Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky
Katedra kybernetiky a umelej inteligencie

Zadanie z predmetu Počítačové videnie

Metóda Lucas-Kanade

Bc. Kamil Adamišín
Bc. Michal Budiš
Bc. Jakub Karľa

Akademický rok: 2019/2020

Inteligentné systémy

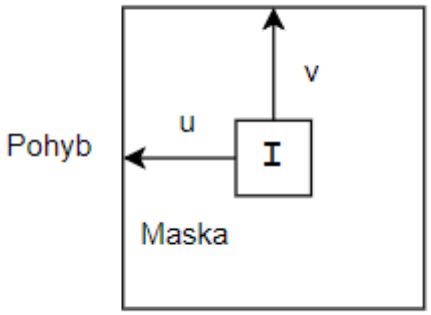
Obsah

1. Znenie zadania

Cieľom zadania je vytvorenie vlastného algoritmu optického toku pomocou metódy nazývanej Lucas-Kanade, ktorá bude sledovať optický tok objektov vo videu a porovnávať nami vytvorený algoristmus s algoritmom vytvoreným v knižnici OpenCV. Algoritmus bude takisto kresliť trajektórie, ktoré tieto body opisujú.

2. Metóda Lucas-Kanade

Za vznikom metódy Lucas-Kanade stoja dvaja počítačoví vedci Bruce D. Lucas a Takeo Kanade. Táto metóda sa často používa pri sledovaní optického toku bodov medzi jednotlivými po sebe idúcimi obrazmi scény napríklad vo videu, kde sa daný objekt pohybuje v scenérií alebo sa pohybuje kamera. Metóda predpokladá, že tok bodov je malý a konštantný v lokálnom susedstve daného bodu, to znamená, že lepšie funguje pri menšej zmene pohybu medzi obrazmi. Využíva kritérium najmenších štvorcov na výpočet rovníc optického toku pre všetky pixely v susedstve. Metóda nepoužíva farebné informácie obrazu, ale snaží sa odhadnúť, ktorým smerom a o akú vzdialenosť sa sledovaný objekt pohol. Každému pohybu sa priradzuje pohybový vektor (u,v) sledovaného bodu a čas medzi jednotlivými obrazmi v scéne.



Obrázok 1 – Pohyb pixelu obrazu

Ak vieme, že zvýšenie jasu z pixelu na pixel (x, y) v smere x je $I_x\ (q_i)$ a zvýšenie jasu z pixelu na pixel v smere y je $I_y\ (q_i)$, tak potom vieme napísať rovnicu zmeny jasu spôsobenú pohybom v smere x o dĺžku V_x a v smere V_y o dĺžku a to:

$$I_x\ (q_1)\ \cdot\ V_x + I_y\ (q_1)\ V_y = -I_t(q_1)$$

$$I_x\ (q_2)\ \cdot\ V_x + I_y\ (q_2)\ V_y = -I_t(q_2)$$

:

kde hodnoty jednotlivých q_i sú derivácie obrazu I vzhľadom na x, y a t. Na porovnávanie jednotlivých bodov medzi obrazmi sa používajú aj susedné pixely, keďže jeden pixel neobsahuje dostatočné množstvo informácií, aby sme ho presne vedeli identifikovať na

rôznych po sebe idúcich obrazoch daného objektu. Preto sa používajú susedné pixely napríklad nxn okolo daného pixelu. Následne nám vznikne viacero takýchto rovníc, ktoré si vieme prepísať do maticovej formy:

$$A = \begin{bmatrix} I_x\,(q_1) & I_y\,(q_1) \\ I_x\,(q_2) & I_y\,(q_2) \\ \vdots & \vdots \\ I_x\,(q_n) & I_y\,(q_n) \end{bmatrix} \qquad v = \begin{bmatrix} V_x \\ V_y \end{bmatrix} \qquad b = \begin{bmatrix} -I_t(q_1) \\ -I_t(q_2) \\ \vdots \\ -I_t(q_n) \end{bmatrix}$$

Tieto neznáme si vieme vypočítať pomocou metódy Lucas-Kanade na základe princípu najmenších štvorcov.

