Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Domen Kren

Prepoznavanje krožnic na digitalnih slikah

diplomsko delo

univerzitetni interdisciplinarni študijski program prve stopnje računalništvo in matematika

Mentor: izr. prof. dr. Gašper Jaklič

Ljubljana 2014

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

V diplomskem delu obravnavajte problem prepoznavanja krožnic na digitalnih slikah. Osredotočite se na študij geometrijskih in algebraičnih metod, ki za dane točke poiščejo središče in radij krožnice, ki se le-tem najbolje prilega. Primerjajte njihovo hitrost in učinkovitost.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Domen Kren, z vpisno številko **63110231**, sem avtor diplomskega dela z naslovom:

Prepoznavanje krožnic na digitalnih slikah

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom izr. prof. dr. Gašperja Jakliča,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 30. avgust 2014 Podpis avtorja:

Na tem mestu bi se najprej zahvalil mentorju, izr. prof. dr. Gašperju Jakliču, za vodenje in pomoč pri izdelavi diplomskega dela. Zahvaljujem se tudi staršem, za vso potrpežljivost in podporo v času študija.

Kazalo

Povzetek

Abstract

1 Uvod **1**

2 Računalniški vid **5**

2.1 Pridobitev slike 6

2.2 Predprocesiranje 6

2.3 Črpanje objektov 6

2.4 Detekcija/segmentacija 8

2.5 Visokonivojsko procesiranje 9

2.6 Odločanje 10

3 Teorija iskanja krožnic **11**

3.1 Izbira metode 11

3.2 Metoda najmanjših kvadratov 13

3.3 Obstoj rešitve 14

3.4 Enoličnost rešitve 14

3.5 Izbira parametrizacije 15

4 Geometrijske metode **17**

4.1 Minimizacija 18

4.2 Levenberg-Marquardt 21

4.3 Chernov-Lesort 22

5 Algebraične metode **25**

5.1 Kasa 26

5.2 Pratt 27

5.3 Taubin 29

6 Testi in analiza **33**

6.1 Robustnost 33

6.2 Natančnost 35

6.3 Hitrost 38

7 Zaključek **41**

8 Priloga: Algoritmi v Matlabu **43**

Seznam uporabljenih kratic

kratica	angleško	slovensko
LS	least-squares	najmanjši kvadrati
DLS	damped least-squares	dušeni najmanjši kvadrati
RANSAC	random sample consensus	konsenz naključnih vzorcev
LM	Levenberg-Marquardt	Levenberg-Marquardt
CL	Chernov-Lesort	Chernov-Lesort

Povzetek

Prepoznavanje krožnic na digitalnih slikah je eden od problemov, ki se pojavljajo pri zaznavanju objektov pri procesiranju raznih slik/videoposnetkov.

Rešitve se uporablja v mnogih znanstvenih postopkih, kot so analize rentgenskih slik, detekcija oči na raznih medijih, sledenje objektov v vesolju,... Seveda uporabljamo detekcijo krožnic tudi v druge namene, kot so prepoznavanje krožnih objektov na raznih videoposnetkih, robotski vid in podobne.

Postopek prepoznavanja poteka od pridobitve slike do končnih rezultatov. Med vmesnimi koraki so še priprava slike, črpanje objektov, razne detekcije, visokonivojsko procesiranje in druge. Sama diplomska naloga se osredotoča na algoritme, ki nam omogočajo prepoznavanje krožnic iz podanih točk, večinoma po metodi najmanjših kvadratov. Predstavljenih je nekaj najpogostejših pristopov, med katerimi iščemo najbolj prilagodljive, robustne, hitre in preproste rešitve. Vsi algoritmi so analizirani in stestirani ter podrobno obrazloženi s primeri kode.

S podrobno analizo in predstavitvijo pristopov omogočamo vnos izbrane rešitve v poljubno aplikacijo. Na podlagi predstavljenih testov se lahko odločimo za algoritem, ki je najbolj primeren in optimiziran za naš program.

Ključne besede: algoritem, zazanavanje, krožnica, analiza, najmanjši kvadrati.

Abstract

Identifying circles on digital images is one of the problems that occur in the perception of objects when processing different images/videos.

Solutions are used in many scientific procedures, such as analysis of X-ray images, eye detection, tracking objects in space,... Of course, we use detection of circles also for other purposes, such as recognition of circular objects in various videos, robot vision and the like.

Identification process takes place from image acquisition to final results that we have obtained with the program. The steps in-between are preparing pictures, drawing objects, various object detection, high-level processing and others. This thesis focuses on algorithms that allow us to fit circles based on specified points by using least-squares method. We present some of the most common approaches, among which we are looking for the most flexible, resilient, quick and simple solutions. All algorithms are analysed, tested and explained with code examples.

With detailed analysis and presentation we enable the entry of the selected solution in any application. Based on the presented tests we can choose an algorithm that is best suited and optimized for our program.

Keywords: algorithm, detection, circle, analysis, least squares.

Poglavje 1

Uvod

Prepoznavanje krožnic na digitalnih slikah, oziroma bolj splošno, prepoznavanje objektov na podanih medijih je eden od problemov, ki jih velikokrat srečamo v modernem računalništvu. Pojavi se tako pri analizah podatkov v mnogih znanostih, kot v zaznavanju krožnih predmetov na videoposnetkih, recimo za potrebe športa, računalniškega vida, ...

Splošen postopek obdelave digitalnega medija [7] je sestavljen iz kar nekaj delov in poteka takole:

- Pridobitev slike - pridobimo jo iz raznih slikovnih senzorjev, ki določajo tip in ločljivost slike
- Predprocesiranje - sliko pripravimo za nadaljno uporabo v naše namene, na primer premaknemo koordinatni sistem slike, poudarimo kontraste, ki nas zanimajo, zmanjšamo šume, spremenimo skalo slike, ...
- Črpanje objektov - pridobimo razne objekte iz slike, kot so robovi, linije, grebeni, koti, vnaprej določene točke, ...
- Detekcija/segmentacija - odločimo se, katere točke, deli slike nas zanimajo in jih pripravimo za nadaljnje procesiranje
- Visokonivojsko procesiranje - iz izbranih točk želimo prepoznati razne oblike, potrditi veljavnost prepoznanih objektov, razdeliti objekte v kategorije, ...

- Odločanje - končna odločitev za aplikacijo(smo/nismo prepoznali oblike, ocenimo kvaliteto slike, rezultate pošljemo za nadaljno uporabo, ...)

V drugem poglavju je vsak del postopka bolj podrobno obrazložen in predstavljen s primeri.

V tem diplomskem delu se bomo osredotočili na zaznavo objektov, bolj natančno, krožnic, na digitalnih slikah. Problema zaznavanja se lahko lotimo na več načinov, a skoraj vsem so skupni algoritmi, ki izbranim točkam dodelijo krožnico. Cilj tega diplomskega dela je predstavitev, analiza in testiranje najbolj pogostih in uporabnih metod za prireditev krožnice dani množici točk.

V naslednjem poglavju bomo nadaljevali s predstavitvijo teorije za kasneje razloženimi in uporabljenimi metodami. Najprej bomo razložili kako se matematično težave lotimo in predstavili možne izbire parametrizacije ter metodo najmanjših kvadratov, ki se pojavlja v skoraj vseh uporabljenih algoritmih. Povedali bomo tudi nekaj o minimizaciji, ki jo potrebujemo za čim bolj natančno iskanje prirejenih krožnic, ter nekaj o obstoju in enoličnosti rešitve, ki jo pridobimo z različnimi metodami.

Za osnovno teorijo bomo pričeli v naslednjem poglavju kar s predstavitvijo prvih metod. Najprej bodo predstavljene geometrijske metode. Osnovna lastnost le teh je iskanje rešitve z minimizacijo ortogonalnih razdalj od točk do krivulje, po kateri s pomočjo iteracije pridemo do poljubno natančne rešitve. Predstavljena bo najbolj znana in popularna Levenberg-Marquardt metoda ali dušeni najmanjši kvadrati in njena izboljšava z drugačno parametrizacijo za še večjo stabilnost. Obe metodi bosta predstavljeni s primeroma kode in sicer se za potrebe tega dela uporablja jezik Matlab.

V naslednjem poglavju bodo sledile tri algebraične metode. Tem je skupno, da se pri prirejanju namesto ortogonalne razdalje minimizira malo prirejena funkcija, predstavljena s preprosto algebraično formulo, ki je veliko lažje izračunljiva. Predstavljene bodo metode Kasa, Pratt in Taubin, spet s pripadajočimi realizacijami v Matlab-u. V vseh se pojavi problem numeričnega reševanja predoločenih sistemov in iskanja lastnih vrednosti, tako da je nekaj besed namenjenih tudi temu.

Nadaljujemo s šestim poglavjem, testi in analiza. Vsi algoritmi bodo analizirani in testirani pod najrazličnejšimi pogoji. Zanima nas robustnost algoritmov. Kako občutljivi so na majhne in velike napake? Kako močno razne napake pri zaznava-

nju vplivajo na končno krožnico? Nam napake na nekaj točkah močno premaknejo rešitev ali pa se jih sploh ne opazi? Zraven spadajo tudi pogoji pod katerimi se algoritem ne izvede, oziroma rešitev ni uporabna. Sledi natančnost algoritmov. Pojem natančne rešitve ni popolnoma definiran, tako da bomo rešitve algoritmov medsebojno primerjali in analizirali, kako se obnašajo nad najrazličnejšimi množicami točk. Za konec pa še nekaj o hitrosti algoritmov. Pogledali si bomo teoretično in dejansko hitrost izvajanje metod in seveda primerjave med uporabljenimi algoritmi. S predstavljenimi rezultati bo bralec dobil predstavo, pod kakšnimi pogoji naj se določene algoritme uporablja.

Za zaključek bo sledil še kratek komentar in hitra ocena predstavljenih algoritmov ter nasveti, kdaj in v kakšnih primerih so le-ti najbolj uporabni.

Poglavje 2

Računalniški vid

Računalniški vid je področje, ki vključuje metode za pridobivanje, procesiranje in analiziranje. To počnemo z namenom pridobivanja informacij, ki jih uporabimo za odločanje pri problemih, ki nas zanimajo. Glavne naloge so prepoznavanje objektov, identifikacija, detekcija, analiza premikanja, strukturiranje tridimenzionalnih prostorov na podlagi slik in odpravljanje napak, šumov na le teh [2].

Računalniški vid in postopek detekcije, povezan z njim, lahko uporabljamo v veliko namenov. Nekaj primerov je kontroliranje procesov, industrijski roboti, navigacija, detekcija dogodkov, organizacija informacij, modeliranje, interakcije, avtomatizirane inšpekcije in podobno. Eden najbolj vidnih in pomembnih je uporaba v medicinske namene, kjer s pomočjo detekcije pomagamo pri analizi, s pregledom mikroskopskih, ultrazvočnih, rentgenskih in drugih diagnostičnih slik.

Seveda pa so postopki in metode pri računalniškem vidu, ki ga uporabljamo, popolnoma odvisne od aplikacije. Določeni sistemi imajo zelo specifične in samozadostne funkcije, ki so recimo povezane z zaznavo ali meritvami, medtem ko obstajajo drugi z velikim številom podsistemov za planiranje, informacijske baze, človeške interakcije in podobno. Implementacija metode za računalniški vid je odvisna tudi od tega ali sta postopek in cilj vnaprej definirana, ali moramo vse skupaj načrtovati s prostorom za učenje in spreminjanje samega algoritma.

Obstaja pa nekaj osnovnih funkcij [7], ki so skupne skoraj vsem in so bolj natančno predstavljene v naslednjih podpoglavjih. S pregledom le teh dobimo tudi razumevanje in boljši vpogled v samo delovanje sistemov, namenjenih računalniškemu vidu.

2.1 Pridobitev slike

Postopek pričnemo s pridobitvijo medija. Najbolj pogost primer je preprosta slika, ki jo naredimo preko raznih senzorjev. Merimo lahko jakost svetlobe, valovne dolžine, temperaturne razlike, zvok v slišnem in supersoničnem spektru, radiacijo, ... Glede na tip senzorja lahko dobimo dvodimenzionalno sliko ali tridimenzionalni prostor.

Nadaljne algoritme ponavadi prilagajamo glede na tip in zapis slik, ki jih želimo obravnavati, če pa algoritem že obstaja, moramo čim bolje prilagoditi medije, ki jih posredujemo. Pomembno je da so slike/videoposnetki primerni za željeno analizo. Za podrobne raziskave morajo biti slike visokokakovostne, medtem ko za živo analizo videoposnetkov to ni tako pomembno. Algoritme in metode moramo tudi prilagajati številu slik, ki jih želimo procesirati. Za posamezne in redke analize so lahko algoritmi bolj potratni, za procesiranje nekaj 100 slik na sekundo(analiza videoposnetkov) pa potrebujemo bolj optimizirane in osredotočene algoritme, da je vse skupaj sploh možno izvajati na sistemih, ki so nam na voljo.

