Алгоритми за детекција на рабови

Стефан Попов
Универзитет Св. Кирил и Методиј
Факултет за информатички науки и компјутерско инженерство
151165 stefan.popov@students.finki.ukim.mk

Резиме

Раб може да се дефинира како множество од поврзани пиксели кои формираат граница меѓу два дисјунктни региони. Детекцијата на рабови е всушност, метод за сегментација на слика во региони на испрекинатост. Детекцијата на рабови е доста истражувано поле во обработката на дигитални слики и има голема практична примена. Во оваа семинарска работа се претставени повеќе алгоритми за детекција на рабови: Prewitt, Sobel, Canny, Marr-Hildreth, Phase-stretch-transfom и Cellular neural networks. На крај има поглавје посветено на споредбена анализа на приложените алгоритми.

Клучни зборови

детекција на рабови, Prewitt оператор, Sobel оператор, Canny детектор, Laplacian of Gaussian оператор, Marr-Hildreth алгоритам, phase-stretch-transform, клеточни невронски мрежи

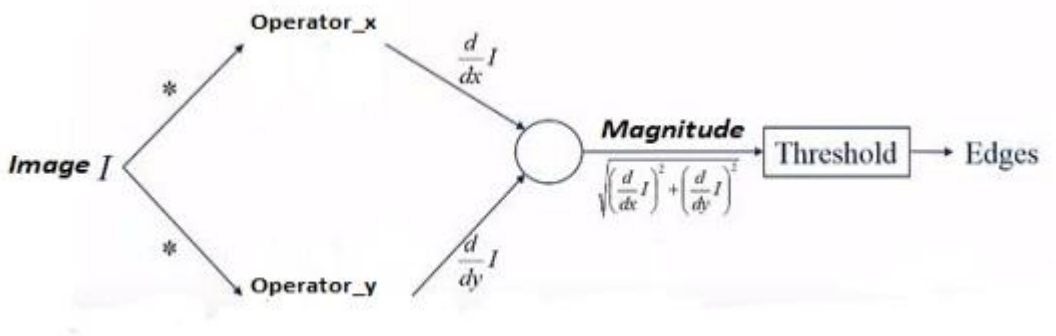
Prewitt, Sobel

Раб во слика е место каде што има нагла промена во (вредноста на функцијата на) интензитетот на сликата. Аналогно на тоа, на местата каде што има рабови, ќе има и екстремни вредности во првиот извод од функцијата на интензитет. Тоа е единствениот, едноставен принцип на кој се основа работата на детекторите од прв ред, меѓу кои се операторите Prewitt и Sobel [1]. Псевдо кодот за детекција на рабови со детектор од прв ред е даден во продолжение.

Алгоритам 1. Општ алгоритам на детектор од прв ред
Влез: **Image**, црно-бела слика
Operator, конкретниот детектор што ќе се користи (Prewitt, Sobel, Roberts...)
Threshold, праг
Излез: Out, црно-бела слика каде пиксел има вредност 255 (бел) ако е раб, и 0 (црн) инаку

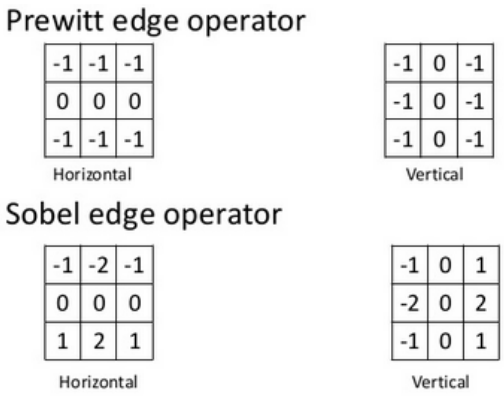
```
1. Filtered_Image = Derivative_filtering(Image, Operator)
2. Out = Thresholding(Filtered_Image, Threshold)
3. return Out
```

Алгоритмот е прилично едноставен и директен: се состои од само две операции. Во првата линија се конволуира влезната слика со влезниот оператор. Постапката на конволуција претставува математичка трансформација на пикселите, што користена во комбинација со детектор од прв ред овозможува апроксимација на првиот извод на сликата во насока на x- и y- оската. Всушност, тоа и се прави овде. Пикселите во сликата ја добиваат апроксимираната вредност на првиот извод. Во следниот чекор тие се разграничуваат: оние што имаат вредност поголема од влезниот праг се дефинираат да бидат рабови, и добиваат максимална вредност (255, бела боја), а сите останати добиваат минимална вредност (0, црна боја) и означуваат отсуство на раб. Поради апроксимацијата на изводот по двете оски, обата операторите има две варијанти соодветно на оската по која апроксимираат. Двете варијанти се користат заедно. Целава постапка графички е илустрирана на Слика 1.



Слика 1. Графичка илустрација на алгоритмот за детекција на рабови со оператори од прв ред. Влезната слика се конволуира во соодветниот правец со x- и y- варијантите на операторите. Добиените вредности по x- и y- оските се квадрираат и од нивниот збир се зема квадратен корен - вредност наречена магнитуда. На крај, на магнитудата ѝ се наметнува праг, за да се разграничи меѓу пиксели што се рабови, и сè останато.

Вредностите на Prewitt и Sobel операторите се следниве:



Слика 2. Prewitt и Sobel оператори.

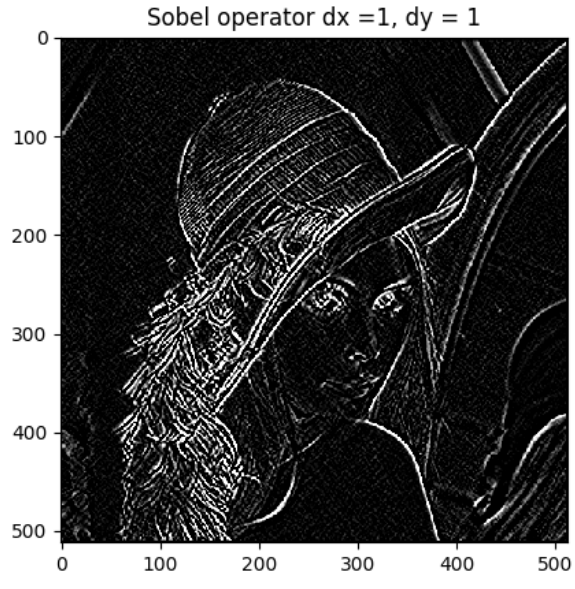
Имплементацијата на операторите во апликацијата е слична со таа во примерите од предавањата. Prewitt операторот е сместен во prewitt.py датотеката и може да се користи на следниов начин (преку терминал со задавање на соодветните аргументи):

```
python prewitt.py <path-to-image> <threshold>
```

