului
127     speed = dist * invTe; //  $V = d / T$ 
128     //Determinam marimea de comanda, folosind regulatorul PID
129     const float pwm = calculateIesirePID(&pidMotor, speed);
130     //Setam semnalul de iesire pentru controlul motorului
131     analogWrite(3, pwm);
132     //reinitializam numarul de fante citite de la encoder
133     encoder_ticks = 0;
134
135     //activare intreruperi globale
136     SREG |= (1 << SREG_I);
137 }
138
139 ISR(INT0_vect)
140 {
141     // dezactivare intreruperi globale
142     SREG &= ~(1 << SREG_I);
143
144     encoder_ticks++;
145
146     // activare intreruperi globale
147     SREG |= 1 << SREG_I;
148 }
```

Aplicația propusă introduce o modalitate prin care se poate controla un motor electric de curent continuu, cu ajutorul PWM. Controlul se poate realiza avându-se în vedere encoderul rezistiv cu care este motorul echipat. Prin configurarea sistemului de întreruperi

este asigurată o frecvență constantă de apariție a întreruperii, astfel încât calculele pentru determinarea poziției curente să se realizeze cât mai bine, iar semnalul de comandă pentru mărirea de ieșire să fie cât mai corect.

7.2.1 Aplicație propusă

Pornind de la exemplul anterior, să se realizeze controlul unei platforme robotice mobile, înzestrată cu două roți de tracțiune la care sunt conectate două motoare electrice de curent continuu. Motoarele sunt prevăzute cu encodere de timp optic. Astfel, pentru determinarea erorii trebuie să se țină cont de numărul de fante citite de la motoare.

7.3 Controlul poziției unui motor DC utilizandu-se control la stare

Controlul în spațiul stărilor permite ajustarea mărimii de intrare într-un sistem dinamic, astfel încât să se mențină mărirea de ieșire la o valoare impusă. La fel ca și un regulator PID, un regulator în spațiul stărilor măsoară mărimile de ieșire ale sistemului și calculează în concordanță mărimile de intrare. Diferența majoră față de un sistem de reglare cu regulator de tip PID este dată de faptul că în cazul controlului la stare este luat în considerare și modelul sistemului care este controlat. Definirea unui model pentru un sistem dinamic poate fi în unele situații o problemă mai complicată, fapt care face ca utilizarea acestui tip de regulator să fie una mai greoaie decât un regulator clasic de tip PID.

În general un regulator cu control la stare este indicat să se utilizeze la controlul sistemelor cu o dinamică mai complexă, sau la sisteme care au mai multe intrări sau ieșiri.

Controlul la stare presupune definirea unui sistem general de ecuații diferențiale, exprimat într-o formă matriceală standard. O astfel de reprezentare va purta numele de reprezentare în spațiul stărilor și este caracterizată de următoarele ecuații:

$$\dot{\mathbf{x}} = \mathbf{A} \cdot \mathbf{x}(t) + \mathbf{B} \cdot \mathbf{u}(t) \quad (7.1)$$

$$\mathbf{y}(t) = \mathbf{C} \cdot \mathbf{x}(t) + \mathbf{D} \cdot \mathbf{u}(t) \quad (7.2)$$

unde:

- $\mathbf{x} \in \mathbb{R}^X$ reprezintă vectorul mărimilor de stare ale sistemului;
- $\mathbf{u} \in \mathbb{R}^U$ reprezintă vectorul mărimilor de intrare;
- $\mathbf{y} \in \mathbb{R}^Y$ reprezintă vectorul mărimilor de ieșire;
- $\mathbf{A} \in \mathbb{R}^{X \times X}$ reprezintă matricea tranzițiilor stărilor, care descrie modul cum o să evolueze stările în situația în care nu sunt aplicate semnale de intrare de control;
- $\mathbf{B} \in \mathbb{R}^{X \times U}$ reprezintă matricea de scalare a mărimilor de intrare, care descrie efectul mărimilor de control asupra stărilor;
- $\mathbf{C} \in \mathbb{R}^{Y \times X}$ reprezintă matricea de scalare a mărimilor de ieșire sau matricea observațiilor, și descrie cum stările sunt mapate pe măsurătorile de pe senzori;
