10. Urmărirea formelor

Urmărirea formelor utilizând estimatori de stare
Filtrul Kalman

În această aplicaţie se va prezenta conceptul de urmărire a formelor într-o secvență video utilizându-se estimatori de stare. În mod particular, se va implementa o buclă de urmărire a poziției 2D a unui obiect cu ajutorul filtrului Kalman [**?**].

10.1 Baze teoretice

10.1.1 Estimatori de stare şi urmărirea formelor

Luând în considerare exemplul de urmărire a mişcării unei persoane într-o secvenţă video, în fiecare imagine a secvenţei de intrare se va determina poziţia acelei persoane. Acest lucru se poate efectua prin diverse metode, fiecare metodă livrând un estimat al poziţiei persoanei în cauză. Estimatul este de obicei imprecis, în special datorită naturii sistemelor de vedere artificială. Acest lucru se datorează lipsei de precizie a senzorului video, aproximărilor din stagiile de segmentare, ocluziilor, umbrelor, sau a modificării siluetei persoanei datorită mişcării braţelor şi a picioarelor. Indiferent de sursa de zgomot, măsurătorile efectuate prin tehnicile de procesare de imagini variază stohastic în jurul valorii reale ce ar fi măsurată de un senzor ideal. Toate aceste imperfecţiuni sunt sumate ca şi zgomot în sistemul vizual de urmărire a formelor.

Scopul unui sistem de urmărire a obiectelor este de a estima mişcarea lor în aşa fel încât să se utilizeze la maxim măsurătorile efectuate. Astfel, efectul cumulativ al unui număr de măsurători poate fi utilizat în observarea traiectoriilor reale ale obiectelor vizualizate. Ingredientul cheie într-un astfel de proces este acela al includerii unui *model de mişcare*[1] al obiectelor. În cazul exemplului de mai sus, un astfel de model poate fi considerat ca și informaţia apriori că o persoană intră în cadrul vizual din stânga şi îl părăseşte prin dreapta.

După cum se poate vedea în Fig. 10.1, acest proces este divizat în două stagii:

- *Predicţia*, unde informaţia deja achiziţionată este utilizată pentru predicţia următoarei locaţii a obiectului;
- *Corecţia*, unde o nouă măsurătoare este efectuată şi apoi fuzionată cu modelul luat în considerare.

Tehnicile utilizate în sistemele de predicţie-corecţie sunt denumite *estimatoare*. Unele dintre cele mai utilizate metode de estimare sunt aşa-numitele filtre *Kalman* [**?**] şi de *particule* [**?**].

10.1.2 Filtrul Kalman

Filtrul Kalman (KF) este în principiu un estimator de stare, care determină cea mai bună aproximare (în sensul erorii pătratice) pentru variabilele de stare asociate unui sistem dinamic

[1] Eng. Motion model

Fig. 10.1 Estimator în buclă închisă bazat pe o fază de predicţie urmată de corecţia modelului.

liniar, în care se exercită perturbaţii cu caracter aleatoriu, pe baza mărimilor măsurabile, care sunt, de asemenea, afectate de zgomote aleatoare.

Dezvoltarea KF este nemijlocit legată de sistemele stohastice. În aceste condiţii se poate considera că filtrul Kalman se poate baza pe ipoteza că sistemul care urmează să fie estimat poate fi modelat ca un proces aleatoriu normal distribuit $X(k)$, având valoarea medie $\bar{x}_k$ (în acest caz $\bar{x}_k = \hat{x}_k$) şi matricea de covarianţă a erorii P_k, k reprezentând momentul de timp. Mărimea $\hat{x}_k$ este cunoscută sub numele de *stare estimată* a stării reale necunoscute x_k a sistemului la momentul de timp k. Starea sistemului poate fi reprezentată, în cazul general, ca un vector n-dimensional:

$$x_k = \begin{bmatrix} x_{k_1} & x_{k_2} & \dots & x_{k_n} \end{bmatrix}^T. \tag{10.1}$$

Scopul urmărit este acela de a se obţine o stare estimată $\hat{x}_k$ cât mai apropiată de starea sistemului la momentul de timp considerat k sau, cu alte cuvinte, ca eroarea de estimare să tindă la zero.