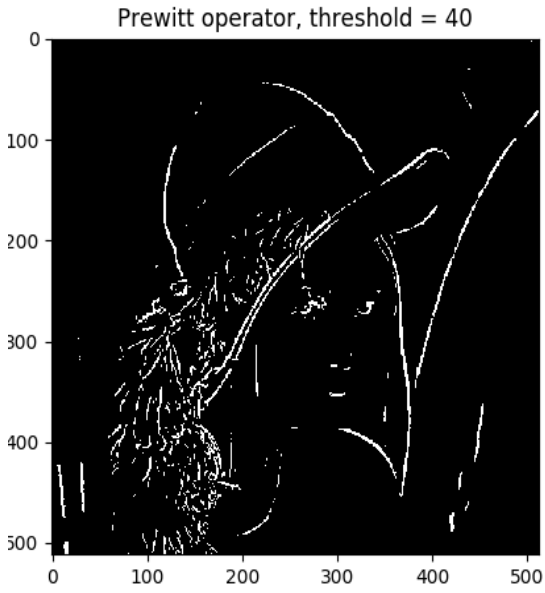
Sobel операторот [3] се наоѓа во sobel.py датотеката и се користи на ист начин, со поголема слобода за варијации:

```
python sobel.py <path-to-image> <dx> <dy>
```

Аргументите `dx` и `dy` може да имаат вредности 0 или 1, со која кажуваат дали да се апроксимира изводот во однос на x- и y- оската соодветно. Зависно од нивната комбинација, се добиваат различни резултати. Комбинација од две нули не е дозволена и фрла соодветен исклучок.



Слика 3. Употреба на Sobel оператор



Слика 4. Употреба на Prewitt оператор

Marr-Hildreth (Laplacian of Gaussian)

Од теорија на калкулус е познато дека во точките каде првиот извод на функцијата има екстремни вредности, вториот извод ќе има нули. Врз овој факт се основа работата на детекторите од втор ред. Еден таков оператор кој го апроксимира вториот извод на функцијата на интензитет на сликата е Лапласовиот. Тој се користи во две варијанти, негативна и позитивна, а ја има следнава вредност:

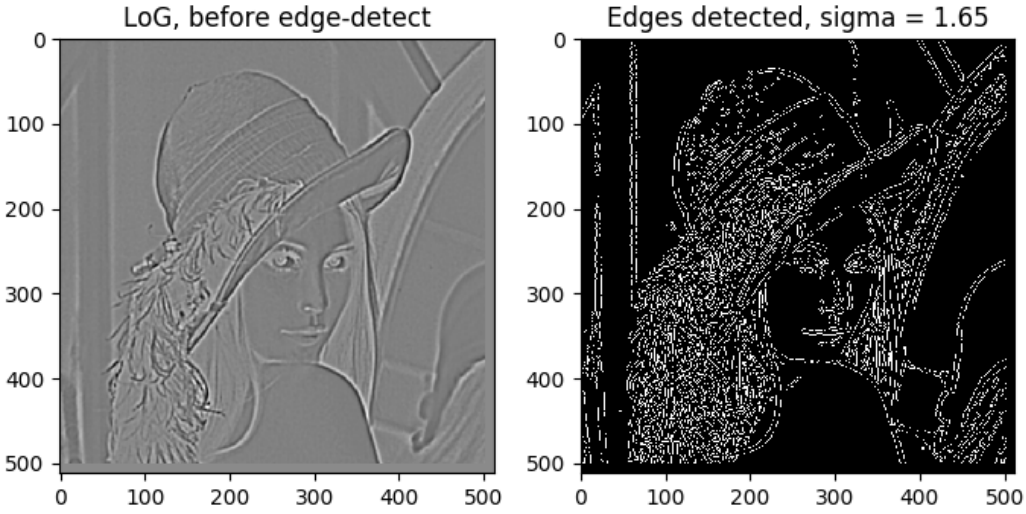
$$\frac{1}{1+a}\begin{bmatrix} a & 1-a & a \\ 1-a & -4 & 1-a \\ a & 1-a & a \end{bmatrix}$$

Сам по себе, Лапласовиот оператор е чувствителен на мали промени и шум, па затоа за подобри резултати се користи во комбинација со Гаусов филтер кој ја измазнува (заматува) сликата. Таа комбинација се нарекува Laplacian of Gaussian и е основата на Marr-Hildreth алгоритмот [4]. Marr-Hildreth прима еден аргумент σ – стандардната девијација на Гаусовиот филтер преку кој се контролира детекцијата. За големо σ алгоритмот ќе детектира поголеми, матни (анг. *blurry*) рабови, а за мало σ ќе детектира помали, фини рабови. Измазнувањето со Гаусовиот филтер и детекцијата на рабови со Лапласовиот оператор во Marr-Hildreth алгоритмот е комбинирано во Laplacian-of-Gaussian (LoG) оператор кој изгледа вака:

$$\nabla^2 G(x,y) = \left[\frac{x^2+y^2-2\sigma^2}{\sigma^4}\right] e^{-\frac{x^2+y^2}{2\sigma^2}}$$

По конвенција, секогаш кога се употребува оваа функција, се користи во негативна варијанта, со знак минус напред. Имплементација на овој алгоритам е сместена во marr-hildreth.py датотеката [14]. Истата може да се користи со командата:

```
python marr-hildreth.py <path-to-image> <sigma>
```



