Zaawansowne algorytmy wizyjne

materiały do ćwiczeń laboratoryjnych

Piotr Pawlik, Tomasz Kryjak

PUBLISHED BY AGH

First printing, March 2018

Contents

1 — Przepływ optyczny

1.1 Cel zajęć

- zapoznanie z zagadnieniem przepływu optycznego,
- poznanie zakresu stosowalności przepływu optycznego i jego ograniczeń,
- poznanie wykorzystania wielu skal w wyznaczaniu przepływu optycznego,
- implementacja metody blokowej,
- zapoznanie z metodami dostępnymi w bibliotece OpenCV – Lucas-Kanade oraz Farneback,
- wykorzystanie przepływu optycznego do segmentacji obiektów ruchomych,
- klasyfikacja obiektów jako sztywne lub nie-sztywne z wykorzystaniem przepływu optycznego.

1.2 Przepływ optyczny (ang. *optical flow*)

Przepływ optyczny to pole wektorowe opisujące ruch pikseli (ew. tylko wybranych tj. punktów charakterystycznych) pomiędzy dwoma kolejnymi ramkami z sekwencji wideo. Inaczej mówiąc, dla każdego piksela na ramce I_{n-1} wyznaczamy wektor przesunięcia w rezultacie którego otrzymamy obraz I_n. Warto jednak zaznaczyć, że przepływ można (a niekiedy wręcz jest to pożądane) obliczać dla ramek o odstępie większym niż '1'.

Przepływ optyczny może być gęsty (ang. *dense*) lub rzadki (ang. *sparse*). W pierwszym przypadku przemieszczenie wyznaczane jest dla każdego piksela, a drugim tylko dla pewnego ich ograniczonego zbioru.

Podstawą działania przepływu optycznego jest „śledzenie" pikseli (lub pewnych lokalnych konfiguracji pikseli tj. otoczeń) pomiędzy ramkami. Zakłada się przy tym, że jasność (kolor) piksela jest stała.

Niestety, jak pokaże ćwiczenie, podejście nie sprawdza się w kilku przypadkach, z których najważniejszy to duże jednorodne powierzchnie. Dlaczego tak się dzieje? Wyobraźmy sobie dość duże kartonowe pudełko. Ma ono szare, jednorodne ściany bez tekstury oraz oświetlone jest równomiernym światłem. Chcemy wyznaczyć przepływ optyczny dla piksela „ze środka" ścianki pudełka. Czy uda nam się znaleźć na kolejnej ramce dokładnie ten punkt? Czy przypisanie jednoznaczne jest możliwe?

Z podanych powyżej przyczyn czasem korzystniej jest zastosować przepływ rzadki, obliczany w uprzednio określonych lokalizacjach. Tutaj mamy dwie możliwości. Albo arbitralnie wybier-

amy punkty np. na kwadratowej lub prostokątnej siatce (wtedy celem jest redukcja czasu obliczeń) albo wyszukujemy tzw. punkty charakterystyczne – mogą to być krawędzie, narożniki lub inne „wyraziste" elementy obrazu. Istnieje szereg algorytmów ich detekcji np. detektor narożników Harrisa, punkty charakterystyczne SIFT lub SURF itp.

W literaturze opisano bardzo dużo metod – proszę zobaczyć na ranking dostępny na stronie `http://vision.middlebury.edu/flow/eval/`. Dwie najpopularniejsze to Horn-Schunck (HS) i Lucas-Kanade (LK). Druga z nich jest dostępna w bibliotece OpenCV.