$$A^T A v = A^T b$$

$$v = (A^T A)^{-1} A b$$

Maticu A^T dostaneme traspozíciou matice A. Predošlú rovnicu si vieme prepísať do nasledujúceho tvaru pre i od 1 po n:

$$\begin{bmatrix} V_x \\ V_y \end{bmatrix} = \begin{bmatrix} \sum_i I_x\,(q_i)^2 & \sum_i I_x\,(q_i) I_y\,(q_i) \\ \sum_i I_y\,(q_i) I_x\,(q_i) & \sum_i I_y\,(q_i)^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i I_x\,(q_i) I_t(q_i) \\ -\sum_i I_y\,(q_i) I_t(q_i) \end{bmatrix}$$

3. Riešenie pomocou knižníc

Na prepočet spomínanej metódy Lucas-Kanade už v dnešnej dobe existujú vytvorené knižnice, ktoré nám po zadaní potrebných parametrov vedia vysokou presnosťou vypočítať optický tok bodov v daného videa. Pred týmto výpočtom si však ešte pomocou metódy na detekciu hrán nazývanej *ShiTomasi* nájdeme body na prvom obraze vo videu, ktoré chceme pomocou metódy Lucas-Kanade sledovať. Príkladom je aj programovací jazyk Python a príslušná knižnica *cv2*, ktorá už obsahuje nástroje na prepočet optického toku bodov pomocou metódy Lucas-Kanade.

Riešenie začína implementovaním knižníc numpy, cv2 a argparse.

```
import numpy as np
import cv2 as cv
import argparse
```

Ďalším krokom je načítanie videa buď z lokálnej pamäte počítača alebo načítaním z nejakej internetovej stránky. Po načítaní videa nasleduje zadanie parametrov pre spomínané metódy *ShiTomasi* a Lucas-Kanade.

```
# ShiTomasi detekcia hrán
feature_params = dict( maxCorners = 100,
                       qualityLevel = 0.3,
                       minDistance = 7,
                       blockSize = 7 )
# Lucas-Kanade
```

```
lk_params = dict( winSize  = (15,15),
                  maxLevel = 2,
                  criteria = (cv.TERM_CRITERIA_EPS | cv.TERM_CRITERIA_COUNT, 10, 0.03))
```

Pred spustením algoritmu Lucas-Kanade si prvý obraz zmeníme z farebného na šedotónový, nájdeme na ňom hrany spustením funkcie pre metódu ShiTomasi so zadanými parametrami a uložíme si masku prvého Framu.

```
ret, old_frame = cap.read()
old_gray = cv.cvtColor(old_frame, cv.COLOR_BGR2GRAY)
p0 = cv.goodFeaturesToTrack(old_gray, mask = None, **feature_params)
mask = np.zeros_like(old_frame)
```

Následne program vchádza do *While*, ktorý sa opakuje až po skončenie videa. Tento *While* obsahuje prepočet metódy Lucas-Kanade medzi jednotlivými Framami (obrazmi) stiahnutého videa. *While* začína načítaním nového obrazu z videa a prepočítaním posunu sledovaných bodov z prvého obrazu na druhý.

```
ret,frame = cap.read()
frame_gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
p1, st, err = cv.calcOpticalFlowPyrLK(old_gray, frame_gray, p0, None, **lk_params)
good_new = p1[st==1]
good_old = p0[st==1]
```

Súradnice z prvého a z druhého obrazu konkrétneho bodu sa uložia do premenných, v našom prípade *good_new* a *good_old* a trajektória medzi nimi sa zaznačí farebnou krivkou.

```
for i,(new,old) in enumerate(zip(good_new, good_old)):
        a,b = new.ravel()
        c,d = old.ravel()
        mask = cv.line(mask, (a,b),(c,d), color[i].tolist(), 2)
        frame = cv.circle(frame,(a,b),5,color[i].tolist(),-1)
```

Posledným krokom je zobrazenie krivky vo videu a uloženie súradníc bodov do premennej pre ďalší výpočet.

```
img = cv.add(frame,mask)
cv.imshow('frame',img)
k = cv.waitKey(30) & 0xff
if k == 27:
    break

old_gray = frame_gray.copy()
p0 = good_new.reshape(-1,1,2)
```

Ako bolo spomenuté, slučka *While* sa opakuje až po ukončenie videa.