- $\mathbf{D} \in \mathbb{R}^{Y \times U}$ reprezintă matricea de scalare a mărimilor de ieșire (matricea de tranziție), și conține informații dacă oricare dintre intrări influențează imediat ieșirile (este de

obicei de valoare zero).

Pornind de la cele 4 matrici anterioare este posibilă descrierea oricărui sistem liniar, care nu este dependent de timp.

Schema bloc generală pentru un sistem de control în spațiul stărilor este prezentată în figura 7.6. Schema propusă permite sistemului să urmărească o mărime de referință impusă.

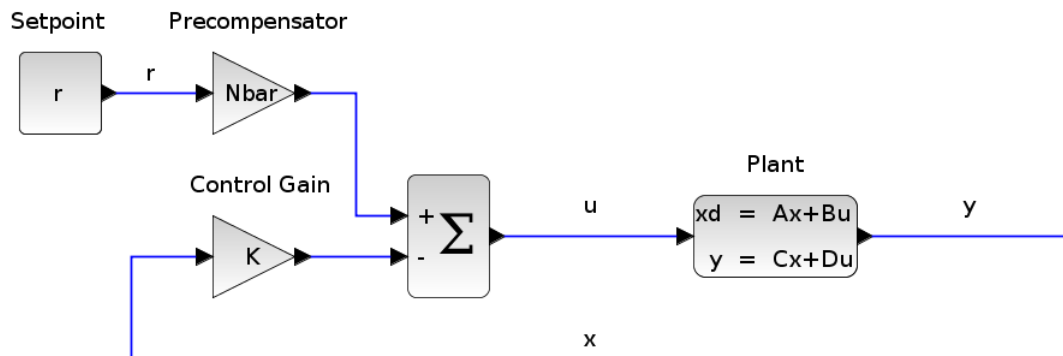


Fig. 7.6 Schema bloc pentru controlul la stare, cu urmărirea referinței.

Matricea $\mathbf{K}$ conține parametrii care trebuie acordați în funcție de sistemul considerat. În funcție de complexitatea sistemului, matricea $\mathbf{K}$ poate deveni destul de mare, iar acordarea ei va constitui un proces destul de dificil. Pentru a rezolva această problemă se poate utiliza un algoritm de tip LQR (*Linear-Quadratic Regulator*).

În schema bloc pentru controlul la stare matricea $\bar{N} \in \mathbb{R}^{U \times Y}$ influențează mărimea de control $\mathbf{u}$ prin introducerea unei compensații relative la semnalul de referință $\mathbf{r}$. Cu ajutorul acestei compensații este posibilă atingerea valorii de referință pentru sistem. Matricea se determină în totalitate din matricele sistemului, fără a fi nevoie de acordarea vreunui parametru.

Pentru aplicația propusă ca și exemplu se va utiliza un model cu următorii parametrii:

- Momentul de inerție al rotorului: $J = 3,2284e^{-6} [kg \cdot m^2]$;
- Constanta de frecare vâscoasă a motorului: $b = 3,50776e^{-6} [N \cdot m \cdot s]$;
- Constanta electrică a motorului: $K_b = 0,0274 [\frac{V \cdot sec}{rad}]$;
- Constanta de cuplu a motorului: $K_t = 0,0274 [\frac{N \cdot m}{A}]$;
- Constanta de rezistență electrică: $R = 4 [\Omega]$;
- Constanta de inductanță electrică: $L = 2,75e^{-6} [H]$;

Pornind de la legea a doua a lui Newton și legea lui Kirchhof se pot obține ecuațiile care descriu funcționarea motorului. Aceste ecuații sunt utilizate pentru a se obține modelul motorului exprimat în spațiul stărilor. Acest lucru se face prin alegerea ca mărimi de stare a poziției motorului, vitezei motorului și a curentului prin armătura. Tensiunea la capetele armăturii este considerată ca și mărime de intrare iar poziția rotorului este considerată ca

și mărime de ieșire. Ecuatiile în spațiul stărilor sunt:

$$\frac{d}{dt} \begin{bmatrix} \theta \\ \dot{\theta} \\ i \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & -\frac{b}{J} & \frac{K}{J} \\ 0 & -\frac{K}{L} & -\frac{R}{L} \end{bmatrix} \cdot \begin{bmatrix} \theta \\ \dot{\theta} \\ i \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \frac{1}{L} \end{bmatrix} V \quad (7.3)$$

$$y = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \theta \\ \dot{\theta} \\ i \end{bmatrix} \quad (7.4)$$

unde, θ reprezintă poziția unghiulară, $\dot{\theta}$ este viteza unghiulară iar i este curentul prin motor. Pentru controlul în spațiul stărilor o să utilizăm librăria StateSpaceControl. Aceasta pune la dispoziție o clasă template care are implementat un regulator în spațiul stărilor, utilizat la sisteme cu mai multe intrări și ieșiri. Aceasta poate fi descărcată gratuit de pe repo-ul: <https://github.com/tomstewart89/StateSpaceControl>.

Pentru a se putea crea regulatorul în spațiul stărilor, este nevoie la început de definirea unui model care să descrie dinamica sistemului ce urmează a fi reglat. Pentru aceasta este nevoie să se definească numărul de stări, de intrări și de ieșiri pe care sistemul le are. Toate acestea se realizează astfel:

```
1      Model<X,U,Y> model; //unde X este numarul de stari, U numarul
      ↪ de intrari iar Y este numarul de iesiri
```

Librăria de control propusă conține și modelul concret al unui motor electric de curent continuu. Modelul are la bază ecuațiile la stare amintite anterior.

Definirea celor patru matrici utilizate pentru descrierea sistemului ce urmează a fi controlat, se face folosind relații de forma:

```
1      model.A << 1, 2, 3,
2              4, 5, 6,
3              7, 8, 9;
```

Pornind de la definițiile anterioare se poate construi și modelul pentru regulator:

```
1      //se vor inlocui valorile pentru X, U si Y cu valorile
      ↪ corespunzatoare pentru sistemul considerat
2      StateSpaceController<X,U> controller(model);
```

Ajustarea parametrilor matricei de control K se face utilizându-se:

```
1      controller.k << 1, 2, 3, 4;
```

Având declarațiile anterioare, pentru definirea sistemului și a parametrilor, mai este nevoie de inițializarea efectivă a regulatorului și de setarea valorii de referință. Acest lucru se poate realiza utilizându-se următoarele linii de cod:

```
1      //Initializare regulator
```

```

2     controller.initialise();
3
4     //Setare marime de referinta
5     controller.r << 2.2;

```

Funcționarea propriu-zisă a regulatorului presupune ca la fiecare etapă să se facă o împropățare a valorilor regulatorului, avându-se în vedere observația curentă y (datele curente citite de la senzori). Regulatorul cu parametri actualizați va putea fi de asemenea accesat. Codul corespunzător este următorul:

```

1     controller.update(y, dt); //dt este un interval fix de timp
2
3     //accesarea marimii de intrare
4     controller.u;

```

Pentru controlul poziției unui motor de curent continuu se poate utiliza urmatorul cod. Acest cod are în vedere declarațiile anterioare și folosește librăria amintită. Codul rulează pe o placă de dezvoltare Arduino Uno.

Algoritmul 7.5 Controlul poziției unui motor de cc utilizându-se controlul la stare

```

1  #include <StateSpaceControl.h>
2
3  //Definim modelul la stare pentru un motor de cc
4  //parametrii sunt: J, b, k, R, L – conform modelului anterior
5  MotorPositionModel model(0.01, 0.1, 0.01, 1, 0.5);
6
7  //Definim modelul la stare al regulatorului. Modelul pentru
   ↪ pozitionarea motorului utilizeaza 3 stari si o intrare
8  StateSpaceController<3, 1> controller(model);
9
10 //Matricea de iesire
11 Matrix<3> y;
12 //constanta de timp
13 const float dt = 0.01;
14
15 void setup()
16 {
17     Serial.begin(115200);
18
19     //Acordarea matricei K. A fost realizata utilizand un tool
   ↪ disponibil in librerie: TuneThoseGains.ipynb
20     controller.K << 9.75, 0.97, 0.40;
21

```

```

22 //Initializarea regulatorului , astfel se poate determina N_bar
23 controller.initialise();
24
25 //Setarea referintei pentru pozitia la 2.2
26 controller.r << 2.2;
27 }
28
29 void loop()
30 {
31 //Y se actualizeaza folosind informatia de la senzor
32 //mecanismul utilizat pentru determinarea Y curent depinde de la
    ↪ aplicatie la aplicatie
33
34 //actualizarea regulatorului , avand in vedere vectorul
    ↪ actualizat al marimilor de iesire
35 controller.update(y, dt);
36
37 // Afisare rezultate
38 Serial << "unghi = " << y(0) << " viteza = " << y(1) << " curent
    ↪ = " << y(2) << '\n';
39
40 delay(dt * 1000);
41 }

```

În cazul aplicației este nevoie de citirea poziției, vitezei și a curentului prin circuit. Pentru primele două mărimi se poate utiliza un encoder optic. Cu ajutorul acestuia se pot contoriza într-o întrerupere numărul de fante care au trecut prin fața dispozitivului de citire și putem afla poziția. Pentru viteză trebuie să ținem cont de timpul necesar realizării unei rotații complete și de distanța parcursă la o rotație completă. Astfel putem determina viteza cu care se rotește axul motorului. Pentru măsurarea curentului este nevoie de utilizarea unui circuit adițional, de tip ACS712.

Circuitul ACS712 este utilizat pentru măsurarea cu exactitate a curentului alternativ sau continuu. Principiul de funcționare are la bază efectul Hall. Mărimea de ieșire a senzorului este una analogică, proporțională cu curentul din circuitul considerat.

Pinii senzorului sunt prezentați în figura 7.7 și au următoarea semnificație:

- Vcc - sursa de alimentare;
- Gnd - semnal de masă;
- Out - semnalul analogic de ieșire care este proporțional cu curentul din circuit;
- Current IN/OUT - pinii pentru conectarea în serie cu circuitul asupra căruia se dorește măsurarea curentului;

Interfațarea motorului cu placa de dezvoltare Arduino este prezentată în figura 7.8.

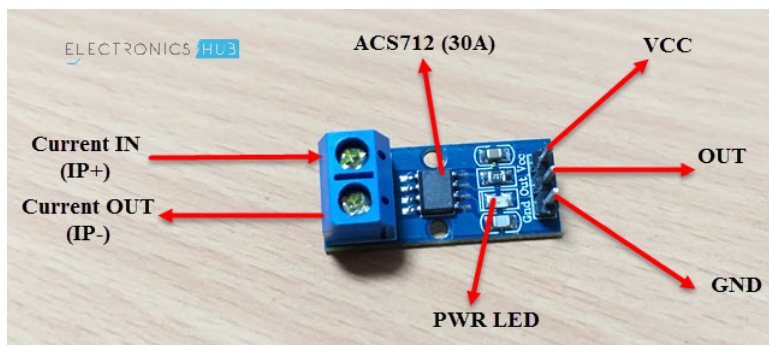


Fig. 7.7 Schema electrică de utilizare a senzorului ACS712.

Motorul este simbolizat de un modul "Load" în schema propusă. Rezultatul măsurătorii se poate afișa în terminal sau pe un LCD. Senzorul poate fi echipat cu 3 tipuri de sensibilitate, iar mărimea de ieșire este:

- pentru un curent de 5A cu o ieșire de 66mV/A;
- pentru un curent de 20A cu o ieșire de 100mV/A;
- pentru un curent de 30A cu o ieșire de 185mV/A.

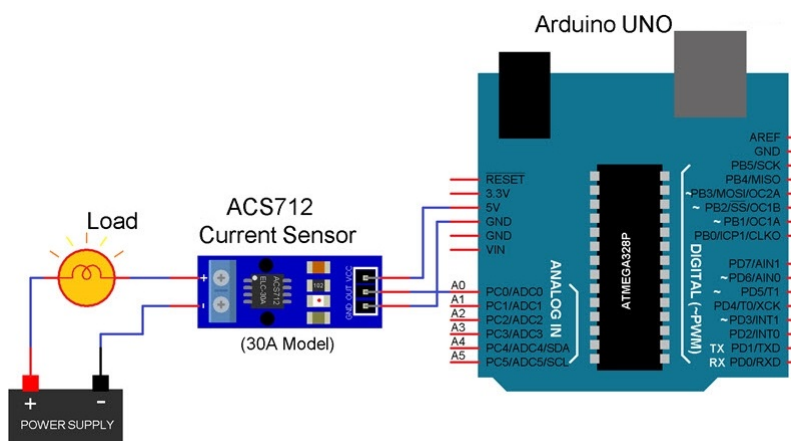


Fig. 7.8 Pinii senzorului ACS712.

Algoritmul 7.6 Determinarea curentului într-un circuit electric folosind senzorul ACS712.

