Eliminacija pozadine u video zapisima

Seminarski rad u okviru kursa
Naučno izračunavanje
Matematički fakultet

Vesna Katanić, Anja Ivanišević

14. septembar 2019.

Sadržaj

1 Uvod

Algoritmi za eliminaciju pozadine u video zapisima su algoritmi koji imaju za cilj detekciju kretanja i odvajanje pozadine od objekata koji se kreću. Problem eliminacije pozadine je bitan problem obrade slika i

2 Primenjeni algoritmi

U ovom radu smo poredili tri različita algoritma za eliminaciju pozadine u video zapisima. Ti algoritmi su: *Basic motion detection*, *Gaussian Mixture Model* i *K Nearest Neighbours*. Ideja svakog od ovih algoritama je da transformišu ulazni video u novi video u kojem će pozadina biti ofarbana u crno a objekti koji se kreću u belo, kako bi se oni izdvojili. U nastavku rada će ovi algoritmi biti predstavljeni.

2.1 Basic Motion Detection

Basic Motion Detection je najjednostavniji algoritam od predstavljenih algoritama. U ovom algoritmu se polazi od pretpostavke da se video I sastoji od statičke pozadine B ispred koje se nalaze objekti koji se kreću. Kako bismo detektovali objekte računa se rastojanje trenutnog modela pozadine i posmatranog frejma. Na osnovu ovog rastojanja pravi se rezultujuća crno-bela slika.

Model pozadine se ažurira na osnovu prethodnog stanja i trenutno posmatranog frejma po sledećoj formuli:
$B_{s,t+1} = (1 - \alpha) * B_{s,t} + \alpha * I_{s,t}$

```
class SkipListNode(object):

    def __init__(self, key=None, level=None):
        self.key = key  # vrednost čvora
        self.forward =[None]*(level+1)  # niz pokazivaca
        self.d = [None]*(level+1)  # niz pomoćnih promenljivih
```

Prilikom umetanja novog elementa u skip listu potrebno je da čuvamo niz *update* u koji za svaki nivo smeštamo poslednji čvor koji smo obišli dok smo tražili mesto za naš novi element. Na slici 1. je crvenom bojom označen put i elementi koji bi u tom primeru bili u nizu *update*. Niz *update* će nam biti potreban da bismo prevezali pokazivače i time dodali novi element u skip listu. Sada kad smo dodali novi element, upravo elementima iz niza *update* bi trebalo ažurirati niz d. Takođe potrebno je i inicijalno izračunati vrednosti niza d za novododati element.

Ažuriranje niza d za elemente niza update sa indeksima većim od nivoa novododatog čvora je jednostavno, potrebno je samo povećati vrednost za jedan. Međutim, u slučaju indeksa koji su manji ili jednaki nivou novog čvora, vrednost d možemo izračunati na osnovu stare vrednosti i vrednosti d za novi čvor. Potrebno je od stare vrednost d elemenata niza update oduzeti vrednost d za novi čvor i dodati jedan. Slično se radi i u slučaju brisanja elemenata iz skip liste, samo što u tom slučaju umesto sabiranja radimo oduzimanje, i obrnuto.

Na ovaj način možemo jednostavnim lokalnim izmenama da ažuriramo vrednost d i time ne ugrozimo vremensku složenost algoritma.

3 Implementacija algoritma

Sada kada smo predstavili kako izgleda struktura skip liste, i kako se ona održava prilikom dodavanja novog elementa, ostaje nam samo da napravimo funkciju koja nalazi k-ti najveći element u skip listi. Pošto imamo pomoćni niz d u svakom čvoru, ovo će sada biti jednostavno.

```
def findKLargest(self, k):
    current = self.head
    pos = 0
    for i in range(self.level, -1, -1):
        # dokle god nismo prosli dovoljno elemenata pomeramo se
        # na sledeći pokazivač i ažuriramo broj elemenata pos
        while current.forward[i] and (pos + current.d[i]) <= k:
            pos = pos + current.d[i]
            current = current.forward[i]
        if pos == k:
            # kada dođemo do k vraćamo rezultat
            return current.key
```

Krećemo od prvog čvora na najvišem nivou i pomeramo se po tom nivou dokle god ne pređemo broj elemenata veći od k. To proveravamo sa $(pos + current.d[i]) <= k$. U svakom koraku pomeramo se na sledeći pokazivač i povećavamo broj pos. Na kraju ćemo doći do k-tog elementa i to vratiti kao rezultat. Vremenska složenost ovog algoritma za skip listu od n elemenata je $O(logn)$.

4 Efikasnost algoritma

Naredne dve slike prikazuju vreme izvršavanja umetanja i brisanja novog elementa u zavisnosti od broja elemenata u skip listi izražene u sekundama.

Slika 1: Vreme umetanja novog elementa izraženo u sekundama u zavisnosti od broja elemenata u skip listi

Slika 2: Vreme brisanja elementa izraženo u sekundama u zavisnosti od broja elemenata u skip listi