Слика 5. Илустрација на работата на Marr-Hildreth алгоритмот

Canny детектор

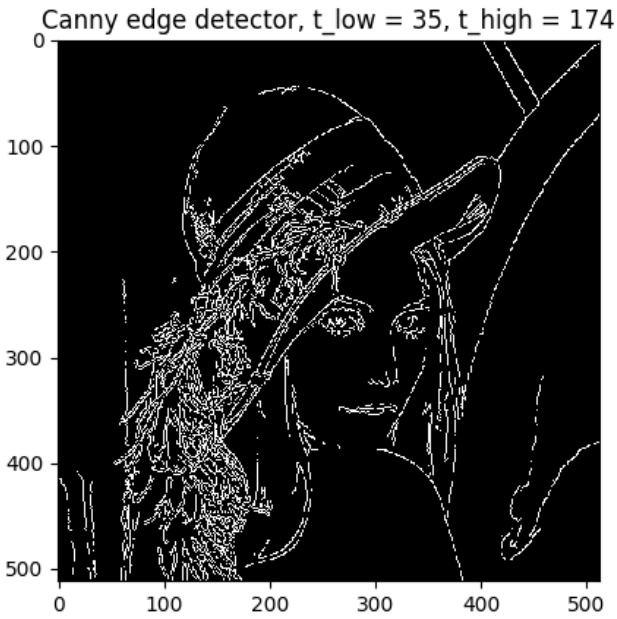
Ова е еден од најкористените алгоритми за детекција на рабови [2]. Работи доста слично како и алгоритмите од прв ред, со мали измени и надополнувања. Canny алгоритмот зема одреден оператор од прв ред и го подобрува, т.е. извршува неколку дополнителни операции со цел да ги зачува рабовите од интерес, а да го отстрани секој несакан шум. Детекцијата на рабови со Canny детекторот се одвива во неколку чекори:

- Измазнување со цел намалување на шум – сликата се конволуира со Гаусов филтер.
- Апроксимација на изводите со некој од операторите од прв ред (најчесто Sobel), има за цел да ги најде најголемите промени во интензитетот на сликата кои одговараат на рабови.
- Истанчување на рабови (non-maximum suppresion) – ги зачувува само локалните максимални апроксимации на изводите а ги брише останатите. Резултатот е истанчен раб, со еднаква ширина насекаде во сликата.
- Бришење на изолирани рабови и спојување на испрекинати делови од раб во еден непрекинат (hysteresis thresholding). Последна операција која ги спојува испрекинатите делови од еден ист раб, а ги брише изолираните испрекинати делови, кои не припаѓаат на поголем раб (шум). Операцијата hysteresis thresholding се применува врз сите пиксели до сега детектирани како рабови. Нека G е магнитуда на еден таков пиксел, според формулата опишана погоре кај детекторите од прв ред. Hysteresis thresholding прима два параметри T_{LOW} и T_{HIGH}, и е дефинирана на следниот начин [1]:
  1. Ако $G < T_{LOW,}$ тогаш пикселот се отстранува (веќе не се смета како раб).
  2. Ако $G > T_{HIGH}$, тогаш пикселот останува (и понатаму се смета како раб).
  3. Ако $T_{LOW} < G < T_{HIGH}$ и барем еден пиксел во 3x3 околина околу тековниот има $G > T_{HIGH}$ тогаш пикселот останува.
  4. Ако нема ниту еден пиксел во 3x3 околина околу тековниот за кој важи $G > T_{HIGH}$, но има барем еден за кој важи $T_{LOW} < G < T_{HIGH}$, тогаш регионот се проширува на 5x5 во кој ќе се бара пиксел за кој ќе важи $G > T_{HIGH}$. Ако се најде барем еден таков, тогаш тековниот пиксел останува означен како раб.
  5. Ако ништо до сега не важи, тогаш пикселот се отстранува.

Последнава операција е најсложениот дел од алгоритмот, но нуди одлични резултати и токму тоа е она што го прави алгоритмот еден од најдобрите и најкористените. Во демо апликацијата детекцијата на рабови со Canny се прави преку командата:

```
python canny.py <path-to-image> <t-low> <t-high>
```

Сликата мора да биде црно-бела, а ако е зададена во друг формат, ќе се претвори во потребниот. Важи логичното ограничување за вредностите на T_{LOW} и T_{HIGH}: $0 < T_{LOW} < T_{HIGH} < 255$.



Слика 6. Илустација на работата на Canny детекторот

Frei-Chen детектор

Методов е варијанта на Sobel операторот, со неколку надополнувања. Покрај хоризонтална и вертикална варијанта, Frei-Chen операторот има оператори кои го апроксимираат изводот и во дијагнонална насока. Тоа се G1, G2, G3 и G4 на Слика 7. Тие се користат за детекција на рабови. G5 – G8 се користат за детекција на линии (линија не е исто што и раб), а G9 се користи за добивање измазнување (*blurring*, *averaging*) [10].

$$G_1 = \frac{1}{2\sqrt{2}}\begin{bmatrix} 1 & \sqrt{2} & 1 \\ 0 & 0 & 0 \\ -1 & -\sqrt{2} & -1 \end{bmatrix} \quad G_2 = \frac{1}{2\sqrt{2}}\begin{bmatrix} 1 & 0 & -1 \\ \sqrt{2} & 0 & -\sqrt{2} \\ 1 & 0 & -1 \end{bmatrix} \quad G_3 = \frac{1}{2\sqrt{2}}\begin{bmatrix} 0 & -1 & \sqrt{2} \\ 1 & 0 & -1 \\ -\sqrt{2} & 1 & 0 \end{bmatrix}$$

$$G_4 = \frac{1}{2\sqrt{2}}\begin{bmatrix} \sqrt{2} & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & -\sqrt{2} \end{bmatrix} \quad G_5 = \frac{1}{2}\begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \quad G_6 = \frac{1}{2}\begin{bmatrix} -1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & -1 \end{bmatrix}$$

$$G_7 = \frac{1}{6}\begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix} \quad G_8 = \frac{1}{6}\begin{bmatrix} -2 & 1 & -2 \\ 1 & 4 & 1 \\ -2 & 1 & -2 \end{bmatrix} \quad G_9 = \frac{1}{3}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