```

1  const int currentPin = A0;
2  //sensitivitatea depinde de tipul senzorului. Pentru cazul de fata
   ↳ este un senzor de 5A
3  int sensitivity = 66;
4  int adcValue= 0;
5  int offsetVoltage = 2500;
6  double adcVoltage = 0;
7  double currentValue = 0;
8
9  void setup()
10 {
11     Serial.begin(9600);

```

```

12 }
13
14 void loop()
15 {
16     adcValue = analogRead(currentPin);
17     //determinare tensiune, pentru un convertor pe 10 biti, cu o
        ↳ tensiune de alimentare de 5000mV
18     adcVoltage = (adcValue / 1024.0) * 5000;
19     currentValue = ((adcVoltage - offsetVoltage) / sensitivity);
20
21     Serial.print("Raw Sensor Value = ");
22     Serial.print(adcValue);
23
24     Serial.print("\t Voltage(mV) = ");
25     Serial.print(adcVoltage,3);
26
27     Serial.print("\t Current = ");
28     Serial.println(currentValue,3);
29
30     delay(2500);
31 }

```

7.3.1 Aplicație propusă

Să se implementeze o aplicație care să permită controlul a două motoare de curent continuu montate pe o platforma mobilă. Ambele motoare trebuie controlate cu ajutorul unui regulator. Se va folosi controlul la stare pentru regulator. Pe platforma mobilă se va monta un senzor de culoare care este capabil să recunoască o dungă albă. În funcție de această bandă, platforma trebuie să se centreze.

7.4 Modul de comunicație wireless

Această aplicație își propune să prezinte o modalitate prin care se pot transmite date între diferite module de tip Arduino. Pentru transferul datelor se vor folosi module de tip emitor-receptor din categoria nRF24L01.

Modulul nRF24L01 este un modul care poate transmite sau recepționa date. Este un modul ieftin și ușor de integrat în aplicații cu plăci de dezvoltare de tip Arduino Uno. Dintre principalele caracteristici ale acestor module putem aminti:

- puterea consumată este de aproximativ 12mA pe parcursul transmisiei (este mai mică și decât puterea consumată de un LED);
- poate funcționa cu rate de transfer de la 250Kbps până la 2Mbps;
- poate emite până la 100m depărtare în spațiu liber și este folosită o antenă;

- poate emite sau recepționa în același timp;
- fiecare modul poate comunica cu până la 6 alte dispozitive;
- folosește banda de 2.4GHz pentru transferul de date;
- are 125 de canale diferite de comunicație.

În figura 7.9 este prezentată configurația pinilor modului. Modulul se conectează la

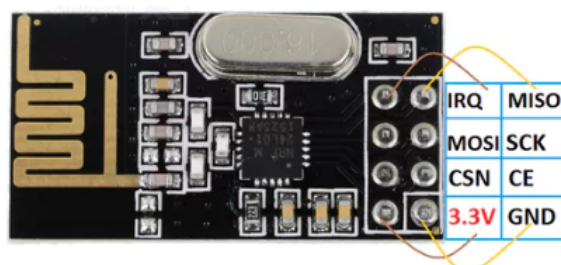


Fig. 7.9 Pinii senzoru nRF24L01.

placa Arduino cu ajutorul comunicației SPI. Configurația pinilor este următoarea:

- 3.3V este tensiunea de alimentare. Modulul poate fi conectat la tensiuni care pornesc de la 1.9V și ajung la 3.6V. Ceilalți pini pot fi conectați la tensiuni de 5V (conectați direct la placa Arduino);
- GND este semnalul de masă;
- MISO, MOSI și SCK sunt cei trei pini necesari realizării comunicației SPI. Ei se conectează la pinii corespunzători comunicației SPI de la placa de dezvoltare Arduino;
- CSN și CE sunt utilizați pentru setarea modului pe starea activ/inactiv. De asemenea pentru setarea funcționării ca emitor sau ca receptor;
- pinul IRQ nu trebuie conectat.

Principul de funcționare al aplicației presupune interfațarea a două module Arduino Uno, folosind modulul de comunicație descris anterior. Astfel, se vor trimite date de la un Arduino către al doilea modul Arduino. Atunci când se va apăsa un buton conectat la prima plăcuță, ledul de pe cea de a doua plăcuță se va aprinde. Schema electrică este prezentată în figura 7.10.

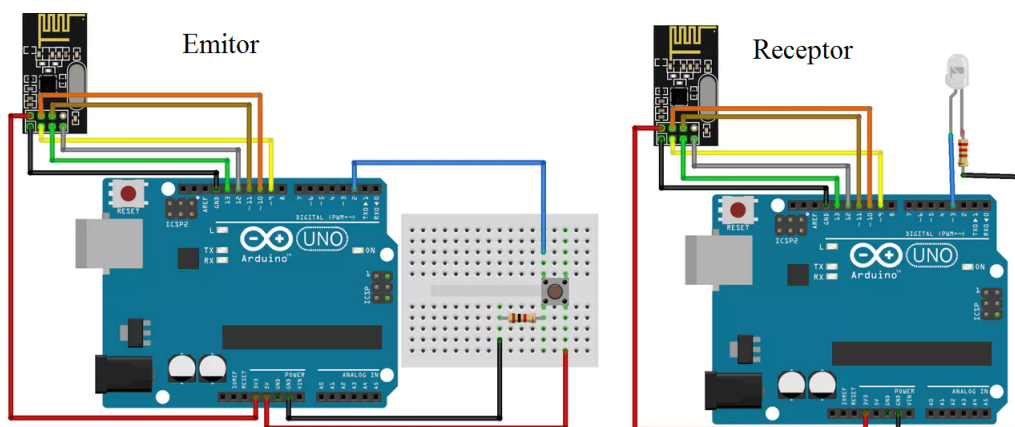


Fig. 7.10 Schemă electrică pentru comunicația wifi între două module Arduino Uno.

Pentru realizarea comunicației propuse se vor utiliza două librării pentru comunicația

wifi și pentru comunicația SPI. Codul corespunzător modulului emitor este următorul:

Algoritmul 7.7 Codul pentru modulul emitor.