10.1.2.1 Modelul procesului

Examinat de-a lungul unei perioade de timp, sistemul este supus unor transformări. Unele aspecte legate de aceste transformări sunt cunoscute şi pot fi modelate. Altele sunt necunoscute, nu pot fi măsurate sau sunt prea complexe pentru a fi modelate. Aceste transformări trebuie să fie aproximate de un model A_k al procesului. În cazul KF, acest model trebuie să fie liniar. În conformitate cu această condiţie, distribuţia normală a stării modelului este menţinută şi după ce starea a fost supusă transformării liniare A_k. Noua valoare estimată a stării $\hat{x}_k$ şi matricea de covarianţă P_k, pentru următorul moment de timp, sunt date de relaţiile:

$$\hat{x}_k = A_{k-1}\hat{x}_{k-1}, \tag{10.2}$$

$$\hat{P}_k = A_{k-1}P_{k-1}A_{k-1}^T. \tag{10.3}$$

unde P_k reprezintă matricea de covarianţă a erorii ce descrie eroarea dintre starea estimată $\hat{x}_k$ şi starea reală necunoscută x_k.

Datorită caracterului aproximativ al lui A_k, valoarea estimată $\hat{x}_k$ a stării este, de asemenea, o valoare aproximată a stării reale x_k. Diferenţa dintre starea reală şi starea estimată este reprezentată cu ajutorul unei variabile aleatoare w_k:

$$x_k = A_{k-1}x_{k-1} + w_{k-1}. \tag{10.4}$$

Valorile variabilei w_k, pentru fiecare moment de timp, sunt necunoscute, dar ele trebuie

să fie utilizate în procesul de îmbunătăţire a stării estimate. Vectorul w_k mai este cunoscut şi sub numele de *vectorul zgomotului procesului* şi este notat cu:

$$p(w_k) = N(0, Q_k), \tag{10.5}$$

unde zero este valoarea medie a distribuţiei, iar Q_k reprezintă *matricea de covarianţă a zgomotului procesului* (ţinând seama de semnificaţia zgomotului precum şi de caracterul aleatoriu al variaţiei sale în timp, necunoscută în sens determinist, se poate considera că funcţia $p(w_k)$ este un proces stohastic centrat (cu medii statistice nule) şi necorelat, mai precis, zgomot alb caracterizat prin matricea de covarianţă Q_k). Deci fiecare element component al vectorului w_k, la fiecare moment de timp, se poate presupune că are o valoare egală cu valoarea medie a distribuţiei, adică cu zero.

10.1.2.2 Ieşirea sistemului

Ieşirea sistemului este în strânsă legătură cu starea acestuia. Dacă această relaţie este cunoscută şi totodată este cunoscută şi starea estimată la momentul de timp imediat următor momentului de timp curent, atunci poate fi estimată mărimea măsurabilă corespunzătoare ieşirii sistemului. În continuare, va fi introdus modelul mărimii măsurabile corespunzătoare ieşirii sistemului şi va fi obţinută relaţia dintre starea şi ieşirea acestuia.

La fel ca şi în cazul stării sistemului, ieşirea acestuia poate fi modelată ca un proces aleatoriu normal distribuit $Z(k)$, având valoarea medie $\hat{z}_k$ şi matricea de covarianţă $S(k)$. Valoarea medie $\hat{z}_k$ reprezintă valoarea estimată a mărimii măsurabile corespunzătoare ieşirii sistemului, valoare care depinde de starea estimată $\hat{x}_k$, la momentul de timp k. Mărimea măsurabilă reală z_k poate fi obţinută prin măsurarea ieşirii sistemului. Această mărime măsurabilă poate fi reprezentată ca un vector m-dimensional:

$$z_k = \begin{bmatrix} z_{k_1} & z_{k_2} & \dots & z_{k_m} \end{bmatrix}^T. \tag{10.6}$$