1.3 Implementacja metody blokowej

1. Utwórz nowy skrypt w języku Python. Wczytaj obrazy *I.jpg* i *J.jpg*. Będą to dwie ramki z sekwencji, oznaczone odpowiednio jako I i J. Wykonaj ich konwersję do odcieni szarości – `cvtColor`. Wyświetl zadane obrazy. Zwizualizuj różnicę wykorzystując polecenie: `absdiff`.
2. Wykorzystując poniższe wskazówki zaimplementuj metodę blokową wyznaczania przepływu optycznego.
   Przyjmij następujące założenia – porównujemy fragmenty obrazu o rozmiarze 3×3 (wykorzystywane dalej oznaczenia (oraz przyjęte wartości dla 3x3):
   $W2 = 1$ ('podłoga' z połowy rozmiaru okna),
   $dX = dY = 1$ (rozmiar przeszukiwania (przesuwania okna) w obu kierunkach).
3. Implementację należy zacząć od dwóch pętli `for` po obrazie. Wykorzystaj parametr $W2$ do „obsługi" przypadku brzegowego – zakładamy, że na brzegu nie wyliczamy przepływu optycznego.
4. Wewnątrz pętli „wycinamy" otocznie z ramki I. Dla przypomnienia niezbędna składnia to: `IO = np.float32(I[j-W2:j+W2+1,i-W2:i+W2+1])`. Uwaga. W rozważaniach przyjmujemy, że `j` – indeks pętli zewnętrznej (po wierszach), `i` – indeks pętli wewnętrznej (po kolumnach). Proszę zwrócić uwagę na składnik „+1" - w Pythonie górny zakres to wartość o 1 większa od największej przyjmowanej wartości - inaczej niż w Matlabie. Konwersja na typ zmiennoprzecinkowy potrzebna jest do dalszych obliczeń.

5. Następnie realizujemy kolejne dwie pętle `for` – przeszukiwanie otoczenia piksela `I(j,i)`. Mają one zakres od $-dX$ do dX i $-dY$ do dY. Proszę nie zapomnieć o „+1".
   Wewnątrz pętli należy sprawdzić, czy współrzędne aktualnego kontekstu (tj. jego środek) mieści się w dopuszczalnym zakresie. Alternatywnie można też zmodyfikować zakres „zewnętrznych" pętli – tak, aby wykluczyć dostęp do pikseli spoza dopuszczalnego zakresu. W tym przypadku dla nieco szerszego brzegu przepływ nie zostanie określony, jednak nie ma to istotnego znaczenia praktycznego.
   Wycinamy otoczenie `JO`, dokonujemy konwersji na *float32* a następnie obliczamy "odległość" między wycinkami `IO`, a `JO`. Można to wykonać instrukcją:
   `np.sum(np.sqrt((np.square(JO-IO))))`.
   Spośród wszystkich 9-ciu wycinków należy znaleźć najmniejszą "odległość" – lokalizację „najbardziej podobnego" do IO fragmentu na obrazie J.
6. Uzyskane współrzędne znalezonych minimów należy zapisać w dwóch macierzach (przykładowo `u` i `v`) – należy je wcześniej (przed główną pętlą) utworzyć i zainicjować zerami (funkcja `np.zeros`).
7. Wyznaczone pole przepływu optycznego można zwizualizować. Służy do tego funkcja `plt.quiver` (z biblioteki *matplotlib*). Uwaga - funkcja quiver domyślnie rysuje punkt (0,0) wykresu w lewym dolnym rogu. Dla zgodności z obrazwm konieczne jest przestawienie początku układu w lewy dolny róg - mozna to zrobić poleceniem:

```
plt.gca().invert_yaxis()
```

1.4 Implementacja wyliczania w wielu skalach

Wykrywanie dużych przemieszczeń zaimplementowaną metodą jest złożone obliczeniowo. Wynika z konieczności użycia dużych wartości parametrów dX i dY, a zatem dla większości lokalizacji (oprócz brzegowych) konieczne będzie sprawdzenie podobieństwa większej liczby kontekstów (więcej operacji SAD). Można wykonać test i zmienić wskazane parametry w swoim kodzie.

Lepszym, a także bardzo często stosowanym, pomysłem jest przetwarzanie obrazu w wielu skalach. Idea pokazana jest na rysunku 1.4.

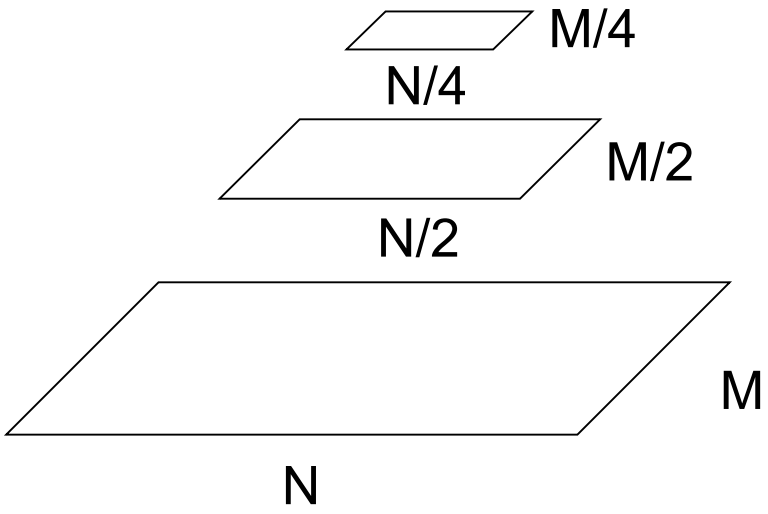

Figure 1.1: Przykład konstrukcji piramidy obrazów (wielu skal)