Na obrázku číslo 2. môžeme vidieť nájdenie hrán alebo bodov na jednom z prvých obrazov vo videu a na obrázku číslo 3. vidíme znázornenie pohybu týchto hrán po prejdení istého času vo videu.

Obrázok 2 - Nájdené body pomocou metódy ShiTomasi

Obrázok 3 - Znázornenie optického toku vypočítaného metódou Lucas-Kanade

4. Vlastná funkcia pre výpočet algoritmu Lucas-Kanade

Programovanie metódy Lucas-Kanade pomocou vzorcov začína veľmi podobne. Prvým krokom je implementovanie knižníc do programu, ktoré v našom prípade používame.

```
import numpy as np
import cv2 as cv
from scipy import signal
import csv
import time
```

Podobne sme si načítali video, na ktorom budeme sledovať hrany na pohybujúcich sa objektoch, obraz sme si previedli na šedotónový a následne sme našli body pomocou metódy *ShiTomasi*, na ktorú sme použili knižnicu *cv2*.

```
filename = str('https://www.bogotobogo.com/python/OpenCV_Python/images/mean_shift_tracking/slow_traffic
_small.mp4')

feature_params = dict( maxCorners = 100,
                       qualityLevel = 0.3,
                       minDistance = 7,
                       blockSize = 7 )

vc=0
if(vc):
    capture= cv.VideoCapture(0)
else:
    capture= cv.VideoCapture(filename)

m, frame1 = capture.read()
height,width,_ = frame1.shape
im1 = cv.cvtColor(frame1, cv.COLOR_BGR2GRAY)

#Najdenie bodov
points = cv.goodFeaturesToTrack(im1, mask = None, **feature_params)
```

Masku prvého obrazu sme si uložili do premennej *mask*.

```
temp = []
mask = np.zeros_like(frame1)
```

Následne vchádzame do cyklu *While,* ktorý sa vykonáva až do skončenia videa. *While* začína načítaním nového obrazu a prevedením tohto obrazu na šedotónový. Na výpočet vektorov metódy Lucas-Kanade sme si vytvorili vlastnú funkciu, ktorú si zavoláme. Funkciou *time.time()* si zaznamenáme čas pred a po vykonaní prepočtov metódy Lucas-Kanade, aby sme si zistili rýchlosť výpočtu.

```
while(capture.isOpened()):
    check, frame2 = capture.read()
    im2 = cv.cvtColor(frame2, cv.COLOR_BGR2GRAY)
    start = time.time()
    newp=LucasKanadeCalc(im1,im2,points)
    end = time.time()
```

Popíšeme si fungovanie vlastnej funkcie *LucasKanadeCalc*, ktorej vstupom je predošlý obraz, nový obraz a súradnice bodov, ktoré sledujeme.

```
def LucasKanadeCalc(prev_frame, new_frame, points):
```

Prvým krokom funkcie je vytvorenie polí a načítanie hodnôt pre *x*, *y* a *t* z predošlého obrazu.

```
kernel_x = np.array([[-1, 1], [-1, 1]])
kernel_y = np.array([[-1, -1], [1, 1]])
kernel_t = np.array([[1, 1], [1, 1]])

mode = 'valid'
fx = signal.convolve2d(prev_frame, kernel_x, mode=mode)
fy = signal.convolve2d(prev_frame, kernel_y, mode=mode)
ft = signal.convolve2d(new_frame, kernel_t, mode=mode) + signal.convolve2d(prev_frame, -
kernel_t, mode=mode)
```

Vytvorili sme si pole, do ktorého si budeme načítavať nové body a nastavili sme si okolie sledovaného bodu na 15 pixelov od daného bodu, čiže mriežku 30x30.

```
w=15
new_points = np.zeros(points.shape)
k = 0
```