10.1.2.3 Modelul mărimii măsurabile

Relaţia dintre ieşirea sistemului şi starea acestuia poate fi aproximată cu ajutorul modelului H_k al mărimii măsurabile, H_k fiind o matrice cu dimensiunea $m \times n$. După obţinerea unei stări estimate, se poate utiliza matricea H_k pentru a se determina valoarea estimată a mărimii măsurabile $\hat{z}_k$ şi matricea de covarianţă S_k:

$$\hat{z}_k = H_k \hat{x}_k, \tag{10.7}$$

$$S_k = H_k P_k H_k^T. \tag{10.8}$$

10.1.2.4 Predicţia şi corecţia

Pentru corectarea stării estimate a sistemului se utilizează diferenţa dintre valoare estimată $\hat{z}_k$ şi valoarea reală z_k a mărimii măsurabile deci, aşa cum s-a mai precizat, eroarea de estimare. Schema bloc a filtrului Kalman este reprezentată în Fig 10.2.

10.2 Cerinţe

1. Să se citească un fișier video de pe HDD;
2. Să se segmenteze un obiect de interes în funcție de culoarea sa;
3. Să se determine centrul de greutate al obiectului segmentat;
4. Să se calculeze predicția și corecția stării (poziției 2D) obiectului utilizându-se filtrul

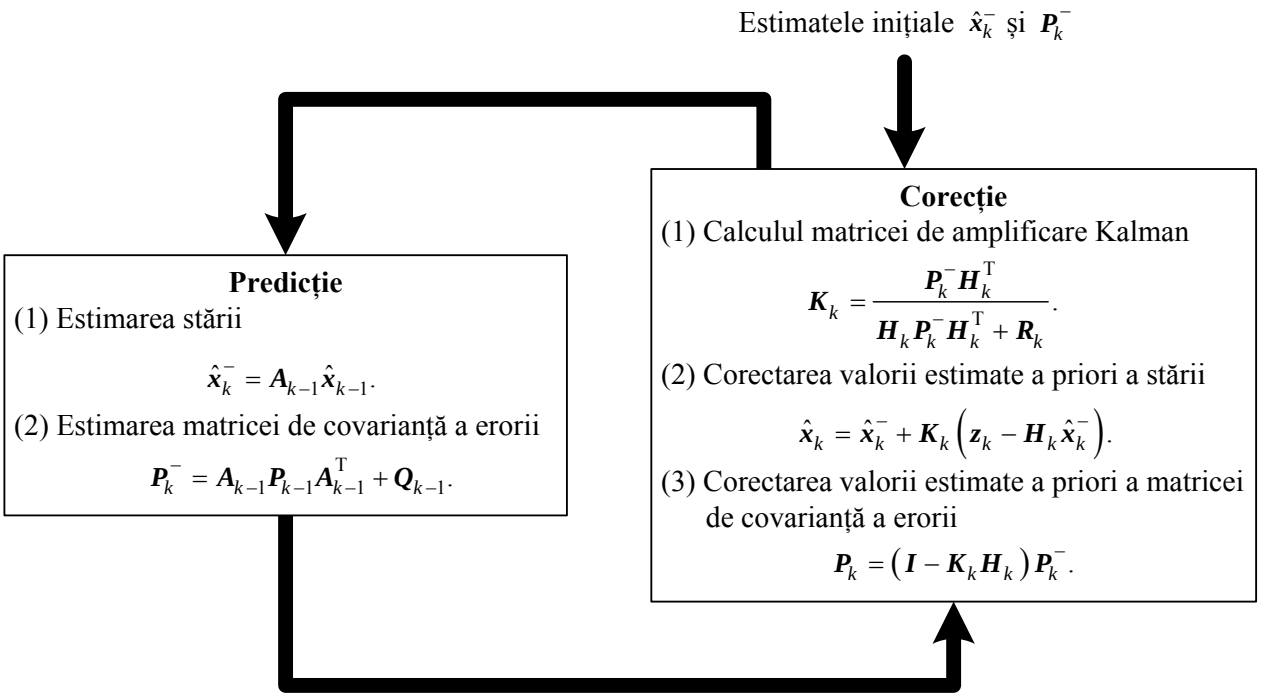

Fig. 10.2 Bucla de predicție-corecție a filtrului Kalman.