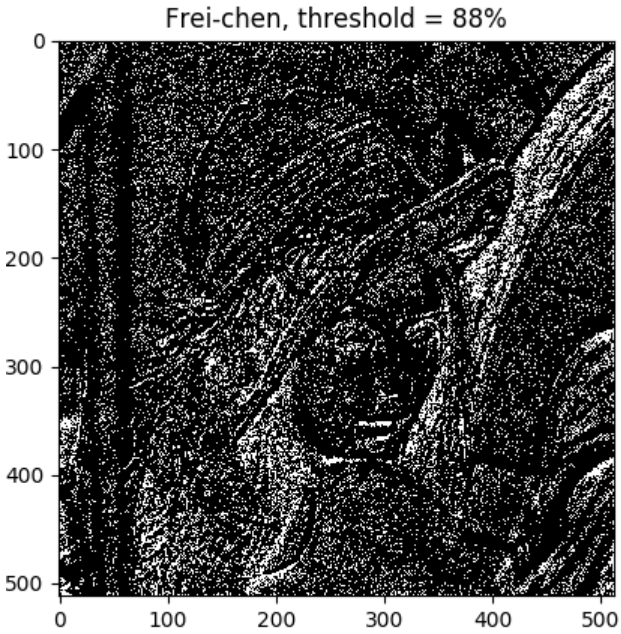
Слика 7. Frei-Chen оператори

Алгоритмот е ист како и оној кај класичните оператори од прв ред, со малку посложена конволуција и математика. Конволуцијата над сликата се прави по следнава формула:

$$\cos e = \sqrt{\frac{M}{S}} \quad where \;\; M = \sum_{k\epsilon\{e\}} (G_k * I)^2 \;\; and \;\; S = \sum_{k=1}^{9} (G_k * I)^2$$

Каде *e* се однесува на множеството оператори за детекција на рабови (G1, G2, G3, G4). Според тоа, сумата за *M* оди до 4, а *cos(e)* е новата вредност на пикселот. Оваа сума го апроксимира изводот по четири оски: x, y, и двете оски дијагонални на нив (со наклон од 45 и

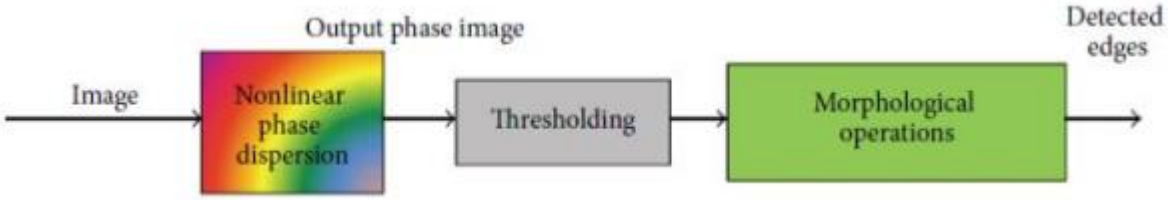
-45 степени, соодветно). Земање косинус од квадратниот корен на изводот, го нормализира истиот а по поставување на праг, се добива финален резултат.



Слика 8. Резултат од Frei-Chen детекторот

Phase-stretch-transform алгоритам

Алгоритмот е прилично нов, развиен е во 2015 година, а инспирацијата за него се добива од набљудуван физички експеримент за детекција на рабови, во реална, печатена слика [5]. Експериментот вклучувал пропагација на светлина со специјални дифракциони (анг. w*arped*) својства низ реална слика. Светлината што поминувала низ сликата откривала премини во интензитетот на сликата, кои потоа може да се третираат како рабови. Софтверскиот алгоритам пробува да го симулира тој експеримент. Илустрација на работата на алгоритмот е прикажана на Слика 9. Прво, оригиналната влезна слика, која треба да биде црно-бела се измазнува со локален филтер, а потоа минува низ нелинеарен фреквентно зависен фазен оператор наречен phase-stretch-transfom или PST, кој применува дво-димензионална фазна функција врз сликата во нејзиниот фреквенциски домен.



Слика 9. Во предложениот метод за детекција на рабови, прво се применува локален филтер за измазнување, а потоа и дифракциона PST операција, по која опционално може да се постави праг на фазата на излезната слика и да се применат некои морфолошки операции, како чекор на претпроцесирање

Големината на фазата зависи директно од фреквенцијата, а бидејќи дигиталните слики имаат поголема фреквенција на рабовите, тие ќе добијат поголема фаза. Во следниот чекор се поставува праг на фазата, по што остануваат само рабовите. На крај алгоритмот применува повеќе морфолошки операции: ерозија, истанчување на рабовите на 1 пиксел ширина, бришење на изолирани, неповрзани пиксели и сл. Поставување на праг е извршување на

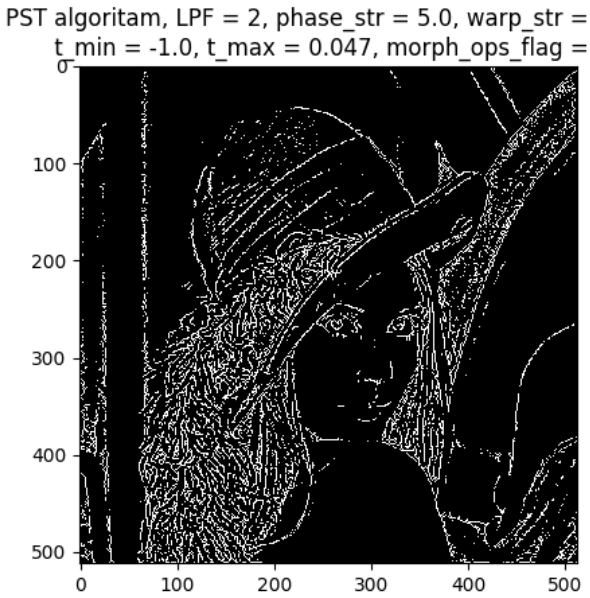
морфолошките операции не е задолжително и може да се изостави. Следува псевдо код за алгоритмот, кој го следи имплементацијата во апликацијата [7].