Schemat działania metody:

- wykonujemy przeskalowanie obrazu do skal: L_0 – rozdzielczość oryginalna, L_1, .. , L_m (najmniejszy obraz),
- obliczamy przepływ optyczny dla skali L_m (np. metodą blokową),
- propagujemy wyniki obliczania przepływu na skalę L_{m-1} – wykorzystujemy je jako wstępne przybliżenie przepływu na poziomie L_{m-1},
- obliczamy dokładniejsze wartości przepływu na poziomie L_{m-1} – (np. metodą blokową),
- postępowanie realizujemy aż do poziomu L_0.

1. Na wstępie proszę stworzyć kopię dotychczas stworzonego skryptu. Następnie należy utworzyć **funkcję** do obliczania przepływu optycznego dla wybranej skali.
   Funkcja powinna mieć następującą postać:
   `def of(I, J, u0, v0, W2=1, dY=1, dX=1):`
   Poszczególne parametry to `I` i `J` – obrazy, `u0,v0` – wstępnie obliczony przepływu optyczny, `W2, dX, dY` – parametry metody (identyczne jak 1.3).
2. Sposób działania funkcji powinien być bardzo zbliżony do stworzonej w rozdziale 1.3. Zasadnicza modyfikacja to uwzględnienie wstępnego przybliżenia przepływu wyznaczonego w poprzedniej iteracji. W tym celu w wewnętrznej pętli `for` (szukanie wzorca w otoczeniu) przy określaniu położenia drugiego kontekstu należy uwzględnić wstępne przesunięcie tj. `u0` i `v0`. Inaczej mówiąc środek otoczenia `JO` należy przesunąć o wektor `[u0,v0]`. Druga modyfikacja to analogiczne przesunięcie wyznaczonej wartości

przepływu – położenia minimum w macierzy `dd`. Wewnątrz funkcji należy dodać jeszcze tworzenie „pustych" macierzy `u, v`.

3. Po napisaniu funkcji dobrze jest przetestować jej działanie dla przypadku skali L_0 – wynik powinien być identyczny jak otrzymywany wcześniej. Oczywiście zakładamy, że początkowe przybliżenie przepływu `u0` i `v0` są zainicjowane wartościami '0'.
4. Implementacja przetwarzania w skalach wymaga kilku modyfikacji w „nadrzędnym" skrypcie. Po pierwsze należy wygenerować piramidę obrazów. Można wykorzystać następującą funkcję:

```
def pyramid(im, max_scale):
  images=[im];
  for k in range(1, max_scale):
    images.append(cv2.resize(images[k-1], (0,0), fx=0.5, fy=0.5))
  return images
```

W tym przypadku zakładamy, że pomniejszenia rozdzielczości zawsze będą dwukrotne. Czyli pomniejszamy obraz o 2, 4, 8 itp. Na potrzeby eksperymentu zakładamy, że ograniczmy się do 3 skal.
Ponadto musimy „obsłużyć" wstępne przybliżenie przepływu optycznego. Dla skali L_m należy zainicjować macierze `u0` i `v0` wartościami 0. Można wykorzystać następujący kod:

```
u0 = np.zeros(IP[-1].shape, np.float32)
v0 = np.zeros(JP[-1].shape, np.float32)
```

`IP, JP` to wygenerowane piramidy obrazów. Proszę zwrócić uwagę na składnię `[-1]` – dostęp do ostatniego elementu.
Kolejny element to pętla po skalach. W pierwszej iteracji nie mamy do dyspozycji wstępnych przybliżeń – zatem wywołujemy po prostu funkcję obliczającą przepływ optyczny. Dla kolejnych należy dokonać przeskalowania przepływu z poprzedniej skali tj. `v=cv2.resize(v, (0,0), fx=2, fy=2, interpolation=cv2.INTER_NEAREST)`.
Szczegóły implementacji do samodzielnej analizy.
5. Porównaj działanie metody ze skalami i bez

Bibliography