```

1  #include <SPI.h>
2  #include <nRF24L01.h>
3  #include <RF24.h>
4
5  //initializare obiect de tip nRF, pe pinii 9 si 10 pentru
   ↪ semnalele CE si CSN
6  RF24 radio(9, 10); // CE, CSN
7
8  //Asignarea unei adrese unice pentru modulul catre care se trimit
   ↪ datele. Aceasta adresa trebuie sa fie identica si pe modulul
   ↪ de receptie
9  const byte address[6] = "00001";
10
11 bool stare_buton = 0;
12
13 void setup()
14 {
15     //Configurare modul comunicatie
16     //Pornirea comunicatiei wifi
17     radio.begin();
18     //Setarea adresei unde o sa fie trimise datele
19     radio.openWritingPipe(address);
20     //Valoarea poate fi setata la minim, aceasta depinzand de
   ↪ distanta dintre emitor si receptor
21     radio.setPALevel(RF24_PA_MIN);
22     //Configurare modul ca si emitor
23     radio.stopListening();
24
25     // configurare intreruperi externe pe int0
26     EICRA = (0 << ISC11) | (0 << ISC10) | (1 << ISC01) | (1 <<
   ↪ ISC00);
27     EIMSK = (0 << INT1) | (1 << INT0);
28     EIFR = (0 << INTF1) | (0 << INTF0);
29     PCICR = (0 << PCIE2) | (0 << PCIE1) | (0 << PCIE0);
30 }
31
32 void loop()
33 {

```

```
34     if(stare_buton)
35     {
36         const char text[] = "Starea butonului este HIGH";
37         //Transmiterea mesajului la receptor
38         radio.write(&text, sizeof(text));
39     }
40     else
41     {
42         const char text[] = "Starea butonului este LOW";
43         radio.write(&text, sizeof(text));
44     }
45
46     //Transmiterea mesajului la receptor
47     radio.write(&stare_buton, sizeof(stare_buton));
48
49     delay(1000);
50 }
51
52 ISR(INT0_vect)
53 {
54     // dezactivare intreruperi globale
55     SREG &= ~(1 << SREG_I);
56
57     //validare buton apasat
58     if (!stare_buton)
59         stare_buton = true;
60     else
61         stare_buton = false;
62
63     // activare intreruperi globale
64     SREG |= 1 << SREG_I;
65 }
```

Algoritmul 7.8 Codul pentru modulul receptor.

```
1 #include <SPI.h>
2 #include <nRF24L01.h>
3 #include <RF24.h>
4
5 //initializare obiect de tip nRF, pe pinii 9 si 10 pentru
  ↪ semnalele CE si CSN
6 RF24 radio(9, 10); // CE, CSN
```

```
7
8 //Asignarea unei adrese unice pentru modulul catre care se trimit
   ↳ datele. Aceasta adresa trebuie sa fie identica si pe modulul
   ↳ de receptie
9 const byte address[6] = "00001";
10 bool stare_buton = 0;
11
12 void setup()
13 {
14     Serial.begin(9600);
15     radio.begin();
16     //Setarea adresei unde o sa fie trimise datele
17     radio.openReadingPipe(0, address);
18     //Valoarea poate fi setata la minim, aceasta depinzand de
   ↳ distanta dintre emitor si receptor
19     radio.setPALevel(RF24_PA_MIN);
20     //Configurare modul ca si receptor
21     radio.startListening();
22
23     //Configurare port de iesire
24     DDRD |= (1<<PD3);
25 }
26 void loop()
27 {
28     //verifica daca sunt date disponibile
29     if (radio.available())
30     {
31         //salvam datele primite
32         char text[32] = "";
33         //citim datele
34         radio.read(&text, sizeof(text));
35         radio.read(&button_state, sizeof(button_state));
36
37         if(stare_buton == HIGH)
38         {
39             PORTD = 1 << PORTD3;
40             Serial.println(text);
41         }
42         else
43         {
44             PORTD = 0 << PORTD3;
```