Kalman;

5. Să se afișeze traiectoriile poziției măsurate și estimate a obiectului.

10.3 Codul sursă al aplicației

```
1  #include <stdlib.h>
2  #include "opencv2/opencv.hpp"
3  #include "opencv2/highgui/highgui.hpp"
4  #include "opencv2/video/tracking.hpp"
5
6  using namespace cv;
7
8  int main(int argc, char ** argv)
9  {
10   Mat ImagineIntrare, ImagineTracking;
11   Mat ImagineHSV, ImagineSegmentata;
12
13   // Vector al traiectoriei obiectului
14   std::vector <Point> TraiectorieMasurata;
15   std::vector <Point> TraiectorieEstimata;
16
17   // Instanta filtrului Kalman
18   KalmanFilter KF(4, 2, 0);
19
20   // Matricile de stare, zgomot si masuratori
21   Mat Stare(4, 1, CV_32F); /* (phi, delta_phi) */
22   Mat ZgomotProces(4, 1, CV_32F);
23   Mat Masuratori = Mat::zeros(2, 1, CV_32F);
24
```

```
25    // Initializarea matricei de stare
26    randn(Stare, Scalar::all(0), Scalar::all(0.1));
27
28    // Initializarea matricei de tranzitii intre stari
29    KF.transitionMatrix = *(Mat_<float>(4, 4) <<
30        1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1);
31
32    // Initializarea filtrului Kalman
33    setIdentity(KF.measurementMatrix);
34    setIdentity(KF.processNoiseCov, Scalar::all(1e-5));
35    setIdentity(KF.measurementNoiseCov, Scalar::all(1e-1));
36    setIdentity(KF.errorCovPost, Scalar::all(1));
37    randn(KF.statePost, Scalar::all(0), Scalar::all(0.1));
38
39    // Clasa utilizata in achizitia de imagini de la camera video a ↩
        calculatorului
40    VideoCapture cap(0);
41
42    // Verificare deschidere camera video
43    if(!cap.isOpened())
44      return -1;
45
46    // Bucla infinita de achizitie de imagini
47    for(;;)
48    {
49      double AriaMaxima(0.0);
50
51      std::vector< std::vector<Point> > Contururi;
52      std::vector<Vec4i> Ierarhie;
53      Point ptPozitiePredictionata;
54      Point ptPozitieMasurata;
55      Point ptPozitieEstimata;
56
57      /*
58       * Predictie pozitie utilizand filtrul Kalman
59       */
60      Mat PredictieKalman = KF.predict();
61      ptPozitiePredictionata.x = PredictieKalman.at<float>(0);
62      ptPozitiePredictionata.y = PredictieKalman.at<float>(1);
63
64      /*
65       * Efectuarea masuratorilor (determinarea pozitiei in imagine a ↩
           obiectului)
66       */
67      // Achizitie imagine de intrare
68      cap >> ImagineIntrare;
69
70      // Conversita imaginii din spatiul de culoare RGB in spatiul HSV
71      cvtColor(ImagineIntrare, ImagineHSV, CV_BGR2HSV);
72
73      // Segmentarea planurilor H, S si V din imaginea HSV
74      inRange(ImagineHSV,
```

```
75          Scalar(10, 150, 100),
76          Scalar(40, 255, 255),
77          ImagineSegmentata);
78
79      // Extragerea contururilor din imaginea segmentata
80      findContours(ImagineSegmentata.clone(),
81          Contururi,
82          Ierarhie,
83          CV_RETR_TREE,
84          CV_CHAIN_APPROX_SIMPLE,
85          Point(0, 0));
86
87      // Extragerea conturului cu cea mai mare arie
88      for (size_t i = 0; i < Contururi.size(); i++)
89      {
90        // Calculul ariei conturului curent
