```
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 Aplicaţie 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 si 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 foloseste 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.

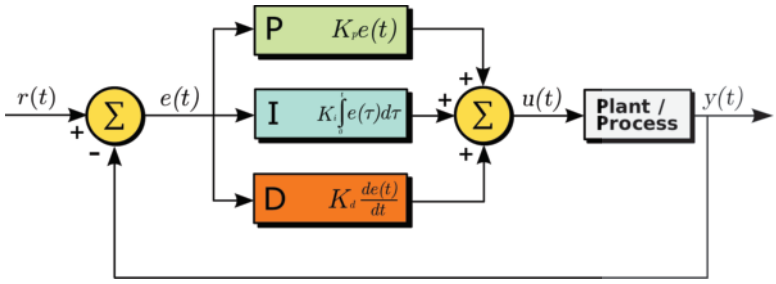

Fig. 7.5 Controlul unui proces utilizându-se un regulator PID.

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 **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 parametrii: 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 Kp, Ki, Kd sunt predefinite. După determinarea mărimii de ieșire se calculează o valoare temporară a erorii, care o sa 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
        ↪ encoderul de pe motor
123     const float dist = encoder_ticks * wheelPerimeter /
        ↪ ticksPerRevolution;
124     //determinam viteza motorului
125     speed = dist * invTe; // V = d / T
126     //Determinam marimea de comanda, folosind regulatorul PID
127     const float pwm = calculareIesirePID(&pidMotor, speed);
128     //Setam semnalul de iesire pentru controlul motorului
129     analogWrite(3, pwm);
130     //reinitializam numarul de fante citite de la encoder
131     encoder_ticks = 0;
132
133   //activare intreruperi globale
134   SREG |= (1 << SREG_I );
135 }
136
137 ISR(INT0_vect)
138 {
139     // dezactivare intreruperi globale
140   SREG &= ~(1 << SREG_I);
141
142    encoder_ticks++;
143
144   // activare intreruperi globale
145     SREG |= 1 << SREG_I;
146 }
```

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