```

45         Serial.println(text);}
46     }
47     delay(5);
48 }

```

Aplicația anterioară poate fi extinsă la transferul bidirecțional între ambele module. Astfel, fiecare dintre cele două module pot avea rolul de emitor sau receptor, în funcție de necesitate. Schema electrică a unui astfel de sistem este în figura 7.11.

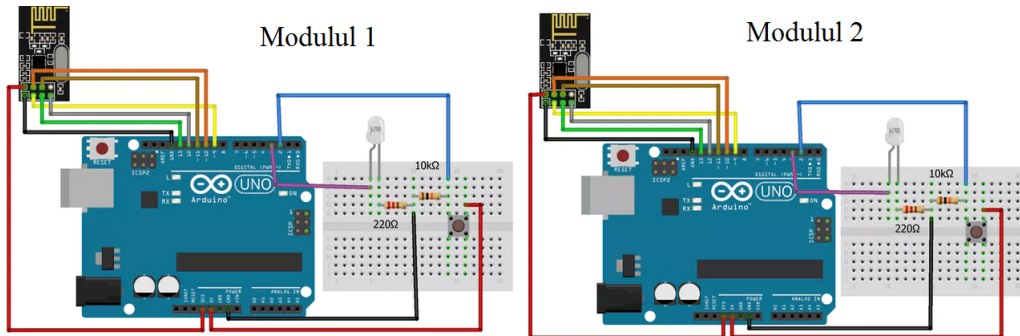


Fig. 7.11 Schemă electrică pentru comunicația wifi între două module Arduino Uno, cu transfer reciproc de date.

Funcțional, sistemul o să transmită o comandă de aprindere led, de la modulul 1 către modulul 2, apoi al doilea modul o să trimită o comandă similară către primul modul. Codul corespunzător modulului 1 este următorul:

Algoritmul 7.9 Codul pentru modulul 1.

```

1  #include <SPI.h>
2  #include <nRF24L01.h>
3  #include <RF24.h>
4
5  //initializare obiect de tip nRF, pe pinii 9 si 10 pentru
   ↳ semnalele CE si CSN
6  RF24 radio(9, 10);
7
8  //Setarea adreselor pentru functia de emitor si receptor
9  const byte addresses[][6] = {"00001", "00002"};
10
11  boolean stare_buton_1 = 0;
12  boolean stare_buton_2 = 0;
13
14  void setup() {
15     //Configurare port pentru LED si buton
16     DDRD |= (1<<PD3);
17
18     //configurare intrerupere pe int1

```

```
19      // configurare intreruperi externe pe int0
20      EICRA = (0 << ISC11) | (0 << ISC10) | (1 << ISC01) | (1 <<
      ↪ ISC00);
21      EIMSK = (0 << INT1) | (1 << INT0);
22      EIFR = (0 << INTF1) | (0 << INTF0);
23      PCICR = (0 << PCIE2) | (0 << PCIE1) | (0 << PCIE0);
24
25      radio.begin();
26      //Setarea adresei la care se vor trimite date
27      radio.openWritingPipe(addresses[1]);
28      //setarea adresei de la care se vor trimite date
29      radio.openReadingPipe(1, addresses[0]);
30
31      radio.setPALevel(RF24.PA_MIN);
32  }
33  void loop()
34  {
35      delay(5);
36      //setare modul ca emitor
37      radio.stopListening();
38      //trimitem date
39      radio.write(&stare_buton_1, sizeof(stare_buton_1));
40      delay(5);
41
42      //setare modul ca receptie
43      radio.startListening();
44      //verifica daca sunt date disponibile
45      while(!radio.available());
46      //daca sunt date disponibile, le citeste
47      radio.read(&button_state1, sizeof(button_state1));
48
49      if (button_state2 == HIGH)
50      {
51          PORTD = 1 << PORTD3;
52      }
53      else
54      {
55          PORTD = 0 << PORTD3;
56      }
57  }
58
```

```

59 ISR(INT0_vect)
60 {
61     // dezactivare intreruperi globale
62     SREG &= ~(1 << SREG_I);
63
64     //validare buton apasat
65     if (!stare_buton_1)
66         stare_buton_1 = true;
67     else
68         stare_buton_1 = false;
69
70     // activare intreruperi globale
71     SREG |= 1 << SREG_I;
72 }

```

Codul corespunzător modului 2 este următorul:

Algoritmul 7.10 Codul pentru modulul 2.