91        double Aria = contourArea(Contururi[i]);
92
93        // Comparatie arie
94        if (Aria > AriaMaxima)
95        {
96          AriaMaxima = Aria;
97
98          // Determinare centru de greutate al obiectului utilizand ↩
              momentele spatiale
99          Moments MomenteSpatiale = moments (Contururi[i], false);
100
101         ptPozitieMasurata = Point2f(MomenteSpatiale.m10 / ↩
              MomenteSpatiale.m00,
102              MomenteSpatiale.m01 / MomenteSpatiale.m00);
103       }
104     }
105
106     // Salvarea pozitiei masurate in imagine a obiectului
107     TraiectorieMasurata.push_back(ptPozitieMasurata);
108
109     /*
110      * Corectia starii obiectului (pozitiei) utilizand noile ↩
          masuratori
111      */
112     Masuratori.at<float>(0) = ptPozitieMasurata.x;
113     Masuratori.at<float>(1) = ptPozitieMasurata.y;
114
115     Masuratori += KF.measurementMatrix * Stare;
116
117     Mat PozitieEstimata = KF.correct(Masuratori);
118
119     Stare = KF.transitionMatrix * Stare + ZgomotProces;
120
121     ptPozitieEstimata.x = PozitieEstimata.at<float>(0);
122     ptPozitieEstimata.y = PozitieEstimata.at<float>(1);
123
```

```
124     // Salvarea pozitiei estimate a obiectului in imagine
125     TraiectorieEstimata.push_back(ptPozitieEstimata);
126
127
128     /*
129      * Afisare rezultate tracking
130      */
131     ImagineTracking = ImagineIntrare.clone();
132
133     // Desenarea traiectoriei masurate si estimate obiectului
134     for (size_t i = 1; i < TraiectorieMasurata.size(); i++)
135     {
136       line(ImagineTracking,
137           TraiectorieMasurata[i-1],
138           TraiectorieMasurata[i],
139           CV_RGB(255, 255, 0),
140           2);
141
142       line(ImagineTracking,
143           TraiectorieEstimata[i-1],
144           TraiectorieEstimata[i],
145           CV_RGB(0, 0, 255),
146           2);
147     }
148
149     // Afisarea marimii predictionate a obiectului
150     circle(ImagineTracking, ptPozitiePredictionata, 6, CV_RGB(100, ↩
          255, 0), 3);
151
152     // Afisarea marimii masurate
153     circle(ImagineTracking, ptPozitieMasurata, 6, CV_RGB(255, 255, 0)↩
          , 3);
154
155     // Afisarea marimii determinata de filtrul Kalman
156     circle(ImagineTracking, ptPozitieEstimata, 6, CV_RGB(0, 0, 255), ↩
          3);
157
158     // Afisare rezultate
159     imshow("Segmentare", ImagineSegmentata);
160     imshow("Tracking", ImagineTracking);
161
162     if(waitKey(30) >= 0) break;
163   }
164
165   return EXIT_SUCCESS;
166 }
```

10.4 Descrierea funcțiilor principale

```
18 KalmanFilter::KalmanFilter(int dynamParams, int measureParams)
```

Constructorul unui filtru Kalman standard.

- `dynamParams`: dimensiunea vectorului de stare;

- `measureParams`: dimensiunea vectorului măsurătorilor.

60 `const Mat& KalmanFilter::predict()`

Calculul predicției stării sistemului.

117 `const Mat& KalmanFilter::correct(const Mat& measurement)`

Corecția stării sistemului utilizând factorul de amplificare Kalman.

- `measurement`: măsurătorile efectuate asupra procesului.