2.2 Predprocesiranje

Po pridobitvi slike/medija, ki ga želimo obravnavati, ga želimo pripraviti točno za naš algoritem. S predprocesiranjem odpravimo napake na samem mediju, ki bi kasneje lahko preprečile ali otežile izvajanje funkcij na sliki.

Nekaj primerov je zmanjševanje šumov na sliki, da kasneje ne dobimo napačnih rezultatov, poudarjanje kontrastov in sprememb za lažje kasnejše zaznavanje, ponovno vzorčenje za potrditev koordinatnega sistema slike, uporaba lokalne skale reprezentacije za bolj primerno vzorčenje in mnoge druge.

2.3 Črpanje objektov

Nadaljujemo z črpanjem objektov. Objekt ni standardno določen pojem in je pogosto vezan na problem, ki ga rešujemo z aplikacijo. Metodo črpanja objektov tretiramo kot nizkonivojsko operacijo pri procesiranju slik, saj je ponavadi prva stvar, ki jo naredimo. Za vsak pixel na sliki preverimo, če je del objekta. V pri-

meru, da je črpanje del večjega algoritma, se ponavadi osredotočimo le na določene dele in tipe objektov v sliki. Glavni tipi, ki jih zaznavamo, so:

- Robovi - robovi so točke na prepreki med dvema regijama
- Koti - iščemo točke na robovih, ki nenadno spremenijo smer (imajo obliko kota)
- Področja - zaznavamo regije točk s podobnimi lastnostmi

Za vsakega od tipov objektov obstaja kar nekaj algoritmov za detekcijo. Kot primer je predstavljen Sobelov operator [2] za iskanje robov.

2.3.1 Sobelov operator

Sobelov operator naredi oceno gradienta za vsako točko na sliki. S tem postopkom pridobimo novo sliko gradientov, kjer so poudarjene robne točke. Za pridobitev uporabimo 2D diskretno konvolucijo prvotne slike z dvema 3×3 matrikama, ki ju imenujemo jedri.

Najprej si bomo pogledali kaj je diskretna konvolucija, saj je zelo pomembna pri predstavljenem postopku. Vzamemo dve matriki,

$$P = \begin{bmatrix} P_{11} & \cdots & P_{1M} \\ \vdots & \ddots & \vdots \\ P_{1N} & \cdots & P_{NM} \end{bmatrix} \quad \text{in} \quad K = \begin{bmatrix} K_{11} & \cdots & K_{1m} \\ \vdots & \ddots & \vdots \\ K_{1n} & \cdots & K_{nm} \end{bmatrix},$$

kjer je P neka $N \times M$ matrika, K pa naše $n \times m$ jedro (kernel). Rezultat konvolucije $P * K$ je matrika

$$O = \begin{bmatrix} O_{1,1} & \cdots & O_{1,M-m+1} \\ \vdots & \ddots & \vdots \\ O_{1,N-n+1} & \cdots & O_{N-n+1,M-m+1} \end{bmatrix},$$

kjer je

$$O(i,j) = \sum_{k=1}^{m} \sum_{l=1}^{n} P(i+k-1, j+l-1) K(k,l)$$

in i teče od 1 do $M-m+1$ ter j teče od 1 do $N-n+1$. Nadaljujemo s predstavitvijo Sobelovega operatorja.

Kot vhod algoritma ponavadi vzamemo sivinsko sliko, ki si jo lahko predstavljamo kot $n \times m$ matriko I (2.1), kjer je vsak piksel slike predstavljen z $I_{nm} \in \mathbb{N}$, ki ponazarja jakost barve,

$$I = \begin{bmatrix} I_{11} & \cdots & I_{1m} \\ \vdots & \ddots & \vdots \\ I_{1n} & \cdots & I_{nm} \end{bmatrix}. \tag{2.1}$$

Sedaj uporabimo dve jedri, ki sta zasnovani, da se maksimalno odzoveta, prva na horizontalne in druga na vertikalne robove. Izračunamo

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * I \qquad \text{in} \qquad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * I,$$

kjer sta G_x in G_y matriki gradientnih aproksimacij za prvotno sliko I in * dvodimenzionalna konvolucijska operacija. Obe aproksimaciji za posamezno točko lahko združimo z

$$G = \sqrt{G_x^2 + G_y^2}$$

ter tako dobimo matriko gradientnih magnitud G za sliko I. Kotno orientacijo robu izračunamo iz

$$\Theta = \arctan(G_y / G_x),$$

kjer v našem primeru orientacija 0 pomeni da smer maksimalnega kontrasta od črne proti beli kaže od leve proti desni. Ostali koti so od tega merjeni v nasprotni smeri urinega kazalca.

Na sliki 2.1 vidimo končni rezultat uporabe Sobelovega operatorja. Najdene so robne točke, ki jih lahko uporabimo za nadaljno procesiranje.

2.4 Detekcija/segmentacija

Odločimo se, kateri deli, objekti, regije slike nas zanimajo za nadaljno procesiranje. Dobri algoritmi za pametno izbiro podatkov so zelo pomembni pri optimizaciji obsežnejših metod.

Slika 2.1: Primer uporabe Sobelovega operatorja za detekcijo robov na digitalni fotografiji

2.5 Visokonivojsko procesiranje

V tem delu ponavadi obravnavamo le manjšo, izbrano množico točk ali regijo [7]. Za to množico potem s primernimi algoritmi testiramo in ji dodajamo lastnosti, ki ji pripadajo.

Lahko na primer preverjamo ali podatki zadoščajo predpostavkam, ki smo jih določili za ta model. Lahko tudi ocenjujemo parametre oblik, ki jih ti podatki predstavljajo, na primer krožnice, elipse ali bolj kompleksne oblike. Izvajamo lahko tudi prepoznavanje in zlivanje slik, najbolj pogosti funkciji, ki se ju uporablja v računalniškem vidu.

Pri prepoznavanju nas zanima, katera oblika in kje se nahaja na sliki. Algoritmi za te vrste prepoznavanja so zelo specifični in nimamo splošne rešitve za zaznavanje poljubnih oblik. Je pa postopek zelo uporaben, saj ga lahko uporabljamo za najrazličnejše namene [9] kot so prepoznavanje obrazov na slikah, splošno iskanje objektov na videoposnetkih, prepoznava besedila na slikah, razni postopki identifikacije s prstnimi odtisi in mnoge druge.

Uporabno pa je tudi zlivanje slik. To vzamemo pri združitvi večih slik v eno, oziroma pri kakršnemkoli združevanju medijev, ki se nanašajo na nek skupen opazovan objekt. Pri zlivanju se pojavi kar nekaj algoritmov, ki jih lahko uporabimo

v razne namene. Lahko povežemo/združimo več slik v eno samo, s prepoznavanjem skupnih točk in združevanjem je mogoče narediti ogromne slike z majhnim fotoaparatom (panoramska slika vidnega vesolja). Uporabni so tudi v zdravstvene namene [1]. Z ustreznimi algoritmi lahko primerjamo na primer slike, narejene z magnetno resonanco. Vzamemo posnetka istega človeka, ju računalniško poravnamo in poudarimo vse spremembe, ki so se zgodile med slikanjema ter tako pomagamo pri diagnozi.

V tem delu sistema naj bi stestirali, analizirali in preverili tisto, kar nas zanima. Ko pridobimo vse možne podatke nam ostane le še odločanje in uporaba rezultatov.

2.6 Odločanje

Na koncu se le še odločimo, kaj hočemo z našimi ugotovitvami narediti za aplikacijo. Primer je avtomatsko potrjevanje zaznanih objektov na sliki, recimo prstni odtisi. Lahko tudi vrnemo rezultate prepoznavanj različnih oblik, smo ali nismo zaznali obraza. V bolj zapletenih in pomembnih primerih lahko rešitev tudi pošljemo za nadaljni človeški pregled, kar počnemo v medicinske, varnostne in prepoznavne namene.

S tem je naš sistem računalniškega vida zaključen in pripravljen za uporabo.

Poglavje 3

Teorija iskanja krožnic

Po kratkem pregledu splošne teorije računalniškega vida se bomo sedaj posvetili specifičnemu problemu iskanja krožnic. Ker krožnice spadajo med primitivne oblike, ki se velikokrat pojavljajo na slikah in v raznih raziskavah, želimo imeti zanesljive, hitre in natančne algoritme za njihovo zaznavo. Zaradi zgornjih razlogov je bilo vloženo veliko dela v to smer in imamo sedaj na razpolago mnogo najrazličnejših metod, ki nam omogočajo zaznavo krožnic skoraj v vsakem primeru, ki si ga lahko zamislimo.

Med najbolj znanimi tipi rešitev so razne metode najmanjših kvadratov, Houghove transformacije, RANSAC metode,... V naslednjem podpoglavju bomo na kratko predstavili glavne vrste iskanj, s katerimi se lahko srečamo in kakšne metode naj v teh primerih uporabimo. Kasneje se bomo usmerili v analizo in testiranje metod, narejenih na podlagi najmanjših kvadratov.

3.1 Izbira metode

Zaznavanja krožnic se lahko lotimo na mnogo načinov. Izbira pravega je odvisna od problema, ki ga želimo rešiti. Načeloma pri zaznavanju prej določenih objektov, glede na podatke, ki jih dobimo, delimo algoritme na tri glavne tipe zaznavanja.

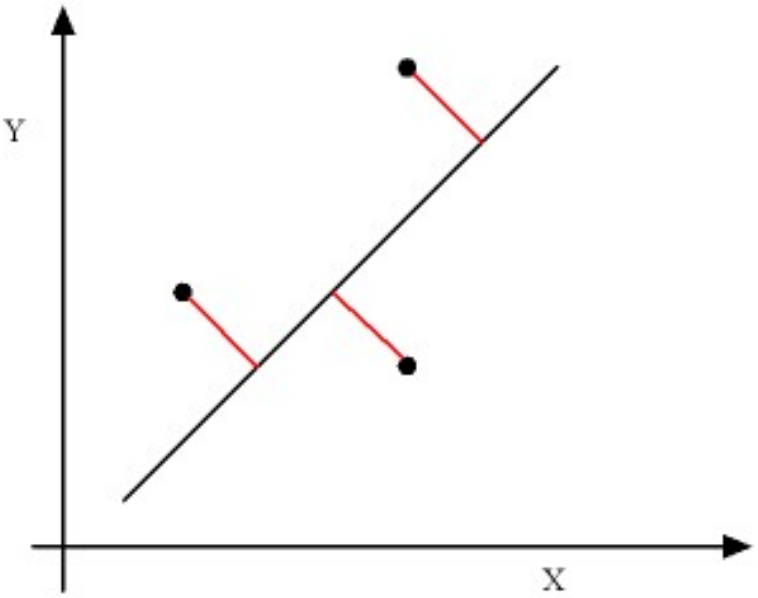
V prvem primeru vemo, katere točke pripadajo iskani krožnici in nas zanimajo parametri le-te. Pri tem načinu uporabimo metode, ki podanim točkam priredijo najboljšo krožnico. Najbolj pogosti metodi za to vrsto sta metoda najmanjših kvadratov - LSF (ang. Least squares fitting), na katero smo se osredotočili pri

tej diplomski nalogi, in razna verjetnostna prirejanja. Glavna ideja teh metod je minimizacija razdalj med točkami in krožnico. Več o tem sledi v naslednjem poglavju. Seveda so te metode najbolj uporabne pri "čistih"podatkih, torej s čim manj šuma in zunanjih točk, ki niso del objekta, ki ga želimo zaznati.

Drugi primer je, ko imamo poleg točk, ki pripadajo krožnici, ki jo želimo zaznati, tudi ostale, zunanje točke, ki se nam lahko pojavijo zaradi močnih šumov, napak pri zaznavanju in izbiranju pravih točk, ... Za to vrsto zazanavanja potrebujemo robustne metode, ki bodo takšne zunanje točke preprosto ignorirale ali pa jim dodelile zelo majhno vrednost pri končnem rezultatu. V ta namen se uporablja razne robustne variacije že zgoraj omenjenih pristopov [10], RANSAC, Houghove transformacije in podobno. Te metode so veliko bolj časovno in pomnilniško zahtevne, zato se ponavadi bolj splača dodelati algoritme za zaznavo robov in izbiro pravih točk ter uporabiti algoritme, omenjene pri prvem primeru.