Алгоритам 2: Phase stretch transform (PST)
Влез: **Image**, црно-бела слика
LPF, scale (1 / дисперзија) за изотропен Гаусов филтер што се користи за измазнување
Phase_strength, фазна јачина на PST операторот
Warp_strength, дифракциона јачина на PST операторот
Threshold_min, праг за негативните вредности на фазата
Threshold_max, праг за позитивните вредности на фазата
Morph_ops_flag, знаме дали да се извршуваат морфолошките операции
Излез: **Out**, црно-бела слика со бели рабови и црна позадина

```
1. Smoothed_image = Low_pass_filtering(Image)
2. PST = Load_PST_Operator(Phase_strength, Warp_strength)
3. Out = Apply_operator(Smothed_Image, PST)
4. if Morph_ops_flag == true then:
   a. Out.removeNoiseEdges()
   b. Out.thinEdges()
   c. Out.bwperim(perim = 4) /* if a pixel has an edge-marked pixel in it's 4 pixel perimeter then mark that pixel an edge too */
   d. Out.thinEdges()
   e. Out.erode()
5. return Out
```

Параметрите на PST операторот, Phase_strength и Warp_strength, директно го контролираат процесот на детекција. Во овој алгоритам постои компромис (trade-off) меѓу резолуцијата во просторниот (spatial) домен на сликата и шумот во детектираните рабови. Поголема фазна јачина (Phase_strength) резултира во подобро справување со шумот во детектираните јазли но дава полоша просторна резолуција. Поголема дифракциона јачина (Warp_strength) детектира појасни рабови но со зголемен шум во истите. Пример и дискусија околу компромисот се дадени во поглавјето за споредби и заклучоци. Како и сите алгоритми во оваа семинарска, и PST е имплементиран во своја датотека. Командата за работа со овој алгоритам е гломазна поради големиот број потребни аргументи, и ја има следнава синтакса:

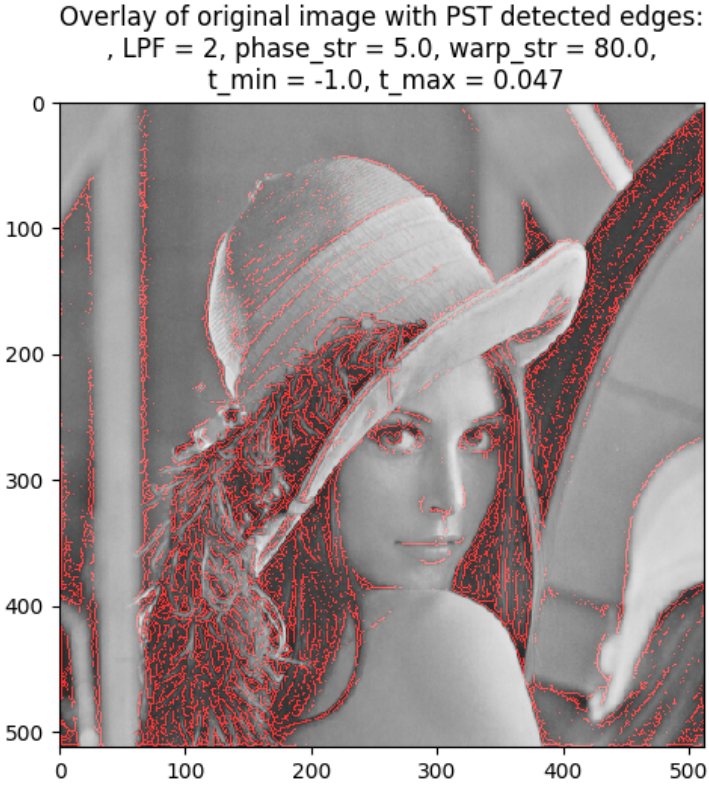
```
python phase-stretch-transform.py <path-to-image> <LPF> <Phase_strength> <Warp_strength> <Threshold_min> <Threshold_max> <Morph_ops_flag>
```



Слика 7. Илустрација на работата на PST алгоритмот

Важно е да се забележи дека за разлика од повеќето имплементирани алгоритми во оваа семинарска, кои зависат само од cv2, numpy и matplotlib библиотеките, овој алгоритам дополнително зависи и од mahotas библиотеката која е потребна за извршување на морфолошките операции. Интересна функционалност е имплементирана во датотеката `pst-overlay.py`. Во неа се прикажува преклопувањето на оригиналната слика со сликата резултат од PST алгоритмот. Таа е добар визуелен показател за работата на алгоритмот, бидејќи може да се оцени дали преклопувањето ги следи рабовите на сликата. Сепак, оваа функционалност може да се користи единствено ако се извршат морфолошките операции врз сликата. Затоа командата за нејзино користење е малку модифицирана во однос на онаа за PST скриптата, бидејќи директно се зема Morph_ops_flag = 1:

```
python pst-overlay.py <path-to-image> <LPF> <Phase_strength> <Warp_strength>
<Threshold_min> <Threshold_max>
```



Слика 8. Илустрација на преклопувањето на детектираните рабови со PST и оригіналната слика