```

1  #include <SPI.h>
2  #include <nRF24L01.h>
3  #include <RF24.h>
4
5  RF24 radio(9, 10);
6
7  //Setarea adreselor pentru functia de emitor si receptor
8  const byte addresses[][6] = {"00001", "00002"};
9
10 boolean stare_buton_1 = 0;
11 boolean stare_buton_2 = 0;
12
13 void setup() {
14     //Configurare port pentru LED si buton
15     DDRD |= (1<<PD3);
16
17     //configurare intrerupere pe int1
18     // configurare intreruperi externe pe int0
19     EICRA = (0 << ISC11) | (0 << ISC10) | (1 << ISC01) | (1 <<
        ↪ ISC00);
20     EIMSK = (0 << INT1) | (1 << INT0);
21     EIFR = (0 << INTF1) | (0 << INTF0);
22     PCICR = (0 << PCIE2) | (0 << PCIE1) | (0 << PCIE0);
23

```

```
24  radio.begin();
25  //Setarea adresei la care se vor trimite date
26  radio.openWritingPipe(addresses[1]);
27  //setarea adresei de la care se vor trimite date
28  radio.openReadingPipe(1, addresses[0]);
29  radio.setPALevel(RF24_PA_MIN);
30  }
31  void loop()
32  {
33      delay(5);
34      //setare modul ca emitor
35      radio.stopListening();
36      //trimitere de date
37      radio.write(&button_state_1, sizeof(button_state_1));
38      delay(5);
39
40      //setare modul ca receptor
41      radio.startListening();
42      //verifica daca sunt date de intrare disponibile
43      while(!radio.available());
44      //citire date
45      radio.read(&button_state1, sizeof(button_state1));
46
47      if (button_state2 == HIGH)
48      {
49          PORTD = 1 << PORTD3;
50      }
51      else
52      {
53          PORTD = 0 << PORTD3;
54      }
55  }
```

7.4.1 Aplicație propusă

Să se implementeze o aplicație care controlează la distanță un motor electric de curent continuu. Transferul de informații între cele două module se realizează folosind modulul nRF24L01. Controlul se implementează avându-se în vedere aplicația cu reglator PID, prezentată anterior. Afișarea informațiilor primite de la modulele de comunicație se pot afișa în terminal sau se poate atașa un afișaj cu LCD.

7.5 Modul de măsurare a distanței

Pentru a se putea realiza măsurarea de distanțe, se poate utiliza un senzor cu ultrasunete de tipul HC-SR04. Acesta poate returna distanțe de la 2cm până la 3-4m. Această familie de senzori este una care este cu prețuri reduse și se potrivește foarte bine în aplicații care necesită consum redus de energie.

Senzorul cu ultrasunete HC-SR04 conține două module în interior. Primul modul este folosit pentru transmiterea de date, prin convertirea unui semnal electric într-o succesiune de pulsuri la o frecvență de 40KHz. Al doilea modul este destinat recepției pulsurilor trimise de primul modul. Dacă acesta le recepționează, o să emită un puls la ieșirea sa, a cărui lățime este dependentă de distanța pe care pulsul emitent a avut-o. Dintre cele mai importante specificații ale sale amintim:

- tensiune de alimentare de 5V;
- curentul de operare de 15mA;
- frecvența de operare de 40KHz;
- distanța minimă și maximă: 2cm - 4m;
- unghi de măsurare de 15 grade.

În figura 7.12 este prezentată configurația pinilor modulului. Configurația pinilor senzorului este următoarea:

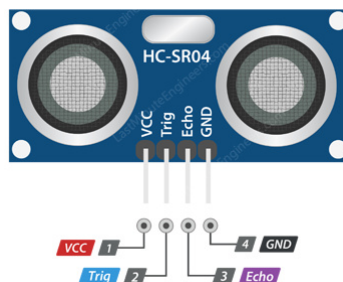


Fig. 7.12 Pinii senzorului HC-SR04.

- Vcc este tensiunea de alimentare (5V);
- GND este semnalul de masă;
- Trig este utilizat pentru declanșarea pulsurilor ultrasonice;
- Echo va produce pe pin un puls cu lățime variabilă, atunci când semnalul reflectat este recepționat.

Pornirea senzorului este dată de aplicarea unui puls de $10\mu s$ pe pinul de Trigger așa cum se poate vedea în figura 7.13. Ca și răspuns la acest impuls aplicat, senzorul transmite o succesiune de pulsuri la o frecvență de 40KHz. Această combinație de 8 pulsuri fac dispozitivul emitent unic, astfel încât să se poată diferenția de alte dispozitive cu ultrasunete. Dacă cele 8 pulsuri sunt reflectate din scenă, receptorul le va înregistra. Pinul de Echo devine activ și începe să formeze semnalul de răspuns, care poate fi interpretat de modulul Arduino. Dacă semnalul emis nu se reflectă, pinul de echo o să genereze un puls de 38ms.

Aplicația exemplu presupune construirea unui modul care să afișeze distanța față de un obiect aflat în fața senzorului. Afișarea se realizează pe un afișaj LCD 16x2. Schema electrică

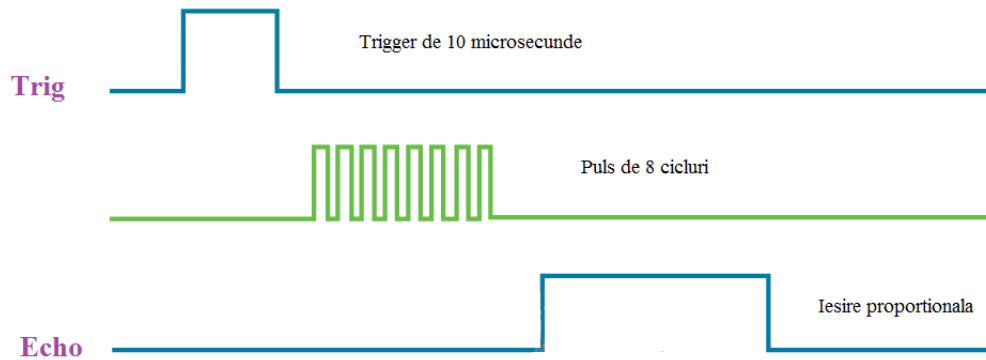


Fig. 7.13 Principiu de funcționare al senzorului cu ultrasunete.

a aplicației propuse este în figura 7.14.

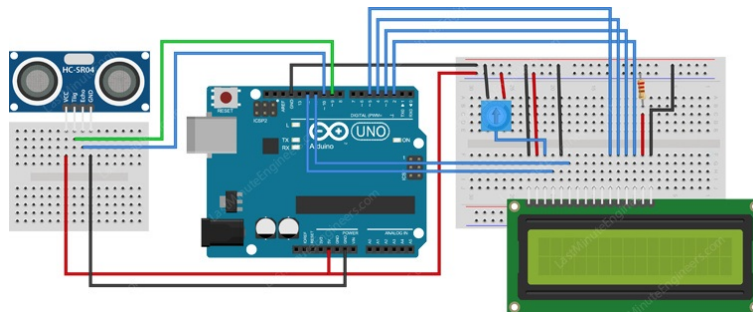


Fig. 7.14 Schema electrică pentru modulul de măsurare a distanței.

Pentru a putea utiliza modulul de afișaj o să utilizăm pachetele LCDBarGraph și LiquidCrystal. Codul corespunzător aplicației propuse este următorul.

Algoritmul 7.11 Codul pentru modulul de măsurare a distanței.