Nasleduje výpočet vektorov posunu pre všetky sledované body, ktorý nám zabezpečí cyklus FOR. V cykle sa najprv načítajú súradnice bodu, ktorý sledujeme a následne sa do premenných uloží spomínané okolie bodu. Tieto okolia prevedieme do premennej A a nasleduje samotný výpočet posunu bodov. Výsledný posun sa pripočíta k súradniciam bodu a nové súradnice bodu si uložíme do poľa. Táto naša funkcia po prepočítaní všetkých súradníc bodov vráti celý zoznam súradníc na ďalšie spracovanie. Celý priebeh cyklu FOR si môžeme pozrieť v nasledujúcom kód.

```
for p in points:
    i = int(p.item(1))
    j = int(p.item(0))
    Ix= fx[i-w:i+w, j-w:j+w].flatten()
    Iy= fy[i-w:i+w, j-w:j+w].flatten()
    It= ft[i-w:i+w, j-w:j+w].flatten()
    b = It
    A = np.vstack((Ix, Iy)).T
    nu = np.matmul(np.linalg.pinv(A), b)

    item1 = float(p.item(0)+nu[0])
    item2 = float(p.item(1)+nu[1])
    arraytemp = np.array([[item1,item2]])
    new_points[k] = arraytemp
    k=k+1
k=0
return np.array(new_points)
```

Po skončení funkcie *LucasKanadeCalc,* sa vraciame do cyklu *While,* kde si najprv vypočítame čas, za ktorý to spomínaná funkcia vypočítala a uložíme si to do poľa.

```
elapsed_time= end*1000 - start*1000
temp.append(elapsed_time)
```

V cykle *For* si rozdelíme súradnice z predchádzajúceho a nového obrazu a medzi týmito súradnicami si znázorníme krivku. Nové súradnice si poznačíme krúžkom.

```
for i,(n,p) in enumerate(zip(newp, points)):
    a = int(n.item(0))
    b = int(n.item(1))
    c = int(p.item(0))
    d = int(p.item(1))
    mask = cv.line(mask, (a,b),(c,d), (0,0,255), 2)
    frame2=cv.circle(frame2,(c,d),5,(0,0,255),2)
    img = cv.add(frame2,mask)
    cv.imshow('frame',img)
    if cv.waitKey(1) & 0xFF == ord('q'):
        break
```

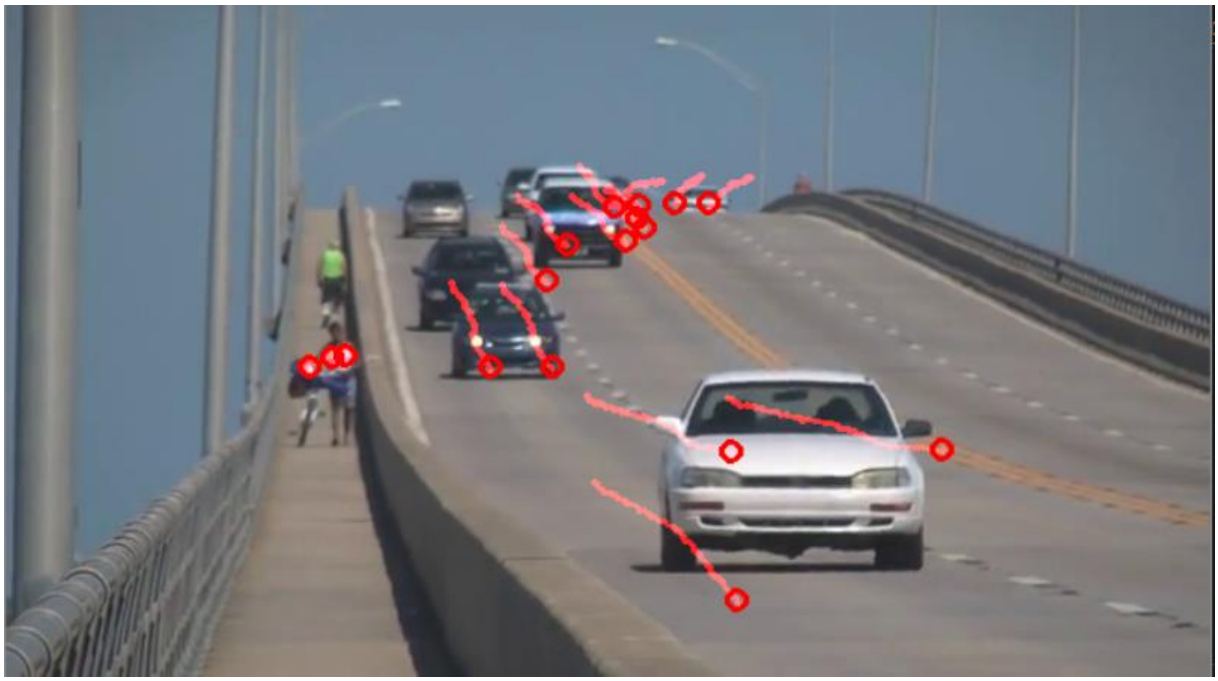
Na konci kódu si uložíme nový obraz a body ako staré pre ďalší výpočet a zapíšeme si dĺžky prepočtu metódy Lucas-Kanade do súboru *vlastnafunkcia.csv.*

```
im1=im2.copy()
points=np.asarray(newp).reshape(-1,1,2)
with open('vlastnafunkcia.csv', mode = 'w') as file:
    writer = csv.writer(file)
    writer.writerow([temp])
```