Детекција на рабови со невронски мрежи

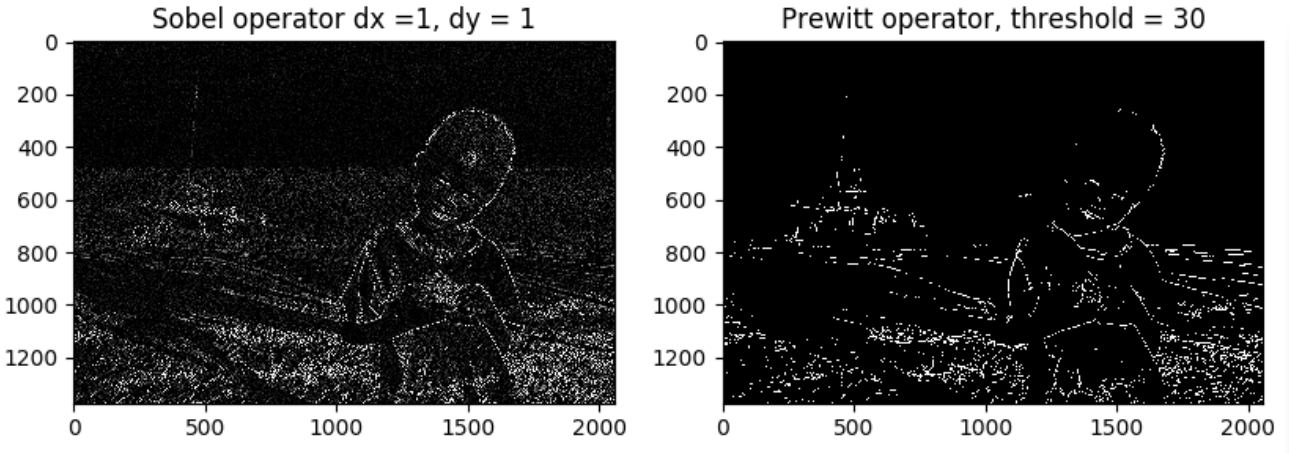
Појавата на многуте различни класични оператори од прв ред, кои во основа вршат иста работа, во комбинација со развојот на невронските мрежи доведе до нов начин на детекција на рабови. Наместо да се користи истиот оператор за детекција кај сите слики, идејата позади невронските мрежи е тие да се користат за да се конструира т.е. научи идеалниот оператор. Гледајќи ги вредностите на операторот како непознати параметри, а третирајќи го прагот (threshold) како наклон (анг. *Bias*) тогаш може да се тренира невронска мрежа да изгради модел на детектор на рабови. Овој начин на детекција станува се поатрактивен и поприменлив во поновата деценија, со развојот на конволуционалните невронски мрежи и развојот на Deep learning проектите во Google. Претставени се неколку алгоритми за детекција на рабови со невронски мрежи [11]. Како што е логично да се претпостави, тие се пресметковно скапи и неизводливи за користење во стандардна секојдневна апликација. Нивната примена (барем до сега) е во суперкомпјутерите кои постигнуваат огромен паралелизам, и секако, се користат за „поголеми“ цели (сателитски снимки, обработка на огромен број слики (поточно Facebook, Google и сл.)). Почнувајќи од 1988 година [12], започнато е истражување на еден нов вид нелинеарно процесирачко коло кое наликува на невронска мрежа и чија примена би била процесирање на високо квалитетно (ultra-speed, high-frame-rate) видео во реално време. Таквото коло е наречено клеточна невронска мрежа (cellular neural network, CNN), а до сега има само неколку компании кои комерцијално го произведуваат [13]. Достапна е имплементација на програма која ја симулира работата на предложената невронска мрежа [8]. Истата е напишана во `cnnClass.py` датотеката, а во програмата се повикува преку командата:

python cnn.py <path-to-source-image> <path-to-destination-image>

Слика 9. Резултат од CNN

Споредби и заклучоци

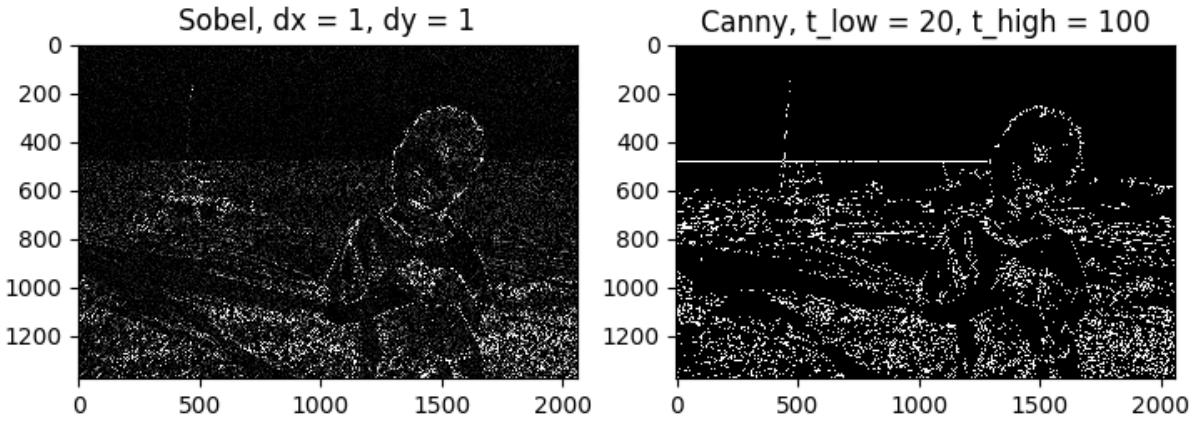
Детекцијата на рабови има многу практични примени. Се користи во обработка на дигитални видеа во реално време, обработка на сателитски слики и снимки, помош при дијагноза кај биомедицински слики, следење и детекција на објекти, препознавање на облици (ракопис, текст, отисоци од прст, говор, лица и сл.). Користењето на правилен алгоритам за детекција е од есенцијално значење за системите кои се соочуваат со овие проблеми. Операторите од прв ред се познати уште од 1980-тите години и се детално проучени. Тие главно се користат во академски цели за изучување на полето на детекција на рабови и ретко каде имаат практична примена. Гледајќи ги само нив, а имајќи го в обзир досегашниот развој на технологијата во поглед на паралелизација, би рекол дека Frei-Chen детекторот е најдобар, иако има значително повеќе пресметки. Негово надградување со non-maximum supression или hysteresis-thresholding би го направило сосема споредлив, ако не и подобар од Canny (сметајќи дека Canny внатрешно користи Sobel оператор). Сепак, нивниот начин на работа веќе е застарен и заменет (или надграден) со некои понапредни техники како невронски мрежи или phase-stretch-transform. Phase-stretch-transfom алгоритмот голема примена наоѓа во биоинформатиката. Неговата способност да ја трансформира сликата во друг домен, и да најде ткн. „скриени рабови“ се покажала исклучително корисна за дијагноза на пациент. Покрај биомедицината, PST алгоритмот е имплементиран и во SAR (synthetic aperture radar) системи [6]. Прикажаната имплементација на клеточните невронски мрежи всушност се користи во процесирање на слика со помош на Raspberry Pi, а штом може да се користи на таков „мини“ уред, тогаш нејзината примена кај помоќни машини е неминовна. Сега следуваат неколку споредбени примери околу работата на имплементираните алгоритми. Сите овие погледи се достапни за генерирање во апликацијата а соодветната команда за нивно повикување е напишана веднаш под сликата, со објаснување за аргументите.