Tretji primer je zaznavanje večih krožnic na eni sliki. Pojavi se tudi, ko ne vemo točno, koliko krožnic imamo, niti kje so postavljene. Pri tem pristopu se uporablja razne algoritme za glasovanje. Tudi tukaj lahko uporabimo malo popravljen RANSAC in prirejeno Houghovo transformacijo za krožnice ter z njimi poiščemo glavne kandidate za središča. Seveda pri teh metodah porabimo veliko več časa in prostora kot v prejšnjih dveh primerih. Močno so tudi odvisne od parametrov, ki jih uporabimo, torej se rezultati lahko zelo spremenijo pri najmanjših spremembah algoritma. Te metode uporabimo kot zadnjo možnost oziroma, če nas problem v to prisili.

To so glavni tipi zaznavanja glede na vhodne podatke, ki jih lahko srečamo pri nekem problemu. Kot je bilo že prej omenjeno, se bomo od sedaj naprej osredotočili na prvi primer, še bolj natančno na algoritme, porojene iz metode najmanjših kvadratov. Za nadaljne primere in teorijo privzamemo, da kot vhodne podatke sprejmemo množico točk, za katere želimo, da so del krožnice in glede na te točke izvajamo po metodi najmanjših kvadratov prirejanje z različnimi algoritmi. V naslednjih poglavjih bomo najprej predstavili osnovno teorijo in nekaj o obstoju in enoličnosti rešitve. Za tem sledi še nekaj implementacij pogostih algoritmov in analiza ter testi le teh.



Slika 3.1: Slika obravnavanih razdalj med točkami in krivuljo

3.2 Metoda najmanjših kvadratov

Iskanje rešitve, ki se najbolje prilega točkam po metodi najmanjših kvadratov [4] je osnovano na minimizaciji razdalje med točkami in iskano krivuljo (slika 3.1). Če imamo n točk oblike (x_i, y_i), kjer je $0 \leq i \leq n$, želimo minimizirati funkcijo

$$F = \sum_{i=1}^{n} d_i^2 \tag{3.1}$$

kjer je d_i Evklidska (geometrična) razdalja od točke (x_i, y_i) do krivulje. V našem primeru ima krivulja parametrizacijo

$$(x-a)^2 + (y-b)^2 = R^2$$

kjer je (a, b) center in R radij iskane krožnice. Iz tega sledi, da je v našem primeru razdalja d_i definirana z

$$d_i = \sqrt{(x_i - a)^2 + (y_i - b)^2} - R. \tag{3.2}$$

Na žalost obstaja direkten algoritem le za iskanje linearnih krivulj. Minimizacija funkcije (3.1) je nelinearen problem, ki nima preproste rešitve. Vsi znani algoritmi so iterativni in računsko dragi.

Pred pregledom algoritmov, pa se bomo posvetili še teoretičnim pogledom reševanja enačb (3.1) in (3.2). Govorili bomo o obstoju, enoličnosti rešitve in pravi izbiri parametrov.

3.3 Obstoj rešitve

Funkcija F v (3.1) je pri parametrizaciji krožnice s parametri a, b in R očitno zvezna [4]. Kot vemo, zvezna funkcija vedno doseže svoj minimum (lahko ni edini), če je definirana na kompaktnem območju. Naša funkcija F je definirana za vse a, b in vse $R \geq 0$, tako da njeno območje ni kompaktno in zaradi tega ta funkcija včasih ne doseže svojega minimuma.

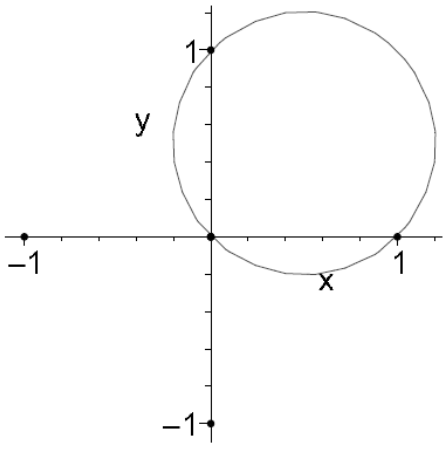
Za primer vzemimo $n \geq 3$ različnih točk, ki ležijo na premici. Potem lahko točke aproksimiramo s krožnico, ki se jim poljubno približa, ampak ker nobena krožnica ne more interpolirati $n \geq 3$ kolinearnih točk, vedno velja da je $F > 0$. Sledi, da najboljša rešitev ne obstaja. Za vsako krožnico, ki jo najdemo, obstaja naslednja, ki se točkam še bolj približa. Najboljša aproksimacija za te točke je premica, ki nam omogoči $F = 0$. Če želimo, da rešitev po metodi najmanjših kvadratov obstaja, moramo v našem modelu pustiti tudi premice, kar od sedaj naprej tudi storimo.

Sedaj vemo, da funkcija F, definirana za krožnice in premice vedno dobi svoj minimum, za vsako množico točk $n \geq 1$ (dokaz v [3]). S tem je obstoj rešitve po metodi najmanjših kvadratov zagotovljen.

3.4 Enoličnost rešitve

Presenetljivo nam LSF ne zagotovi enoličnosti rešitve. Tudi če funkcija F zavzame svoj minimum kot krožnica, najboljši približek ni zagotovo enoličen, saj lahko obstajajo tudi druge krožnice, ki minimizirajo F.

Primer, kdaj in kako se to lahko zgodi, je vzet iz [4]. Imamo štiri točke $(\pm 1, 0)$ in $(0, \pm 1)$ ter $k \geq 4$ točk, ki so točno v izhodišču $(0, 0)$. Ta primer je invarianta pri rotaciji γ za $\pi/2$. Torej, če krožnica S minimizira F, pri rotacijah za $\pi/2$,π in $3\pi/2$ dobimo tri druge krožnice, ki tudi minimizirajo F. Le še preveriti moramo, da velja $\gamma(S) \neq S$.



Slika 3.2: Slika prikazuje množico točk brez enolične rešitve

Opazimo, da je $\gamma(S) = S$ le v primeru, ko ima S središče v izhodišču. Pokazati moramo, da take krožnice ne morejo minimizirati F. Če ima krožnica radij r in center v $(0,0)$, potem je $F = 4(1-r^2) + kr^2$. Minimum te funkcije dosežemo v $r = 4/(k+4)$ in je enak $F_0 = 4k(k+4)$. Po predpostavki $k \geq 4$ lahko zagotovimo, da je minimum ≥ 2.

Vse kar potrebujemo, je najti krožnico za katero je $F < 2$. Vzemimo tisto, ki gre skozi točke $(0,0)$,$(0,1)$ in $(1,0)$. Zgreši le dve točki, $(-1,0)$ in $(0,-1)$, ter preprosto lahko vidimo da je $F < 2$. Ker F zavzame vrednosti, ki so manjše od F_0(ko je $k \geq 4$), krožnica, ki se najbolje prilega tem točkam, nima središča v $(0,0)$. Ne najdemo najboljše krožnice, vidimo pa da ni enolična. Slika 3.2 prikazuje zgornji primer.

3.5 Izbira parametrizacije

Težava s parametrizacijo krožnic s parametri a, b, R je, da lahko naključno postanejo zelo veliki, ko imamo nepopolne podatke, in so zato velikokrat napačno aproksimirani. Ne samo, da to vodi do izgube natančnosti pri aproksimaciji, ko odštevamo dve skoraj enaki števili v (3.2), ampak tudi povzroči pojave ravnih predelov in dolin v objektni funkciji, kar povzroči veliko težav iterativnim algoritmom, kjer se pojavi možnost, da se ne izvedejo.

Zaradi teh razlogov lahko uvedemo alternativno parametrizacijo, predstavljeno in uporabljeno v [4, 3], kjer je enačba krožnice definirana implicitno z

$$A(x^2 + y^2) + Bx + Cy + D = 0.$$

Opazimo da v primeru ko velja $A \neq 0$, dobimo krožnico in ko $A = 0$, premico, tako da lepo združuje obe možnosti, s katerimi se lahko srečamo v našem delu. Parametri morajo tudi zadoščati neenakosti $B^2 + C^2 - 4AD > 0$, da dobimo netrivialno krivuljo [3]. Ker morajo biti parametri definirani le do skalarnega večkratnika natančno, lahko vpeljemo pogoj

$$B^2 + C^2 - 4AD = 1.$$

Če dodatno zahtevamo $A \geq 0$, potem bo vsaka krožnica ali premica zadoščala enolični četverici (A, B, C, D) in obratno. Zaradi tehničnih razlogov parametra A ne omejimo, kar nam da dvojno parametrizacijo (vsaki krožnici pripadata dve četverici (A, B, C, D)). Za pretvorbe med geometrijskimi parametri a, b, R in algebraičnimi (A, B, C, D) imamo naslednje formule

$$a = -\frac{B}{2A}, \quad b = -\frac{C}{2A}, \quad R^2 = \frac{B^2 + C^2 - 4AD}{4A^2}$$

in

$$A = \pm\frac{1}{2R}, \quad B = -2Aa, \quad C = -2Ab, \quad D = \frac{B^2 + C^2 - 1}{4A}.$$

Razdaljo med točko (x_i, y_i) in krožnico lahko zapišemo kot

$$d_i = 2\frac{P_i}{1 + \sqrt{1 + 4AP_i}} \tag{3.3}$$

kjer je

$$P_i = A(x_i^2 + y_i^2) + Bx_i + Cy_i + D.$$

Zapis s parametri A, B, C, D ima veliko prednosti. Med njimi so, da lahko vsakega od parametrov eksplicitno omejimo in tako nimamo težav z naključnimi zelo velikimi števili, objektivna funkcija je sedaj gladka v vsakem parametru(na primer ni več singularnosti, ko se krožnica približuje premici) in iterativne metode nam ne morejo več pobegniti v neskončnost. Zaradi tega bomo novo parametrizacijo uporabljali pri nekaterih algoritmih in pozneje tudi pokazali prednosti, ki jih prinese pri določenih pogojih.

Poglavje 4

Geometrijske metode

V tem in v naslednjem poglavju bomo predstavili praktične rešitve za prirejanje krožnic množici točk. Naša glavna naloga je še vedno minimizacija prvotne nelinearne funkcije (3.1). Vse praktične rešitve se delijo v dve glavni večji skupini [3]:

1. iterativni algoritmi, ki konvergirajo k minimumu funkcije

$$F = \sum_{i=1}^{n} d_i^2$$

   kjer so d_i geometrijske (ortogonalne) razdalje od podanih točk do krožnice. Take vrste minimizacije F imenujemo *geometrijske metode za prirejanje*;

2. približni algoritmi, ki zamenjajo d_i z neko drugo, prirejeno funkcijo f_i in nato minimizirajo

$$F = \sum_{i=1}^{n} f_i^2.$$

   Ponavadi so f_i definirane s preprostimi algebraičnimi formulami. Te vrste algoritmov imenujemo *algebraične metode za prirejanje*.

Če pristopa primerjamo, geometrijske metode veljajo za bolj natančne (mnenje velja predvsem na podlagi praktičnih preizkusov; več o tem v šestem poglavju). Ampak po zasnovi je vsaka geometrijska metoda iterativna, ima možnost divergence in je v vsakem primeru računsko draga. Po drugi strani algebraične metode veljajo za preproste, zanesljive in hitre.

Pogosto uporabimo algebraično metodo za prvotno oceno parametrov pri nekem geometrijskem algoritmu in s tem poskušamo še izboljšati prvotni rezultat. So pa določeni moderni algebraični algoritmi že tako natančni, da so geometrijske izboljšave komaj opazne. V večini praktičnih primerov je dobro zastavljen algebraični pristop popolnoma zadosten.

Nadaljujemo s predstavitvijo nekaj osnovnih minimizacijskih pristopov, ki jih potrebujemo kasneje pri izpeljavi zelo znane Levenberg-Marquardt geometrijske metode in njene izboljšave za še večjo stabilnost. Kasneje sledi še nekaj najpopularnejših algebraičnih metod.

4.1 Minimizacija

Začnemo s kratkim opisom osnovnih metod za minimizacijo gladkih funkcij večih spremenljivk, predvsem tistih prilagojenih za metodo najmanjših kvadratov.