Po skončení celého cyklu *While* zatvoríme okno a ukončíme celý program.

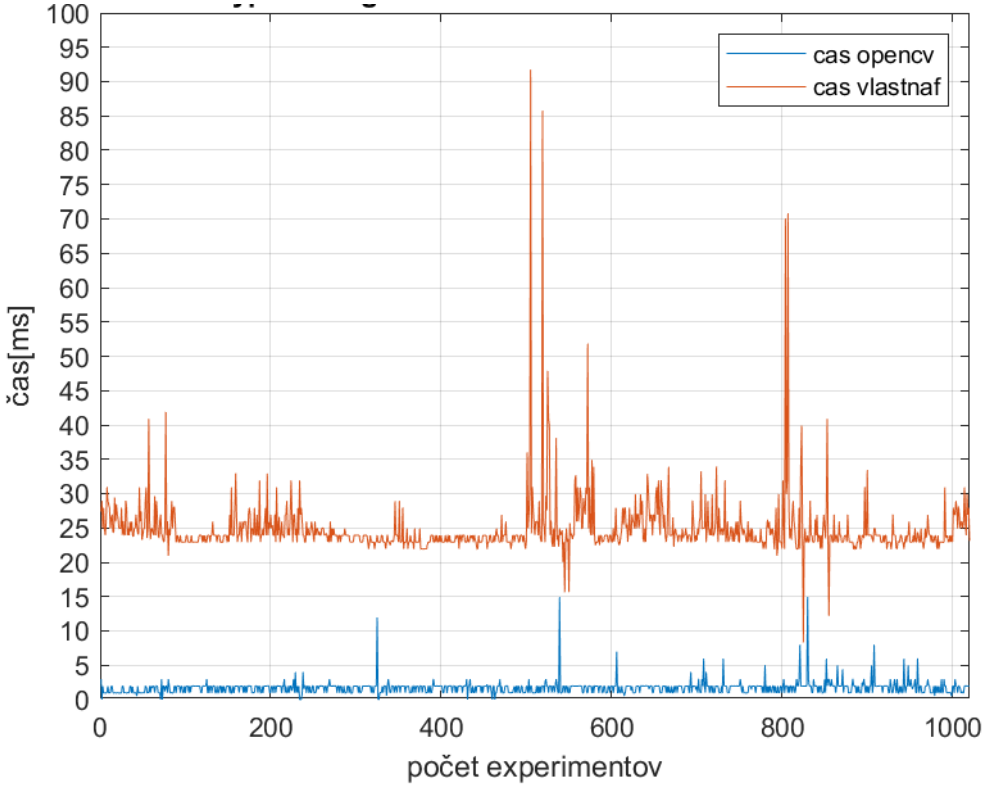
```
capture.release()
cv.destroyAllWindows()
```