Слика 10. Sobel vs Prewitt

```
python sobel-vs-prewitt.py <path-to-image> <dx> <dy> <threshold>
(python sobel-vs-prewitt.py <sobel-args> <prewitt-args>)
```

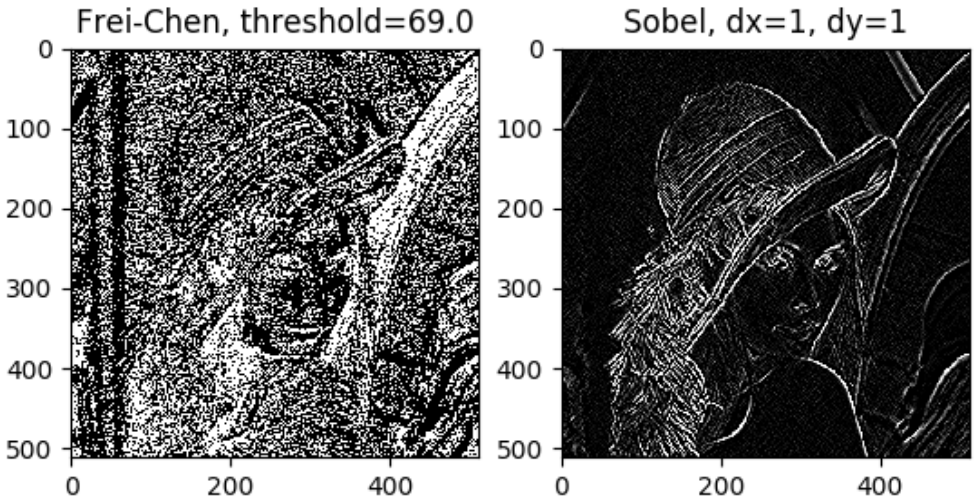
За примеров: `python sobel-vs-prewitt.py 1 1 30`



Слика 11. Sobel vs Canny

```
python sobel-vs-canny.py <path-to-image> <dx> <dy> <t_low> <t_high>
(python sobel-vs-canny.py <sobel-args> <canny-args>)
```

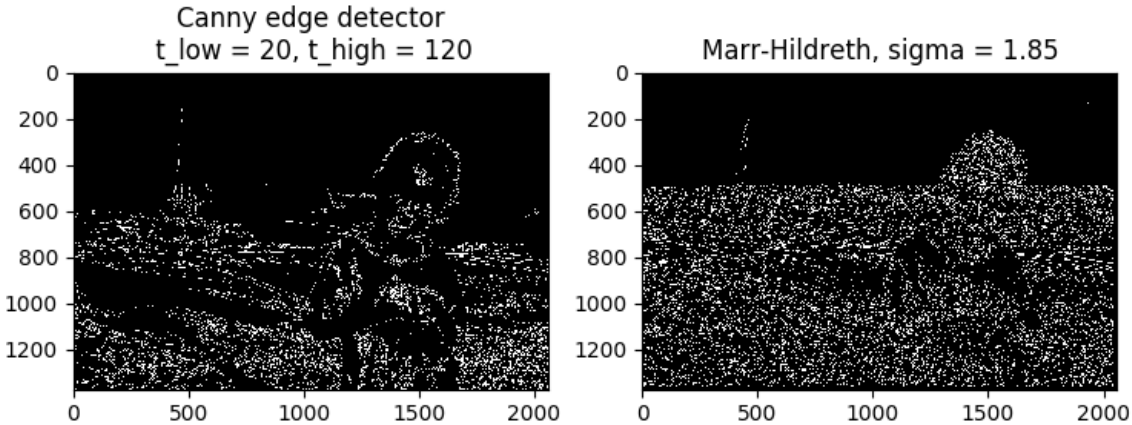
За примеров: `python sobel-vs-canny.py example.jpeg 1 1 20 100`



Слика 12. Frei-Chen vs Sobel

```
python frei_chen-vs-sobel.py <path-to-image> <threshold> <dx> <dy>
```

За примеров: `python frei_chen-vs-sobel.py Lenna.png 69 1 1`



Слика 13. Canny vs Marr-Hildreth

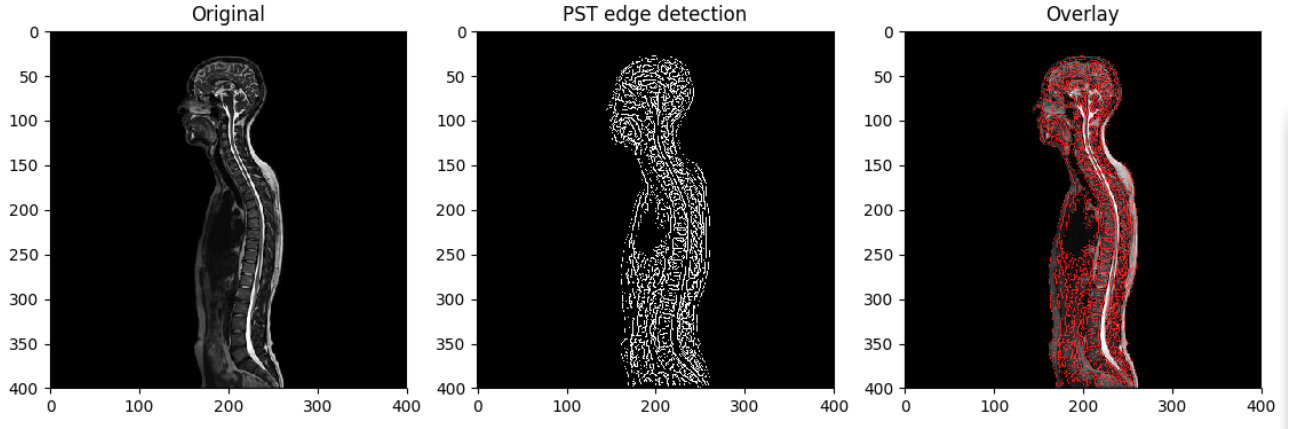
```
python canny-vs-marr_hildreth.py <path-to-image> <t_low> <t_high> <sigma>
```

```
(python canny-vs-marr_hildreth.py <canny-args> <marr-hildreth-args>)
```

За примеров: `python canny-vs-marr_hildreth.py 20 120 1.85`.