Gradientni spust. Pri prvi predstavljeni metodi gre za iskanje minimuma v nasprotni smeri izračunanega gradienta funkcije pri neki začetni vrednosti. Torej iščemo minimum gladke funkcije $F : \mathbb{R}^k \to \mathbb{R}$;

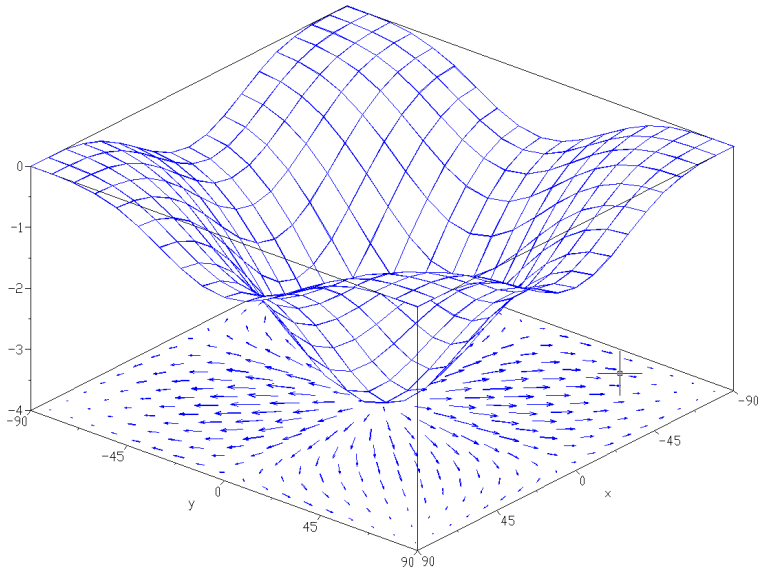
$$z = F(a) \in \mathbb{R}, \quad a = (a_1, ..., a_k) \in \mathbb{R}^k.$$

Iterativni postopki ponavadi izračunajo zaporedje točk $a^{(0)}, a^{(1)}, ...$ ki konvergira k točki, kjer F doseže svoj minimum. Začetno točko $a^{(0)}$ nekako izberemo, nato postopek sledi določenim pravilom za izračun $a^{(i+1)}$ iz $a^{(i)}$. To pomeni, da moramo za iskanje minimuma samo opisati pravila, kako dobimo naslednjo aproksimacijo a' iz trenutne aproksimacije a.

Vedno predpostavimo, da je F odvedljiva. Iz tega sledi, da lahko poiščemo gradientni vektor $\nabla F(a)$ (slika 4.1) in naslednji logični korak je, da se iz a premaknemo v nasprotno smer od $\nabla F(a)$, kjer se funkcija F zmanjšuje najhitreje. To metodo imenujemo *gradientni spust* in jo opišemo z naslednjo formulo

$$a' = a - \eta \nabla F(a),$$

kjer je $\eta > 0$ faktor hitrosti spusta. Določanje velikosti faktorja je zelo delikatno. Če nastavimo preveliko vrednost, lahko območje padca preskočimo, če pa premajhno, je naš napredek zelo počasen. Najlažji pristop je, da nastavimo $\eta = 1$,



Slika 4.1: Slika prikazuje smer gradientnih vektorjev glede na funkcijo

izračunamo a' in pogledamo, če je rezultat sprejemljiv. V primeru da ne, faktor η zmanjšamo in poskušamo ponovno, dokler nam ne uspe.

V splošnem je gradientni spust zelo zanesljiva metoda, a konvergira zelo počasi (v najboljšem primeru linearno).

Gauss-Newtonova metoda. Ta metoda je namenjena prav za reševanje problema nelinearnih najmanjših kvadratov. Torej imamo funkcijo

$$F(a) = \sum_{i=1}^{n} f_i^2(a)$$

k spremenljivk $a = (a_1, ..., a_k)$. Predpostavimo, da je $n > k$ in da so f_i odvedljive funkcije. Najprej aproksimiramo $F(a+h)$ s kvadratnim delom Taylorjevega razvoja:

$$F(a+h) \approx F(a) + D^T h + \frac{1}{2} h^T H h$$

kjer je

$$D = \nabla F(a) = 2 \sum_{i=1}^{n} f_i(a) \nabla f_i(a)$$

gradient od F in

$$H = \nabla^2 F(a) = 2 \sum_{i=1}^{n} [\nabla f_i(a)][\nabla f_i(a)]^T + 2 \sum_{i=1}^{n} f_i(a) \nabla^2 f_i(a) \tag{4.1}$$

Hessejeva matrika od F. Nato pri izračunu Hessejeve matrike (4.1) zadnjo vsoto odstranimo in H zamenjamo z

$$H^* = 2\sum_{i=1}^{n}[\nabla f_i(a)][\nabla f_i(a)]^T.$$

Razlog za to odločitev je, da odstranjeni del zelo malo vpliva na matriko in spreminjanje matrike ne vpliva na limitno točko, ampak le na pot do nje. Pridobimo pa tudi na tem, da je nova H^* vedno pozitivno semidefinitna in ni nam potrebno računati drugih odvodov.

Kvadratni izraz

$$F(a) + D^T h + \frac{1}{2}h^T H^* h$$

ima sedaj minimum, ki ga dosežemo v h, ki zadošča enačbi

$$D + H^* h = 0.$$

Iz tega sledi, da je $h = -(H^*)^{-1}D$ in s tem dobimo naslednje aproksimacijo, ki je $a' = a + h$.

Na koncu dodamo še nekaj metod linearne algebre, in sicer definiramo matriki $f = (f_1(a), ..., f_n(a))^T$ in

$$J = \begin{pmatrix} \partial f_1/\partial a_1 & \cdots & \partial f_1/\partial a_k \\ \vdots & \ddots & \vdots \\ \partial f_n/\partial a_1 & \cdots & \partial f_n/\partial a_k \end{pmatrix}$$

S tem dobimo $D = 2J^T f$ in $H^* = 2J^T J$. Iz tega sledi da je h rešitev

$$J^T J h = -J^T f, \tag{4.2}$$

kar je enakovredno rešitvi predoločenega linearnega sistema

$$Jh \approx -f,$$

ki je klasičen problem najmanjših kvadratov v linearni algebri.

Za numerično reševanje sistema (4.2) obstaja kar nekaj metod. Priporočena je na primer metoda Choleskega, ki je precej hitra in natančna. Če potrebujemo večjo numerično stabilnost, lahko uporabimo tudi QR razcep ali SVD.

Konvergenca te metode je že malo boljša. Brez spreminjanja matrike H bi bila konvergenca kvadratična. Z našo majhno spremembo to rahlo pokvarimo, a za to metodo še vedno velja, da konvergira skoraj kvadratično.

4.2 Levenberg-Marquardt

Prva predstavljena in kasneje tudi realizirana in testirana metoda bo Levenberg-Marquardtova. Temelji na popravku/izboljšavi Gauss-Newtonove metode in uporablja najboljše lastnosti zgoraj predstavljenih metod.

Uspešnost Gauss-Newtonove metode je lahko zelo problematična v primeru ko je matrika $N = J^T J$ skoraj singularna. Zaradi tega spremenimo matriko N v

$$N_\lambda = N + \lambda I$$

kjer je $\lambda > 0$ kontrolni parameter in I identiteta. Sedaj torej rešujemo nov sistem

$$N_\lambda h = -J^T f$$

za določitev h. Opazimo da je matrika N_λ za $\lambda > 0$ vedno pozitivno definitna in da so vse lastne vrednosti od N_λ večje ali enake λ.

V novem algoritmu računamo vrednosti malo drugače. Po izračunu h pogledamo, če nova aproksimacija $a' = a + h$ zmanjša vrednost F. Če je $F(a') < F(a)$, vzamemo nov približek in zmanjšamo λ za nek faktor (zmanjšamo vpliv λI in se s tem približujemo Gauss-Newtonovi metodi). V nasprotnem primeru približek a' zavrnemo in zvišamo λ za določen faktor ter ponovno rešimo rahlo spremenjeno enačbo $N_\lambda h = -J^T f$. To ponavljamo, dokler izračunani h in nov približek ne privedeta do nižje vrednosti F. To se bo zgodilo slej kot prej, saj se za velike λ metoda približuje gradientnemu spustu.

Z drugimi besedami, ko se λ viša, se izračunani vektor h ne samo zniža ampak se začne obračati v negativni smeri gradientnega vektorja $-\nabla F(a)$. Ko gre $\lambda \to \infty$, se dolžina h približuje nič, smer pa konvergira k smeri $-\nabla F(a)$.

Vidimo da Levenberg-Marquardt metoda združuje dve že prej predstavljeni klasični ideji:

1. Kvadratično aproksimacijo funkcije, ki deluje zelo dobro v bližini minimuma in nam omogoča zelo hitro, skoraj kvadratično konvergenco.

2. Uporabo gradientnega spusta, ki omogoča natančnost in zanesljivost tudi v težkih primerih.

Sedaj imamo vse potrebne formule za implementacijo osnovnega Levenberg-Marquardt algoritma. Ta algoritem velja za enega najboljših geometrijskih algoritmov tako v teoriji kot v praksi. Dejanski primer implementacije v Matlabu si lahko pogledate v osmem poglavju in sicer v prilogi 8.1.

4.3 Chernov-Lesort

Ker ima večina geometrijskih algoritmov težave z divergenco, bomo predstavili še en, malo bolj nenavaden algoritem, ki nam zagotavlja konvergenco. Gre za uporabo prej razloženega Levenberg-Marquardt algoritma pri drugačni parametrizaciji. To alternativno parametrizacijo smo že omenili v tretjem poglavju in sicer gre za zapis krožnice s formulo

$$A(x^2 + y^2) + Bx + Cy + D = 0$$

pod pogojem, da je

$$B^2 + C^2 - 4AD = 1.$$

Sedaj bomo obrazložili, kako uporabimo Levenberg-Marquardt algoritem na novem parametričnem prostoru.

Najprej definiramo novo koordinato Θ, ki je definirana z

$$B = \sqrt{1 + 4AD}\cos\theta, \quad C = \sqrt{1 + 4AD}\sin\theta$$

in s tem zamenja B in C. Sedaj lahko izvedemo minimizacijo funkcije (3.1) v parametričnem prostoru (A, D, θ). Razdalja d_i je izražena z enačbo (3.3), kjer je

$$\begin{aligned} P_i &= A(x_i^2 + y_i^2) + \sqrt{1 + AD}(x_i\cos\theta + y_i\sin\theta) + D \\ &= Az_i + Eu_i + D \end{aligned}$$

in zaradi krajšega zapisa uvedemo nove oznake $z_i = x_i^2 + y_i^2$, $E = \sqrt{1 + 4AD}$, $u_i = x_i\cos\theta + y_i\sin\theta$, $Q_i = \sqrt{1 + 4AP_i}$ in $R_i = 2(1 - Ad_i/Q_i)/(Q_i + 1)$. Parcialni odvodi glede na parametre se potem glasijo:

$$\begin{aligned} \partial d_i/\partial A &= (z_i + 2Du_i/E)R_i - d_i^2/Q_i \\ \partial d_i/\partial D &= (2Au_i/E + 1)R_i \\ \partial d_i/\partial\theta &= (-x_i\sin\theta + y_i\cos\theta)ER_i \end{aligned}$$

Na tem lahko uporabimo klasično Levenberg-Marquardt metodo. Algoritem, ki ga dobimo, je precej zapleten in drag glede na ostale metode, ki se uporabljajo, ampak konvergira zelo hitro, zato ga v splošnem lahko primerjamo z ostalimi.

Prednost algoritma je, kot smo že omenili, neodvisnost od začetnega približka. Vedno konvergira k minimumu. Ima nekaj težav pri singularnostih, ki se pojavijo v F, a se da vse odpraviti brez večjih težav. Več o sami izpeljavi in bolj podrobni analizi lahko najdete v [4], kjer sta avtorja tudi zasnovala idejo tega algoritma. Implementacijo v Matlabu najdete v osmem poglavju pod prilogo 8.2.

Poglavje 5

Algebraične metode

V prejšnjem poglavju smo razdelili vse praktične pristope za dodeljevanje krožnic točkam v dve večji grupi: (1) geometrijske, ki minimizirajo ortogonalno razdaljo od točk do krivulje in (2) algebraične, ki minimizirajo druge, večinoma algebraične, izraze.

V tem poglavju se bomo posvetili drugi vrsti metod. Najprej pa si bomo pogledali nekaj razlogov za njihovo uporabo.

(A) Geometrijske metode so iterativne in potrebujejo začetni približek, ki ga uporabijo za reševanje. Njihova hitrost in natančnost je precej odvisna od izbire le tega. Algebraične metode so direktne in nezahtevne procedure, ki nam lahko pomagajo pri njegovi izbiri.

(B) Določene aplikacije potrebujejo hitro obdelavo velikega števila podatkov. Preprosto si ne moremo privoščiti iterativnih metod, zato je hitra neiterativna metoda včasih preprosto edina možnost.