Obrázok 4 - Detekované body obrazu



Obrázok 5 - Trajektórie bodov

5. Experimenty



Obrázok 6 - Graf trvania výpočtov funkcií

Porovnávali sme čas vykonania algoritmu medzi desiatym a jedenástym frame-om aby sme zabezpečili, že každé meranie má rovnaké podmienky. Ako je možné vidieť nami naprogramovaný algoritmu je pomalší ako algoritmus z OpenCV knižnice. Priemerný čas

výpočtu algoritmu medzi frameom 10 a 11, ktorý sme naprogramovali my je *24.86101ms*, a z knižnice OpenCV je *1.725239ms*.

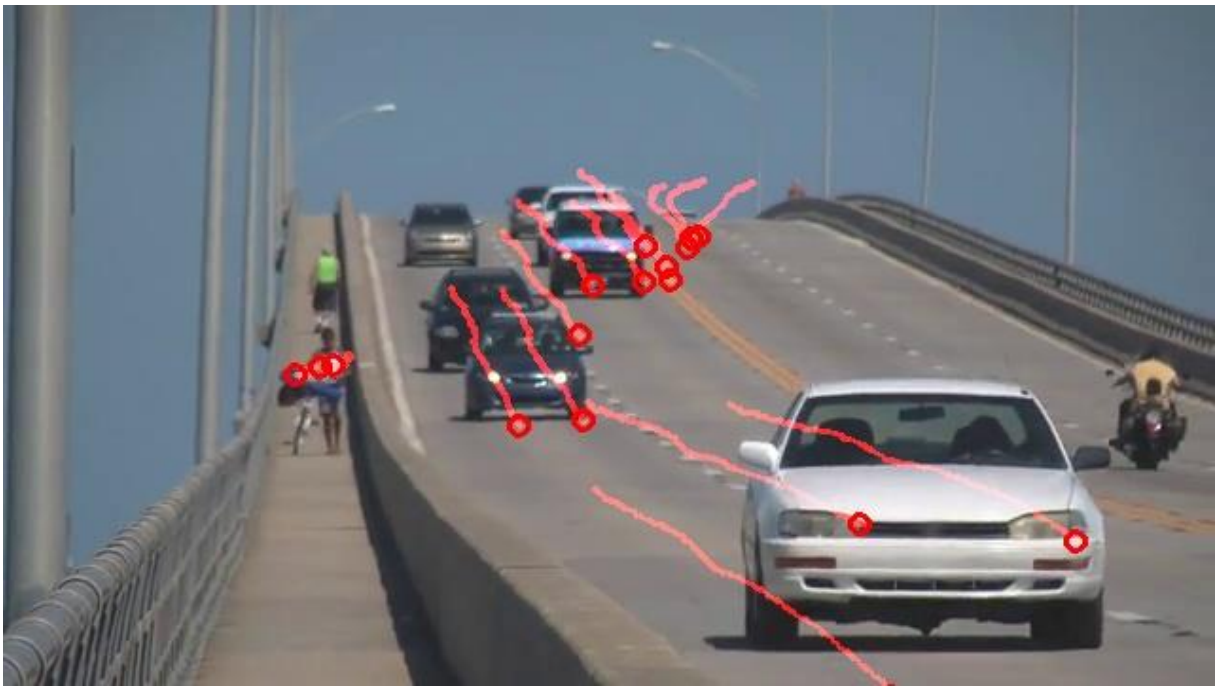
Tabuľka 1 - Vektory medzi frameom 10 a 11