Како што може да се забележи, Canny алгоритмот е супериорен над Marr-Hildreth, а и над класичните оператори од прв ред, поради следниве работи:

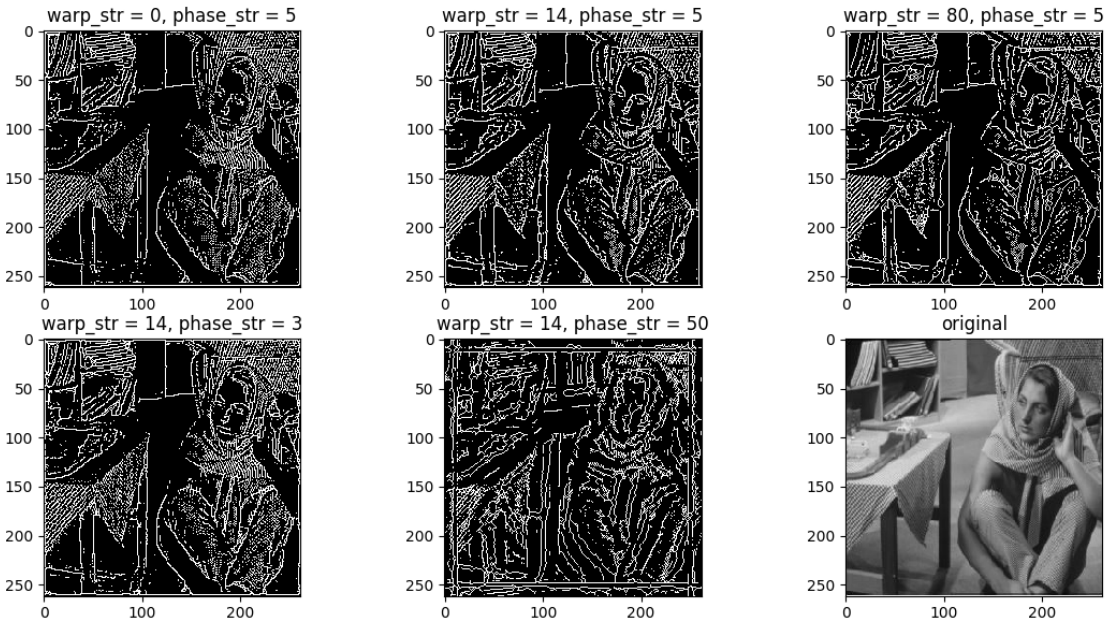
- Користи комбинација од класичен оператор од прв ред со Гаусов филтер за да го намали шумот во сликата.
- Чекорот на двојно разграничување (hysteresis-thresholding) прв пат се појавува кај Canny и го отстранува проблемот на изолирани и неповрзани рабови со кој се соочуваат останатите алгоритми.
- Marr-Hildreth има неколку ограничувања: често детектира лажни рабови, има висока локализациска грешка (не може правилно да ја одреди позицијата на работ) и тешко се справува со криви рабови. Кај Canny тие работи не се среќаваат.



Слика 14. PST and PST ovelay

```
python pst-comparison.py <LPF> <Phase_strength> <Warp_strength> <Threshold_min>
<Threshold_max>
```

Генерира поглед за споредба меѓу оригиналната слика, сликата со детектирани рабови, и преклопувањето меѓу нив.



Слика 15. Влијание на PST аргументите

```
python pst-arg-influence.py
```

Овој пример е целосно „хард кодиран“ т.е. нема никаква интеракција со корисникот и се извршува без аргументи, онака како што е напишан. Зависно директно од патеката до сликата која се обработува како и од рачно запишаните параметри на PST алгоритмот. Служи единствено за илустрација на влијанието на параметрите на PST операторот (Phase_strength, Warp_strength) врз крајниот резултат. Сите слики во првиот ред имаат иста вредност за Phase, а се менува Warp. Таму може да се забележи дека средната слика има најдобри перформанси во поглед на шумот кај детектираните рабови, за разлика од другите две. Сликите во вториот ред и средната слика имаат ист Warp, а се менува Phase. За големо Phase може да се забележи дека е значително намален шумот во рабовите, но нивната резолуција е далеку од добра. Оптималната вредност треба да направи компромис меѓу шумот и резолуцијата, и најчесто се определува со рачно нагодување. Во сите слики се користени исти вредности за останатите параметри и тоа LPF = 2, Thresold_min = -1, Threshold_max = 0.047, Morph_ops_flag = 1.

Референцирани линкови и користена литература

[1] Rashmi, Mukesh Kumar, Rohini Saxena, “Algorithm and technique on various edge detection: A survey”, Signal and Image Processing: An International Journal (SIPIJ) Vol.4 No.3, June 2013

[2] John Canny, “A Computational approach to Edge detection”, IEEE Transactions on pattern analysis and machine intelligence, Vol. PAMI-8, No.6, November 1986

[3] Samta Gupta, Susmita Ghosh Mazumdar, “Sobel edge detection algorithm”, International journal of computer science and management research Vol.2, Issue 2, February 2013, ISSN 2278-733X

[4] David Marr, Ellen Catherine Hildreth, "Theory of edge detection." Proc. R. Soc. Lond. B, 207:187–217, 1980

[5] Mohammad H. Asghari, Bahram Jajali, “Edge detection in digital images using dispersive phase stretch transform”, Hindawi Publishing Corporation, International Journal of Biomedical Imaging, Volume 2015, Article ID 687819, 6 pages

[6] Chistos V. Ilioudis, Carmine Clemente, Mohammad H Asghari, Bahram Jajali and John J. Soraghan, “Edge detection in SAR images using phase stretch transform”

[7] Madhuri Suthar, Ph.D. candidate, Jalali Lab, Department of Electrical and Computer Engineering, UCLA:

https://github.com/JalaliLabUCLA/Image-feature-detection-using-Phase-Stretch-Transform

(Имплементацијата на Phase stretch transform)

[8] CNN Image Processing library:

https://github.com/ankitaggarwal011/PyCNN

(Имплементацијата на CNN)

[9] Презентациите од предавањата на предметот

[10] A technical blog by Daniel Racos, posted on January 30, 2011:

http://rastergrid.com/blog/2011/01/frei-chen-edge-detector/

[11] Joon K. Paik, Aggelos K. Katsaggelos, „Edge detection using a neural network“, Conference paper in Acoustics, Speech and Signal Processing, 1988

[12] L.O. Chua, L. Yang, „Cellular neural network: Theory“, IEEE Transactions on Curcuits and Systems (Volume: 35, Issue: 10, Oct 1988)

[13] Текст за комерцијалната достапност на CNN процесори:

https://en.wikipedia.org/wiki/Cellular_neural_network#Technology

[14] Marr-Hildreth:

https://github.com/adl1995/edge-detectors