(C) Dobro narejeni in optimizirani algebraični algoritmi (primer sta kasneje obrazloženi Pratt in Taubin metodi) statistično delujejo skoraj tako natančno kot geometrijski. V veliko aplikacijah bo dobra algebraična metoda popolnoma zadostna, vse dodatne geometrijske izboljšave bodo neopazne.

V naslednjih poglavjih bomo predstavili tri algebraične metode. Prva, Kasa, je najbolj preprosta, a v določenih primerih zelo nenatančna. Drugi dve, Pratt in Taubin, sta že bolj kompleksni in zelo dobri, skoraj neopazno slabši od geometrijskih metod. Kot pri geometrijskih metodah, bomo predstavili glavno idejo, izpeljavo in implementacijo v Matlabu.

5.1 Kasa

Verjetno najbolj preprost in očiten pristop je Kasa metoda za dodeljevanje krožnic. Namesto klasične ortogonalne dolžine d_i vzamemo

$$f_i = (x_i - a)^2 + (y_i - b)^2 - R^2,$$

torej minimiziramo funkcijo

$$F_1 = \sum_{i=1}^{n} [(x_i - a)^2 + (y_i - b)^2 - R^2]^2.$$

Vidimo da je $f_i = 0$, če in samo če točka (x_i, y_i) leži na krožnici. Vidimo tudi, da je f_i majhen, če je točka blizu krožnice, torej je ideja minimizacije logična.

Odvodi te funkcije so nelinearni, a jih lahko naredimo linearne s preprosto spremembo parametrov

$$B = -2a, \quad C = -2b, \quad D = a^2 + b^2 - R^2.$$

Iz tega sledi

$$F_1 = \sum_{i=1}^{n} (z_i + Bx_i + Cy_i + D)^2,$$

kjer spet uporabimo okrajšavo $z_i = x_i^2 + y_i^2$.

Sedaj pri parcialnem odvajanju funkcije glede na B, C, D dobimo preprost sistem linearnih enačb

$$\begin{aligned} \overline{xx}B + \overline{xy}C + \overline{x}D &= -\overline{xz} \\ \overline{xy}B + \overline{yy}C + \overline{y}D &= -\overline{yz} \\ \overline{x}B + \overline{y}C + D &= -\overline{z} \end{aligned} \tag{5.1}$$

kjer uporabimo notacijo

$$\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i, \quad \overline{xx} = \frac{1}{n}\sum_{i=1}^{n} x_i^2, \quad \overline{xy} = \frac{1}{n}\sum_{i=1}^{n} x_i y_i, \quad \ldots$$

Ta sistem enačb nato rešimo z neko metodo metodo za reševanje linearnih sistemov, kot je LU razcep.

S tem dobimo B, C, D in z njimi le še izračunamo a, b, R z

$$a = -\frac{B}{2}, \quad b = -\frac{C}{2}, \quad R = -\frac{\sqrt{B^2 + C^2 - 4D}}{2}.$$

Kar se tiče obstoja, po metodi Kasa vedno dobimo rešitev, kot vidimo po sistemu enačb, ki jih rešujemo. Pri enoličnosti pa imamo dve možnosti. Algoritem vedno vrne eno rešitev, razen če so vse točke kolinearne, potem jih dobimo več.

Implementacija algoritma v Matlabu se nahaja v prilogi 8.3.

5.2 Pratt

Naslednja predstavljena bo Prattova metoda za iskanje krožnic. Pri analizi pristranskosti Kasa metode [3], opazimo da omenjena metoda pravzaprav minimizira

$$F_1 = \sum_{i=1}^{n} d_i^2 (d_i + 2R)^2.$$

Če so točke blizu krožnice, velja $|d_i| \ll R$, tako da je

$$F_1 \approx 4R^2 \sum_{i=1}^{n} d_i^2. \tag{5.2}$$

Chernov in Ososkov sta zato v [5] predlagala popravek algoritma, z dodajanjem dodatnega koeficienta. S tem dobimo $F_2 = (2R)^{-2} F_1$ ali

$$F_2(a, b, R) = (2R)^{-2} \sum_{i=1}^{n} [(x_i - a)^2 + (y_i - b)^2 - R^2]^2. \tag{5.3}$$

Pratt je vzel to idejo in iz nje izpeljal svoj algoritem, ki nam omogoča hitro in učinkovito reševanje zgornjega sistema. Spet se bomo vrnili k parametrom A, B, C, D, ki smo jih predstavili v poglavju 3. Zapis enačbe (5.3) v novih koordinatah je

$$F_2 = \sum_{i=1}^{n} \frac{(Az_i + Bx_i + Cy_i + D)^2}{B^2 + C^2 - 4AD},$$

kar si lahko poenostavljeno predstavljamo kot minimizacijo

$$F_3 = \sum_{i=1}^{n} (Az_i + Bx_i + Cy_i + D)^2 \tag{5.4}$$

pod pogojem

$$B^2 + C^2 - 4AD = 1.$$

Za reševanje tega minimizacijskega problema uporabimo nekaj metod linearne algebre. Najprej zapišemo funkcijo (5.4) v matrično obliko kot

$$F_3 = ||XP||^2 = P^T(X^TX)P,$$

kjer je $P = (A, B, C, D)^T$ vektor parametrov in

$$X = \begin{pmatrix} z_1 & x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ z_n & x_n & y_n & 1 \end{pmatrix}$$

razširjena matrika podatkov. Pogoj zapišemo kot

$$P^TBP = 1, \tag{5.5}$$

kjer je

$$B = \begin{pmatrix} 0 & 0 & 0 & -2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -2 & 0 & 0 & 0 \end{pmatrix}.$$

Sedaj, po teoriji o vezanih ekstremih, uvedemo Lagrangeov multiplikator η in minimiziramo funkcijo

$$\varphi(P, \eta) = P^T(X^TX)P - \eta(P^TBP - 1).$$

Odvajanje glede na P nam da pogoj $(X^TX)P = \eta BP$, torej mora biti η lastna vrednost matričnega para (X^TX, B), P pa njen pripadajoč lastni vektor. Matrika B je simetrična in ima štiri realne lastne vrednosti $\{1, 1, 2, -2\}$. Če je X^TX pozitivno definitna, so po Sylvestrovem zakonu o inerciji [8] lastne vrednosti matričnega para (X^TX, B) tudi vse realne, točno tri od njih so pozitivne in ena je negativna.

Moramo le še pogledati, kateri par (η, P) reši naš problem, torej minimizira F_3. Vidimo da je

$$F_3 = P^T(X^TX)P = \eta P^TBP = \eta,$$

zaradi pogoja (5.5), iz česar sledi da je F_3 minimizirana, ko je η najmanjša pozitivna lastna vrednost matrike $B^{-1}(X^TX)$. Po izračunu le te najdemo še pripadajoči lastni vektor $P = (A, B, C, D)$ in z njim končni rezultat algoritma.

To je glavna ideja tega algoritma, nekaj besed pa bomo namenili še izbiri algoritma za iskanje lastnih vektorjev. Uporabimo lahko poljubni algoritem, paziti pa moramo na stabilnost. V našem primeru smo uporabili SVD razcep in z njegovo pomočjo poiskali lastne vrednosti ter vektorje. Ta pristop velja za precej numerično stabilnega, uporabimo pa lahko tudi hitrejše, seveda na račun stabilnosti. Primer hitrejšega pristopa si lahko ogledate v [3], kjer je opisan tudi algoritem z uporabo Newtonove metode.

Realizacija v Matlabu se nahaja v prilogi 8.4.

5.3 Taubin

Taubinova metoda za iskanje krožnic je zelo podobna Prattovi. Spomnimo se aproksimacije (5.2), ki smo jo tam uporabili. Če spet predpostavimo, da je $|d_i| \ll R$, lahko to uporabimo kot

$$R^2 \approx (d_i + R)^2 = (x_i - a)^2 + (y_i - b)^2.$$

To lahko še izboljšamo s povprečjem

$$R^2 \approx \frac{1}{n}\sum_{i=1}^{n}(x_i - a)^2 + (y_i - b)^2,$$

kjer se bodo pozitivni in negativni odstopi izničili. Sedaj lahko (5.2) zapišemo kot

$$F_1 \approx \frac{4}{n}[\sum_{i=1}^{n}(x_i - a)^2 + (y_i - b)^2][\sum_{i=1}^{n} d_i^2].$$

Ker hočemo minimizirati $\sum d_i^2$, lahko to dosežemo s minimizacijo funkcije

$$F_4(a, b, R) = \frac{\sum_{i=1}^{n}[(x_i - a)^2 + (y_i - b)^2 - R^2]^2}{4n^{-1}\sum_{i=1}^{n}(x_i - a)^2 + (y_i - b)^2},$$

kar je glavna ideja Taubinove metode.

Ponovno vpeljemo parametre A, B, C, D, tokrat dobimo funkcijo

$$F_4 = \frac{\sum_{i=1}^{n}(Az_i + Bx_i + Cy_i + D)^2}{n^{-1}\sum_{i=1}^{n}(4A^2z_i + 4ABx_i + 4ACy_i + B^2 + C^2)}.$$

Kot v prejšnjem primeru poenostavimo minimizacijo na (5.4), pod pogojem

$$4A^2\overline{z} + 4AB\overline{x} + 4AC\overline{y} + B^2 + C^2 = 1.$$

Vidimo da pravzaprav minimiziramo enako funkcijo kot pri Prattovem algoritmu, edina razlika je v pogoju, ki ga dodamo.

V matrični formi je to minimizacija

$$F_3 = ||XP||^2 = P^T(X^T X)P,$$

kjer pogoj zapišemo kot $P^T TP = 1$, kjer je

$$T = \begin{pmatrix} 4\overline{z} & 2\overline{x} & 2\overline{y} & 0 \\ 2\overline{x} & 1 & 0 & 0 \\ 2\overline{y} & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

Kot v prejšnjem algoritmu je rešitev $P = (A, B, C, D)$, ki je lastni vektor matričnega para $(X^T X, T)$.

Vidimo, da če fiksiramo parametre A, B, C, lahko izračunamo pogojen globalen minimum v D z enačbo

$$D = -A\overline{z} - B\overline{x} - C\overline{y}.$$

Če predpostavimo, da so podatki centrirani ($\overline{x} = \overline{y} = 0$), potem lahko preprosto parameter D zamenjamo z $-A\overline{z}$ in ga odstranimo. Nova funkcija za minimizacijo izgleda sedaj takole

$$F_5(A, B, C) = \sum_{i=1}^{n} (A(z_i - \overline{z}) + Bx_i + Cy_i)^2,$$

pod pogojem

$$4\overline{z}A^2 + B^2 + C^2 = 1.$$

Sedaj uvedemo še nov parameter $A_0 = 2\overline{z}^{1/2}A$ in dobimo

$$F_5(A_0, B, C) = \sum_{i=1}^{n} (A_0 \frac{z_i - \overline{z}}{2\overline{z}^{1/2}} + Bx_i + Cy_i)^2$$

pod pogojem

$$A_0^2 + B^2 + C^2 = 1.$$

Funkcijo sedaj zapišemo v matrični formi

$$F_5 = ||X_0 P_0||^2 = P_0^T (X_0^T X_0) P_0$$

kjer je $P_0 = (A_0, B, C)^T$ in

$$X_0 = \begin{pmatrix} (z_1 - \overline{z})/(2\overline{z}^{1/2}) & x_1 & y_1 \\ \vdots & \vdots & \vdots \\ (z_n - \overline{z})/(2\overline{z}^{1/2}) & x_n & y_n \end{pmatrix}.$$

Minimum funkcije je ponovno lastni vektor, ki pripada najmanjši lastni vrednosti matrike $X_0^T X_0$. Matrika je simetrična in pozitivno definitna, torej so vse lastne vrednosti realne in pozitivne.

Vektor P_0 lahko poiščemo na veliko načinov. V praksi smo uporabili SVD algoritem za iskanje lastnih vrednosti in poiskali lastni vektor, ki pripada najmanjši. Druge, tudi hitrejše rešitve si lahko pogledate v [3]. Realizacija v Matlabu se nahaja v prilogi 8.5.

Poglavje 6

Testi in analiza

V tem poglavju se bomo posvetili testiranju in analizi prej predstavljenih metod. Vsi uporabljeni algoritmi so vzeti iz [6], razloženi in izpeljani v [3] ter se nahajajo tudi v poglavju 8.