	FRAME 10-11 OpenCV		FRAME 10-11 Vlastná F.		Rozdiel	
	u	v	u	v	u	v
Bod 1	-0,24332	0,672577	-0,37726	1,023867	0,133946	0,35129
Bod 2	-0,02432	0,59201	-0,1519	0,785927	0,127581	0,193916
Bod 3	0,081055	0,614944	-0,15418	0,841241	0,235233	0,226297
Bod 4	0,392303	0,682571	0,562452	0,952298	0,170148	0,269727
Bod 5	-0,35153	0,769989	-,030965	1,017315	0,041881	0,247326
Bod 6	-0,30191	0,662354	-0,33352	0,845948	0,03161	0,183594
Bod 7	-0,11978	0,623062	-0,30562	0,856737	0,185841	0,233675
Bod 8	0,696533	0,525558	1,089864	0,853791	0,393331	0,328232
Bod 9	0,165253	0,472069	-0,23913	0,788538	0,404382	0,31647
Bod 10	-0,37836	0,532234	-0,66838	0,631622	0,290025	0,099387
Bod 11	-0,4823	0,304604	-0,5966	0,672748	0,114305	0,368145
Bod 12	-0,63501	0,219215	-0,83852	0,536469	0,203512	0,317253
Bod 13	-0,32109	0,455566	-0,29195	0,842403	0,029144	0,386837
Bod 14	-0,74643	0,488464	-0,31816	0,676028	0,428266	0,187564
Bod 15	0,550659	0,915497	0,84461	1,135151	0,293951	0,219655
Bod 16	-0,67746	0,361122	-0,34115	0,665146	0,336311	0,304024
Bod 17	-0,23929	0,698936	-0,18469	0,964984	0,054603	0,266048

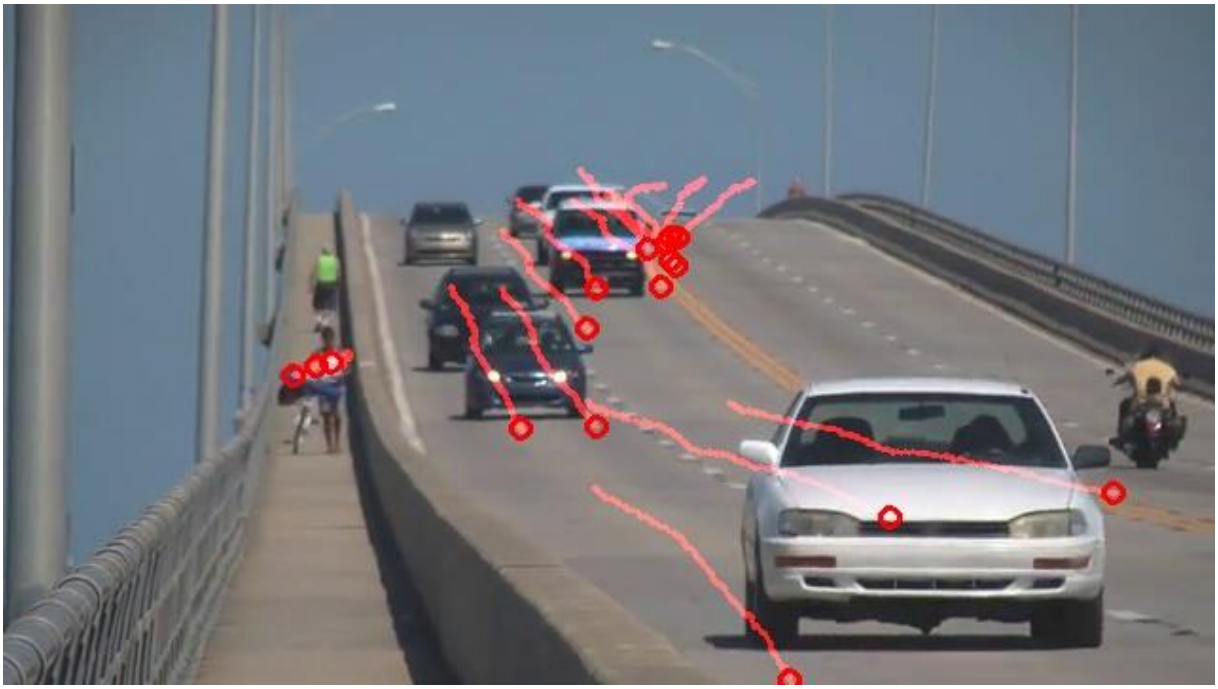
V tabuľke je možné vidieť vektory u, v vypočítané funkciou z knižnice OpenCV a nami naprogramovanou funkciou. Rozdiely týchto vektorov nie sú až také veľké. No čím video ďalej pokračuje zvyšujú sa aj tieto rozdiely.