```

1 //include biblioteca LiquidCrystal
2 #include <LiquidCrystal.h>
3
4 //include biblioteca LcdBarGraph
5 #include <LcdBarGraph.h>
6
7 //Definim distanta maxima pe care dorim sa o atingem
8 #define distanta_maxima 200
9
10 // Creem un obiect de tip LCD conectat la pinii/semnalele: (rs,
    ↪ enable, d4, d5, d6, d7)
11 LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
12
13 //creem un obiect LCD Bargraph
14 LcdBarGraph lbg(&lcd, 16, 0, 1);
15

```

```
16 const int trigPin = 9;
17 const int echoPin = 10;
18 long durata;
19 int distanta;
20
21 void setup()
22 {
23     //initializam interfata afisajului LCD 16x2
24     lcd.begin(16,2);
25     //Configuram pinii de intrare sau iesire
26     pinMode(trigPin, OUTPUT);
27     pinMode(echoPin, INPUT);
28 }
29
30 void loop()
31 {
32     // Scriem un puls pe pinul Trigger al modului HC-SR04
33     digitalWrite(trigPin, LOW);
34     delayMicroseconds(2);
35     digitalWrite(trigPin, HIGH);
36     delayMicroseconds(10);
37     digitalWrite(trigPin, LOW);
38
39     //Masuram raspunsul de pe pinul Echo
40     duration = pulseIn(echoPin, HIGH);
41
42     // Determinam distanta din durata semnalului
43     // Utilizam 343 m/s pentru viteza sunetului
44     distanta = duration*0.034/2;
45
46     // Afisam rezultatele pe LCD
47     lcd.setCursor(0,0);
48     lcd.print("Distanța: ");
49     lcd.print(distanta);
50     lcd.print(" cm");
51
52     // Desenam un bargraph pe a doua linie a LCD-ului
53     lcd.setCursor(0,1);
54     lbg.drawValue(distanta, distanta_maxima);
55     delay(500);
56 }
```

7.5.1 Aplicație propusă

Să se implementeze un modul de măsurare bidirecțional la distanță. Pentru aceasta se vor utiliza două module de senzori cu ultrasunete HC-SR04 care vor transmite informația către o placă de dezvoltare Arduino Uno. Informația va fi transferată pe baza unui modul wifi nRF24L01 către un al doilea modul Arduino. Acesta va afișa informația pe un dispozitiv LCD dar și în consola de terminal.

Bibliografie

- [1] *The Successive Approximation Analog to Digital Converter*. London: Springer London, 2008, pp. 89–91. [Online]. Available: https://doi.org/10.1007/978-1-84800-119-0_13
- [2] Atmel. [Online]. Available: <https://en.wikipedia.org/wiki/Atmel>
- [3] Autodesk-Tinkercad. [Online]. Available: <https://www.tinkercad.com/>
- [4] B. Craft, *Arduino Projects For Dummies*, 1st Ed. England: For Dummies, 2013.
- [5] D. Floroian and G. Macesanu, *Programarea si utilizarea sistemelor cu microprocesoare*. Brasov: Ed. Universitatii Transilvania, 2014.
- [6] FTDI. [Online]. Available: <https://en.wikipedia.org/wiki/FTDI>
- [7] Microchip. [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf
- [8] PointTutorials. [Online]. Available: https://www.tutorialspoint.com/cprogramming/c_constants.html
- [9] D. Ungureanu, *Programare obiectuală folosind C++*. Braşov: Ed. Universităţii Transilvania, 2009.