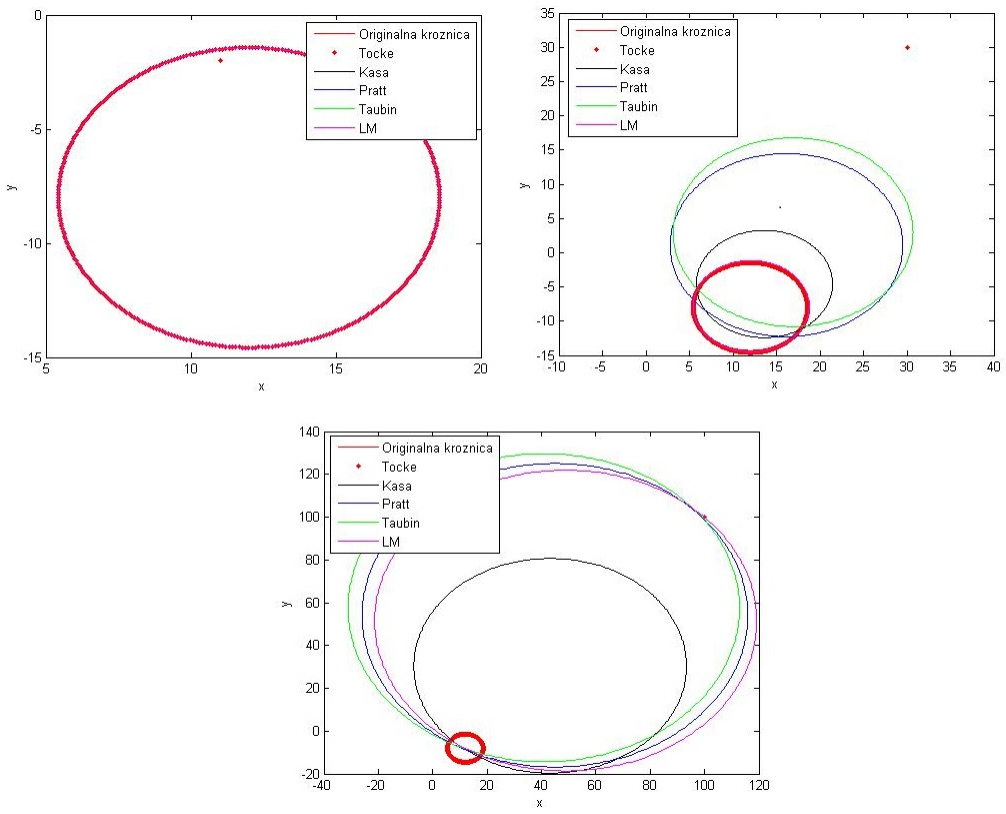
Začeli bomo z robustnostjo in natančnostjo algoritmov, za konec pa bomo na hitro ocenili tudi hitrost predstavljenih metod. Osredotočili so bomo bolj na primerjave med predstavljenimi algoritmi in predstavitev situacij, kjer bi bila uporaba vsakega od njih primerna.

6.1 Robustnost

Pričeli bomo z robustnostjo algoritmov. Kot test bomo vzeli 360 točk, ki ležijo na krožnici s središčem v $(12, -8)$ in radijem $R = 6.57$. Kot opomba, pri testiranju geometrijskih metod uporabljamo le Levenberg-Marquardt algoritem, ki služi kot predstavnik vseh ostalih, saj za preproste primere z dobrim začetnim približkom nimamo težav s konvergenco.

Začnemo s primerom, ko vse točke ležijo na krožnici. V tem primeru vsi predstavljeni, geometrijski in algrebraični, algoritmi najdejo točno rešitev brez napake (slika 6.2).

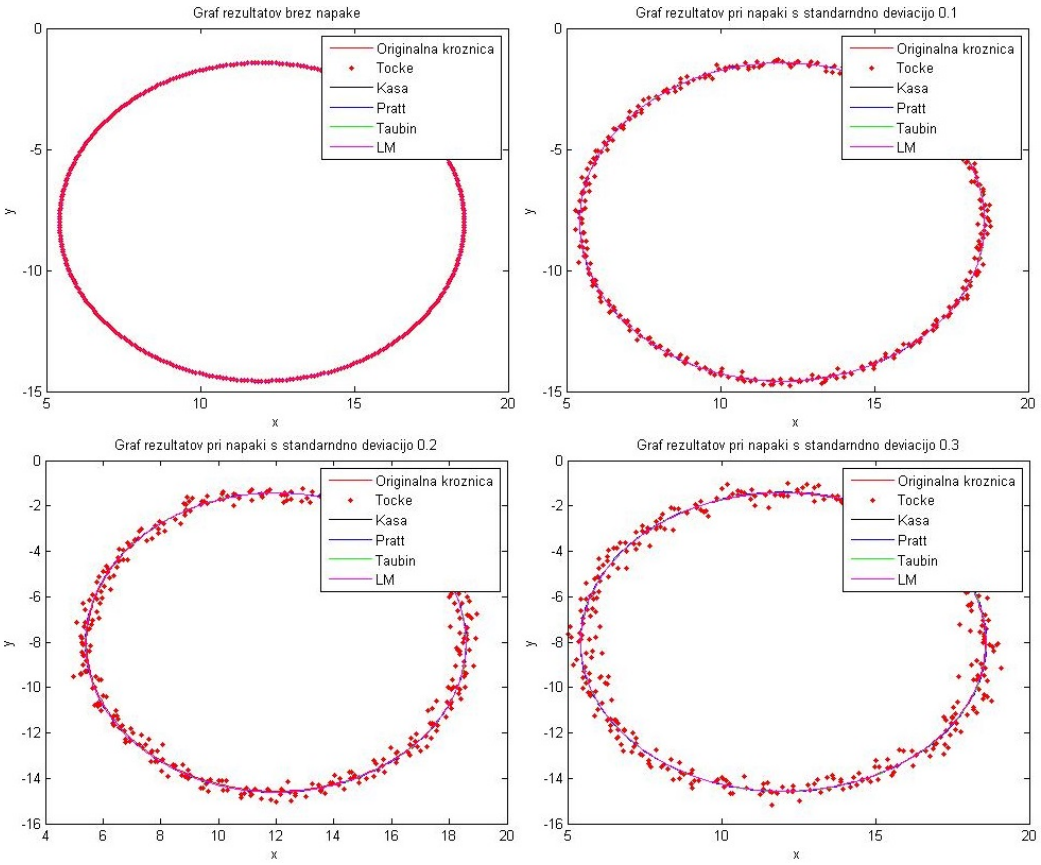
Zanima pa nas seveda kako se obnesejo pri morebitnih napakah. Najprej pogledamo zunanje točke (niso del opazovanega objekta). Že iz zasnove in izpeljave algoritmov vidimo, da so vse predstavljene metode zelo občutljive na njih, glede na to da pri vsakem pristopu minimiziramo neko vrsto razdalje od točke. Napaka ni



Slika 6.1: Slika prikazuje deformiranost rešitev pri zunanjih točkah

tako očitna pri manjših odstopih kot lahko vidimo na sliki 6.1, pri bolj oddaljenih zunanjih točkah pa preprosto algoritmi odpovedo. To je seveda po pričakovanjih, v takih primerih uporabimo bolj robustne metode kot so RANSAC, Houghove transformacije, razne druge metode za glasovanje, ..., ki posamezne točke, ki preveč odstopajo, preprosto odstranijo.

Druga vrsta so šumi in napake pri zaznavanju. Pri tem tipu smo preverili zanesljivost algoritmov pri standarno normalno porazdeljenih napakah na točkah. Uporabljamo enake podatke kot v zgornjem primeru, le da vsako izmed izračunanih točk na krožnici premaknemo za nek koeficient, ki ga pridobimo iz standardno normalne porazdelitve z neko prej določeno standardno deviacijo. Kot vidimo na sliki 6.2, se vse metode precej dobro obnesejo pri šumih in rahlo razpršenih točkah. Tudi pri pregledu parametrov se rezultati algebraičnih metod le malo razlikujejo



Slika 6.2: Slika prikazuje rezultate algoritmov pri različnih standardnih deviacijah napak

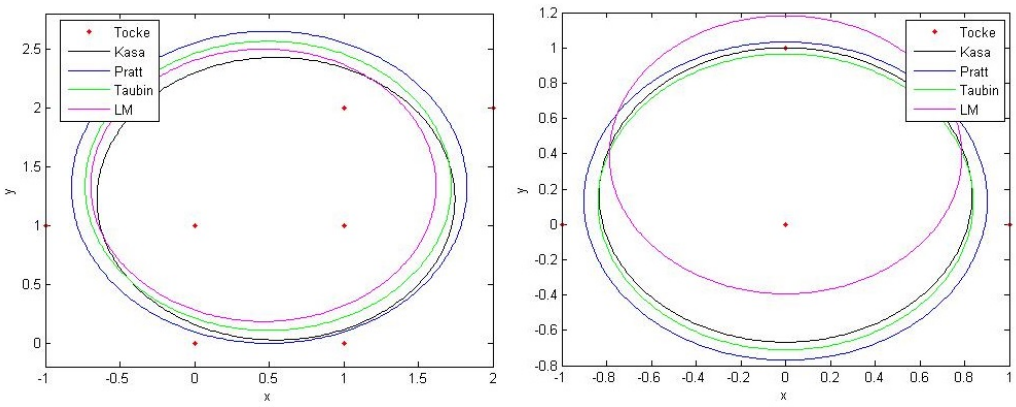
od geometrijskih. Napake so skoraj neopazne.

Vidimo da se algoritmi dobro obnesejo na primerih, kjer zaznavamo krožnico na razpršenih podatkih, a moramo biti zelo pazljivi na zunanje točke.

6.2 Natančnost

Nadaljujemo z natančnostjo algoritmov. Tu si bomo pogledali glavne razlike med algoritmi in primere zaznavanja na zelo malo točkah oziroma na nepopolnih podatkih (dodeljevanje krožnice krožnemu loku).

Najprej, kako se predstavljene metode obnašajo na naključni množici točk. Ta

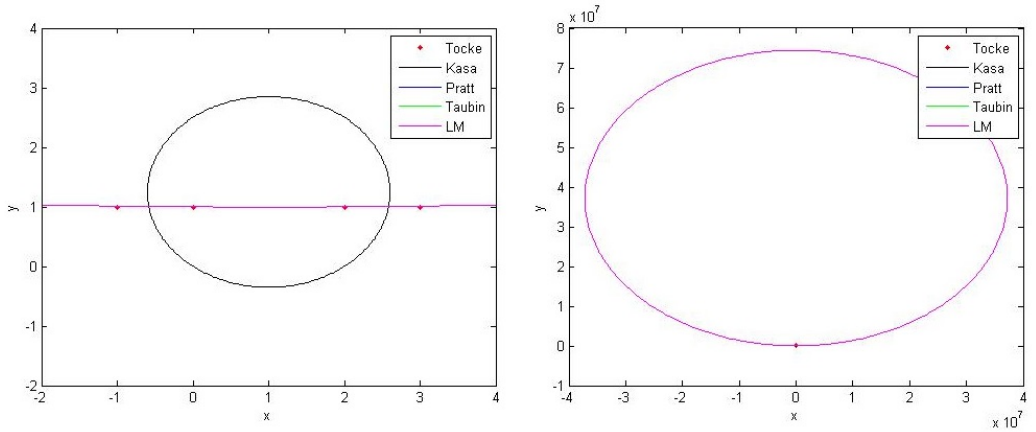


Slika 6.3: Slika prikazuje rezultate algoritmov na naključno porazdeljenih točkah