Na veľkosť týchto vektorov taktiež vplýva ja veľkosť okolia bodu, ktorú si zadáme v našej funkcií. Na nasledujúcich obrázkoch je možné vidieť rozdiely výpočtov vo frame 120.

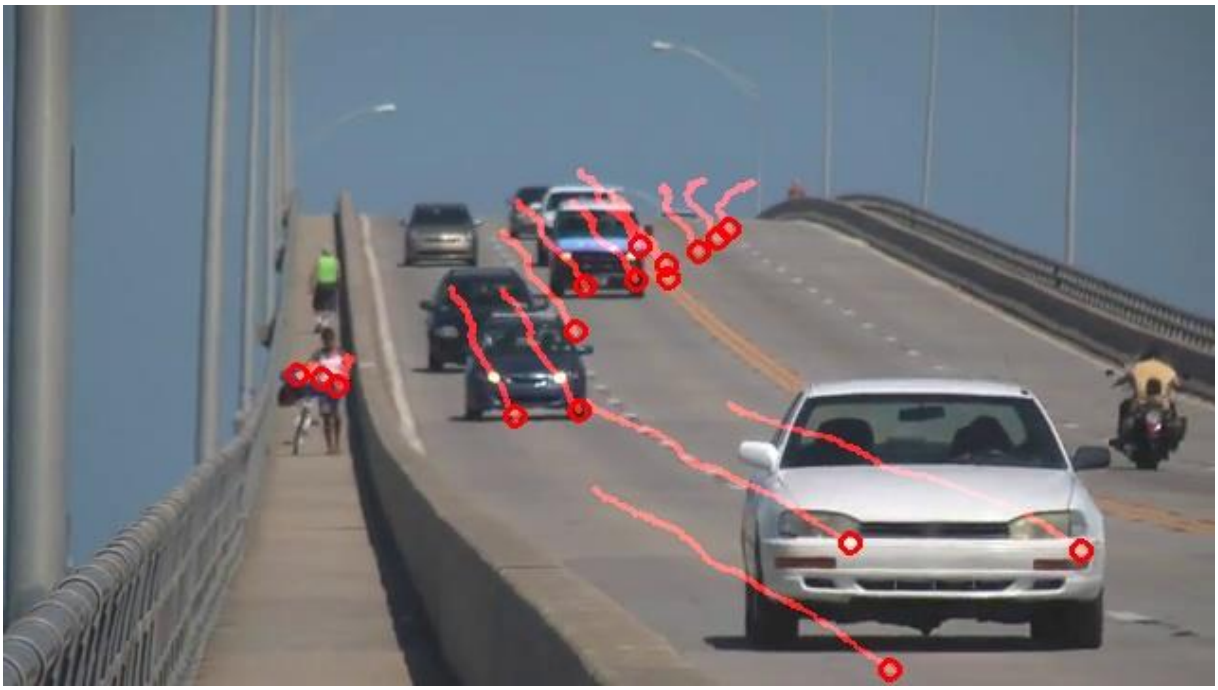
Obrázok 7 - Frame 120 pri veľkosti okolia 9



Obrázok 8 - Frame 120 pri veľkosti okolia 15



Obrázok 9 - Frame 120 pri veľkosti okolia 30



Obrázok 10 - Frame 120 pri veľkosti okolia 50

Obrázok 11 - Frame 120 pri veľkosti okolia 150

Vykonaním experimentov zmenou veľkosti okolia bodu sme zistili, že náš algoritmus nepracuje lepšie pri malej veľkosti okolia bodu, ale ani pri veľmi veľkej veľkosti. Najideálnejšia veľkosť tohto okolia je od 15 do 50px.