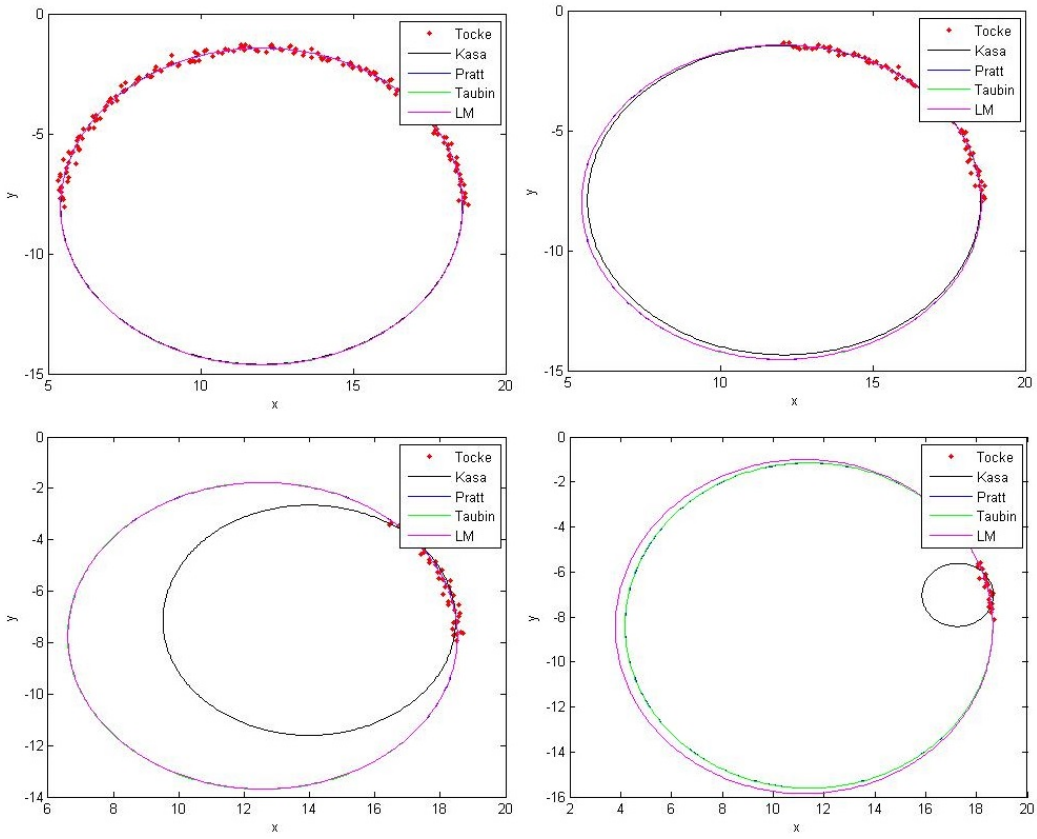
primer uporabimo, ker najbolj poudari razlike med rezultati. Kot lahko vidimo na sliki 6.3, nam geometrijska metoda LM poda rešitev z najmanjšim radijem in rahlo premaknjenim središčem glede na ostale. Seveda, če iščemo rešitev po najmanjših geometrijskih razdaljah do točk, nam ta metoda ponudi najbolj natančen rezultat. A kot smo že prej omenili, najboljša krožnica, ki se prilega neki množici točk, ni določena. Algebraične metode ponudijo zelo podobne rešitve, Kasa z najmanjšim radijem, zatem Taubinova in na koncu Prattova metoda z največjim radijem.

Nadaljujemo s primerom kolinearnih točk. Rezultate prikazuje slika 6.4, kjer vidimo, da geometrijska LM metoda konvergira proti neskončno veliki krožnici skozi točke. Algoritem se ustavi pri maksimalnem možnem številu iteracij. Kasa algoritem nas opozori, da rang matrike ni poln in vrne majhno krožnico, ki aproksimira te točke. Oba, Prattov in Taubinov, algoritma pa nas opozorita, da je matrika za izračunavo lastnih vrednosti singularna in da gre za premico. Pri teh dveh metodah so vrnjeni parametri za krožnico neskončni.

Kot zadnje si bomo pogledali aproksimacije krožnic na krožnih lokih. Nekaj testov si lahko ogledate na sliki 6.5, uporabili smo enake podatke kot pri testih robustnosti in sicer s standardno devianco napake 0.1 na 1/2, 1/4, 1/8, 1/16 krožnice. Naredili smo tudi bolj obsežen test, katerega rezultate si lahko ogledate v tabeli 6.1. Gre za tabelo, ki prikazuje izračunane približke radija glede na dolžino krožnega loka. Pri vsaki meritvi smo vzeli 20 po krožnem loku enakomerno po-



Slika 6.4: Slika prikazuje rezultate algoritmov na kolinearnih točkah



Slika 6.5: Slika prikazuje rezultate algoritmov na krožnih lokih

	Kasa			Taubin			LM		
α	q1	q2	q3	q1	q2	q3	q1	q2	q3
360°	6.5563	6.5714	6.5863	6.5563	6.5714	6.5863	6.5557	6.5707	6.5857
300°	6.5563	6.5718	6.5870	6.5564	6.5719	6.5871	6.5557	6.5712	6.5864
240°	6.5530	6.5705	6.5888	6.5536	6.5712	6.5895	6.5530	6.5707	6.5889
180°	6.5365	6.5669	6.5972	6.5408	6.5712	6.6014	6.5403	6.5709	6.6010
120°	6.4674	6.5408	6.6134	6.4971	6.5723	6.6461	6.4971	6.5725	6.6465
90°	6.3316	6.4604	6.5970	6.4367	6.5693	6.7104	6.4375	6.5693	6.7112
60°	5.7595	6.0105	6.2924	6.2611	6.5644	6.9024	6.2639	6.5655	6.9050
45°	4.7370	5.0626	5.4397	6.0426	6.5684	7.2179	6.0508	6.5738	7.2215
30°	2.3218	2.6633	3.0368	5.4548	6.5471	8.1643	5.4635	6.5638	8.1601

Tabela 6.1: Kvartilni približki za radij krožnice glede na velikost krožnega loka pri treh metodah (Kasa, Taubin, LM); pravi radij je $R = 6.5700$

razdeljenih točk s standardno devianco napake 0.1. Nato smo za vsako meritev naredili 10000 ponovitev in zapisali kvartile izračunanih radijev, ki nam pokažejo približno v katerem intervalu se nahaja večina rezultatov. V splošnem se izkaže, da Kasa algoritem začne pri krajših lokih močno podcenjevati radij in vrača skoraj neuporabne rezultate. Prat in Taubin se zelo dobro držita geometrijske rešitve in začneta rahlo odstopati šele pri precej kratkih lokih. Velja, da je Taubinov malo boljši od Prattovega algoritma [3], a sta oba tako blizu geometrijski rešitvi, da lahko v večini realnih problemov uporabimo kateregakoli od treh. Seveda moramo imeti v mislih, da je vsak geometrijski algoritem vsaj 4-5 krat počasnejši od algebraičnega.

6.3 Hitrost

Na kratko bomo ocenili tudi hitrost uporabljenih algoritmov. To bomo storili v dveh delih, saj se časovna zahtevnost med geometrijskimi in algebraičnimi algoritmi močno razlikuje. V splošnem se algebraični algoritem izvede hitreje kot ena iteracija geometrijskega.

Začeli bomo torej z geometrijskimi, bolj točno, dvema geometrijskima, ki smo ju predstavili predhodno. Tu nas zanima število računskih operacij na iteracijo in

hitrost konvergence. S tema podatkoma lahko ocenimo približno hitrost geometrijskega algoritma. Levenberg-Maequardtov algoritem potrebuje $12n + 41$ računskih operacij s plavajočo vejico na iteracijo, medtem ko jih Chernov-Lesortov potrebuje kar $39n + 40$. Izkaže pa se da ima Chernov-Lesortov poleg konvergence iz poljubnega začetnega približka tudi eno najhitrejših med geometrijskimi algoritmi (potrebuje veliko manj iteracij od ostalih). V testnih rezultatih v [4] opazimo, da je Levenberg-Maequardtov le enkrat hitrejši od Chernov-Lesortov algoritma, kar se nam v določenih primerih splača žrtvovati zavoljo konvergence. Tu lahko tudi omenimo, da Levenberg-Maequardtov velja za enega najhitrejših geometrijskih algoritmov [4]. Kot vidimo, je hitrost algoritma močno odvisna od hitrosti konvergence, ki pa je odvisna od natančnosti začetnega približka. V ta namen se ponavadi uporablja hitre algebraične metode, ki nam podajo začetni približek, ki ga potem z izbrano geometrijsko metodo še izboljšamo.

Sedaj pa še nekaj o hitrosti algebraičnih metod. Kasa algoritem velja za najhitrejšega, saj zaradi svoje preprostosti preprosto ne dovoli dodatne optimizacije. Tudi oba predstavljena, Pratt in Taubin, algoritma se mu lahko zelo približata s pravo implementacijo. Za hitro izvajanje uporabimo Newtonovo metodo pri iskanju najmanjše lastne vrednosti (izpeljavi za oba algoritma v [3]). Seveda v tem primeru izgubimo kar precej numerične stabilnosti. To rešujemo z iskanjem lastnih vrednosti s pomočjo SVD razcepa, ki nam omogoča veliko večjo stabilnost ko so matrike skoraj singularne. Uporabili smo ga tudi v tej diplomski nalogi. Slednji pristop velja za dva do trikrat počasnejšega od Kasa ali implementacije z Newtonovim algoritmom. Še vedno pa so vsi algebraični pristopi vsaj desetkrat hitrejši od geometrijskih.

Poglavje 7

Zaključek

V diplomski nalogi smo si najprej na kratko pogledali nekaj osnovnih pojmov pri računalniškem vidu in predstavili nekaj pristopov, ki jih lahko uporabimo pri implementaciji splošne rešitve za zaznavanje objektov.

V naslednjem delu smo se usmerili na problem zazanavanja krožnic. Predstavili smo nekaj osnovnih prijemov, osredotočili pa smo se na zaznavanje z metodo najmanjših kvadratov. V tej smeri smo nadaljevali tudi v naslednjih dveh poglavjih s predstavitvijo geometrijskih in algebraičnih metod, narejenih na podlagi najmanjših kvadratov. Pri geometrijskih smo predstavili Lavenberg-Marquardt metodo in njeno izpeljavo za zagotovljeno konvergenco iz poljubnega začetnega približka. Za predstavnike algebraičnih pa smo predstavili in izpeljali Kasa, Prattovo in Taubinovo metodo.

Nadaljevali smo s testiranjem omenjenih algoritmov, kjer smo si pogledali najprej nekaj o njihovi robustnosti, kasneje pa še o natančnosti in hitrosti. Ugotovili smo, da so vse pregledane metode močno občutljive na zunanje točke, se pa zelo dobro obnesejo pri šumih na podatkih, ki smo jih simulirali s standardno normalno porazdeljenimi napakami. Naredili smo tudi nekaj simulacij algoritmov na naključnih točkah, kjer smo si pogledali njihove splošne karakteristike. Tu se je pokazala pristranskost algebraičnih algoritmov, na primer Kasa velikokrat podcenjuje radij ipd. Testirano je bilo tudi obnašanje pri kolinearnih točkah in zaznavanje na krožnih lokih. Za kolinearne točke Prattov in Taubinov edina najdeta premico, Kasa poišče krožnico, geometrijski Lavenberg-Marquardtov pa konvergira proti krožnici z neskončnim radijem. Pri krožnih lokih spet opazimo pristranskost Kasa

algoritma, močno podcenjuje radije, Pratt in Taubin pa sta zelo lepo držala rezultate blizu geometrijskim približkom, skoraj neopazno se razlikujeta. Hitrosti algoritmov smo ocenili samo na hitro. Algebraični se ne razlikujejo veliko, imajo zelo podobne hitrosti izvajanja. Pri geometrijskih je Lavenberg-Marquardtov precej hitrejši od Chernov-Lesortovega, a moramo paziti na začetni približek in njegovo konvergenco.

Diplomska naloga služi kot pregled problema iskanja/zaznavanja krožnic z metodo najmanjših kvadratov. Pregledali smo nekaj osnovnih, najbolj uporabljenih metod in jih testirali, za boljši občutek, kdaj je uporaba posamezne najbolj primerna. S tem delom omogočamo hitro in utemeljeno izbiro algoritma za naš problem ter obenem tudi bolj teoretičen pregled, ki ga lahko uporabimo za nadaljne akademske potrebe.

Poglavje 8

Priloga: Algoritmi v Matlabu

Vsi priloženi algoritmi so vzeti iz [6]. Podrobnosti o izpeljavi in realizaciji lahko najdete v [3].

Priloga 8.1: Levenberg-Marquardtov algoritem za krožnice

```
function Par = LM(XY,ParIni,LambdaIni)

%-------------------------------------------------------------------------
%
%     Geometrijska metoda za dodeljevanje kroznic (minimizacija ortogonalnih razdalj)
%     bazirana na standardni Levenberg-Marquardt metodi
%        v polnem (a,b,R) parametricnem prostoru
%
%     Vhodni podatki:  XY(n,2) je mnozica tock n koordinat x(i)=XY(i,1), y(i)=XY(i,2)
%              ParIni = [a b R] je prvotni priblizek (vpise uporabnik)
%              LambdaIni je zacetna vrednost popravnega faktorja lambda
%                (ni obvezno; ce manjka, ga LM nastavi na 1)
%
%     Izhodni podatki: Par = [a b R] je dodeljena kroznica:
%                           sredisce (a,b) in radij R
%
%-------------------------------------------------------------------------

if (nargin < 3), LambdaIni = 1; end;  % ce Lambda(zacetni) ni podan ga nastavi na 1

epsilon=0.000001;     % tolerance

IterMAX = 50;       % maksimalno stevilo iteracij;

lambda_sqrt = sqrt(LambdaIni);

Par = ParIni;      % zacnemo s prvotnim priblizkom

[J,g,F] = CurrentIteration(Par,XY);     % izracunamo objektivno funkcijo in njene odvode

for iter=1:IterMAX          %  glavna zanka

    while (1)          %  podzanka za spreminjanje parametra lambda

        DelPar = [J; lambda_sqrt*eye(3)]\[g; zeros(3,1)];   % kandidat
```

```
        progress = norm(DelPar)/(norm(Par)+epsilon);
        if (progress < epsilon)  break;  end;                % pravilo za ustavitev
        ParTemp = Par - DelPar';
        [JTemp,gTemp,FTemp] = CurrentIteration(ParTemp,XY);
                % objektivna funkcija + odvodi

        if (FTemp < F && ParTemp(3)>0)          %   izboljsava
           lambda_sqrt = lambda_sqrt/2;   % zmanjsamo lambda, naslednja iteracija
           break;
        else                                %   ni izboljsave
           lambda_sqrt = lambda_sqrt*2; % povecamo lambda, ponovno izracunamo
           continue;
        end
    end

    if (progress < epsilon)  break;  end;              % pravilo za ustavitev
    Par = ParTemp;  J = JTemp;  g = gTemp;  F = FTemp;  % ponastavimo iteracijo
end    %  konec glavne zanke
end    %    LM

%================ function CurrentIteration ================

function [J,g,F] = CurrentIteration(Par,XY)

%    izracunamo objektivno funkcijo F njene odvode za trenuten priblizek Par

Dx = XY(:,1) - Par(1);
Dy = XY(:,2) - Par(2);
D = sqrt(Dx.*Dx + Dy.*Dy);
J = [-Dx./D, -Dy./D,  -ones(size(XY,1),1)];
g = D - Par(3);
F = norm(g)^2;

end  %  CurrentIteration
```

Priloga 8.2: Chernov-Lesortov algoritem za krožnice

```
function Par = LMA(XY, ParIni)

%--------------------------------------------------------------------------
%
%     Geometrijska metoda za dodeljevanje kroznic (minimizacija ortogonalnih razdalj)
%     bazirana na Levenberg-Marquardt metodi z
%     "algebraicnimi parametri" A,B,C,D  pod pogojem B*B+C*C-4*A*D=1
%        N. Chernov and C. Lesort, "Least squares fitting of circles",
%        J. Math. Imag. Vision, Vol. 23, 239-251 (2005)
%
%     Vhodni podatki:  XY(n,2) je mnozica tock n koordinat x(i)=XY(i,1), y(i)=XY(i,2)
%              ParIni = [a b R] je prvotni priblizek (vpise uporabnik)
%
%     Izhodni podatki: Par = [a b R] je dodeljena kroznica:
%                           sredisce (a,b) in radij R
%
%--------------------------------------------------------------------------

factorUp=10;  factorDown=0.04;
lambda0=0.01;  epsilon=0.000001;
IterMAX = 50;  AdjustMax = 20;
Xshift=0;  Yshift=0;  dX=1;  dY=0;

n = size(XY,1);       % stevilo tock

%    zacnemo s prvotnim priblizkom

anew = ParIni(1) + Xshift;
bnew = ParIni(2) + Yshift;

Anew = 1/(2*ParIni(3));
aabb = anew*anew + bnew*bnew;
Fnew = (aabb - ParIni(3)*ParIni(3))*Anew;
Tnew = acos(-anew/sqrt(aabb));
if (bnew > 0)
    Tnew = 2*pi - Tnew;
end

VarNew = VarCircle(XY, ParIni);

%     inicializiramo lambdo in iteracijo

lambda = lambda0;  finish = 0;

for iter=1:IterMAX

    Aold = Anew;  Fold = Fnew;  Told = Tnew;  VarOld = VarNew;

    H = sqrt(1+4*Aold*Fold);
    aold = -H*cos(Told)/(Aold+Aold) - Xshift;
    bold = -H*sin(Told)/(Aold+Aold) - Yshift;
    Rold = 1/abs(Aold+Aold);

%           racunamo momente

    DD = 1 + 4*Aold*Fold;
    D = sqrt(DD);
    CT = cos(Told);
    ST = sin(Told);

    H11=0; H12=0; H13=0; H22=0; H23=0; H33=0; F1=0; F2=0; F3=0;

    for i=1:n
```

```
        Xi = XY(i,1) + Xshift;
        Yi = XY(i,2) + Yshift;
        Zi = Xi*Xi + Yi*Yi;
        Ui = Xi*CT + Yi*ST;
        Vi = -Xi*ST + Yi*CT;

        ADF = Aold*Zi + D*Ui + Fold;
        SQ = sqrt(4*Aold*ADF + 1);
        DEN = SQ + 1;
        Gi = 2*ADF/DEN;
        FACT = 2/DEN*(1 - Aold*Gi/SQ);
        DGDAi = FACT*(Zi + 2*Fold*Ui/D) - Gi*Gi/SQ;
        DGDFi = FACT*(2*Aold*Ui/D + 1);
        DGDTi = FACT*D*Vi;

        H11 = H11 + DGDAi*DGDAi;
        H12 = H12 + DGDAi*DGDFi;
        H13 = H13 + DGDAi*DGDTi;
        H22 = H22 + DGDFi*DGDFi;
        H23 = H23 + DGDFi*DGDTi;
        H33 = H33 + DGDTi*DGDTi;

        F1 = F1 + Gi*DGDAi;
        F2 = F2 + Gi*DGDFi;
        F3 = F3 + Gi*DGDTi;
    end

    for adjust=1:AdjustMax

%           razcep  Choleskega

        G11 = sqrt(H11 + lambda);
        G12 = H12/G11;
        G13 = H13/G11;
        G22 = sqrt(H22 + lambda - G12*G12);
        G23 = (H23 - G12*G13)/G22;
        G33 = sqrt(H33 + lambda - G13*G13 - G23*G23);

        D1 = F1/G11;
        D2 = (F2 - G12*D1)/G22;
        D3 = (F3 - G13*D1 - G23*D2)/G33;

        dT = D3/G33;
        dF = (D2 - G23*dT)/G22;
        dA = (D1 - G12*dF - G13*dT)/G11;

%           posodobimo parametre

        Anew = Aold - dA;
        Fnew = Fold - dF;
        Tnew = Told - dT;

        if (1+4*Anew*Fnew < epsilon && lambda>1)
            Xshift = Xshift + dX;
            Yshift = Yshift + dY;

            H = sqrt(1+4*Aold*Fold);
            aTemp = -H*cos(Told)/(Aold+Aold) + dX;
            bTemp = -H*sin(Told)/(Aold+Aold) + dY;
            rTemp = 1/abs(Aold+Aold);

            Anew = 1/(rTemp + rTemp);
            aabb = aTemp*aTemp + bTemp*bTemp;
            Fnew = (aabb - rTemp*rTemp)*Anew;
```

```
            Tnew = acos(-aTemp/sqrt(aabb));
            if bTemp > 0
               Tnew = 2*pi - Tnew;
            end
            VarNew = VarOld;
            break;
        end

        if 1+4*Anew*Fnew < epsilon
           lambda = lambda * factorUp;
           continue;
        end

        DD = 1 + 4*Anew*Fnew;
        D = sqrt(DD);
        CT = cos(Tnew);
        ST = sin(Tnew);

        GG = 0;

        for i=1:n
            Xi = XY(i,1) + Xshift;
            Yi = XY(i,2) + Yshift;
            Zi = Xi*Xi + Yi*Yi;
            Ui = Xi*CT + Yi*ST;

            ADF = Anew*Zi + D*Ui + Fnew;
            SQ = sqrt(4*Anew*ADF + 1);
            DEN = SQ + 1;
            Gi = 2*ADF/DEN;
            GG = GG + Gi*Gi;
        end

        VarNew = GG/(n-3);

        H = sqrt(1+4*Anew*Fnew);
        anew = -H*cos(Tnew)/(Anew+Anew) - Xshift;
        bnew = -H*sin(Tnew)/(Anew+Anew) - Yshift;
        Rnew = 1/abs(Anew+Anew);

%             pogledamo ce izboljsamo

        if VarNew <= VarOld      %   izboljsava
           progress = (abs(anew-aold) + abs(bnew-bold) + abs(Rnew-Rold))/(Rnew+Rold);
           if progress < epsilon
              Aold = Anew;
              Fold = Fnew;
              Told = Tnew;
              VarOld = VarNew;
              finish = 1;
              break;
           end
           lambda = lambda * factorDown;
           break;
        else                     %   ni izboljsave
           lambda = lambda * factorUp;
           continue;
        end
    end
    if finish == 1
       break;
    end
end
```

```
H = sqrt(1+4*Aold*Fold);
Par(1) = -H*cos(Told)/(Aold+Aold) - Xshift;
Par(2) = -H*sin(Told)/(Aold+Aold) - Yshift;
Par(3) = 1/abs(Aold+Aold);

end  % LMA

function Var = VarCircle(XY,Par)

%-------------------------------------------------------------------------
%
%              racunanje varianc razdalj
%              od tock (XY) do kroznice Par = [a b R]
%
%-------------------------------------------------------------------------


n = size(XY,1);        % stevilo tock
Dx = XY(:,1) - Par(1);   Dy = XY(:,2) - Par(2);
D = sqrt(Dx.*Dx + Dy.*Dy) - Par(3);

Var = D'*D/(n-3);

end  %  VarCircle
```

Priloga 8.3: Kasa algoritem za krožnice

```
function Par = Kasa(XY)

%-------------------------------------------------------------------------
%
%      Preprosta algebraicna metoda za dodeljevanje kroznic (Kasa)
%       I. Kasa, "A curve fitting procedure and its error analysis",
%       IEEE Trans. Inst. Meas., Vol. 25, pages 8-14, (1976)
%
%      Vhodni podatki:  XY(n,2) je mnozica tock n koordinat x(i)=XY(i,1), y(i)=XY(i,2)
%
%      Izhodni podatki: Par = [a b R] je dodeljena kroznica:
%                             sredisce (a,b) in radij R
%
%-------------------------------------------------------------------------

n = size(XY,1);       % stevilo tock
Z = XY(:,1).*XY(:,1) + XY(:,2).*XY(:,2);

XY1 = [XY ones(n,1)];
P = XY1 \ Z;

Par = [P(1)/2 , P(2)/2 , sqrt((P(1)*P(1)+P(2)*P(2))/4+P(3))];

end   %  Kasa
```

Priloga 8.4: Prattov algoritem za krožnice

```
function Par = PrattSVD(XY)

%--------------------------------------------------------------------------
%
%     Pratt algebraicna metoda za dodeljevanje kroznic
%      V. Pratt, "Direct least-squares fitting of algebraic surfaces",
%      Computer Graphics, Vol. 21, pages 145-152 (1987)
%
%     Vhodni podatki:  XY(n,2) je mnozica tock n koordinat x(i)=XY(i,1), y(i)=XY(i,2)
%
%     Izhodni podatki: Par = [a b R] je dodeljena kroznica:
%                           sredisce (a,b) in radij R
%
%--------------------------------------------------------------------------

centroid = mean(XY);   % center vhodnih podatkov

[U,S,V]=svd([(XY(:,1)-centroid(1)).^2+(XY(:,2)-centroid(2)).^2,...
        XY(:,1)-centroid(1), XY(:,2)-centroid(2), ones(size(XY,1),1)],0);

if (S(4,4)/S(1,1) < 1e-12)   % ce singularna
    A = V(:,4);
    disp('Pratt singular case');
else                         % normalen primer
    W=V*S;
    Binv = [0 0 0 -0.5; 0 1 0 0; 0 0 1 0; -0.5 0 0 0];
    [E,D] = eig(W'*Binv*W);
    [Dsort,ID] = sort(diag(D));
    A = E(:,ID(2));
    for i=1:4
        S(i,i)=1/S(i,i);
    end
    A = V*S*A;
end

Par = -(A(2:3))'/A(1)/2 + centroid;
Par = [Par , sqrt(A(2)^2+A(3)^2-4*A(1)*A(4))/abs(A(1))/2];

end   % PrattSVD
```

Priloga 8.5: Taubinov algoritem za krožnice

```
function Par = TaubinSVD(XY)

%------------------------------------------------------------------------
%
%     Taubin algebraicna metoda za dodeljevanje kroznic
%      G. Taubin, "Estimation Of Planar Curves, Surfaces And Nonplanar
%                  Space Curves Defined By Implicit Equations, With
%                  Applications To Edge And Range Image Segmentation",
%      IEEE Trans. PAMI, Vol. 13, pages 1115-1138, (1991)
%
%     Vhodni podatki:  XY(n,2) je mnozica tock n koordinat x(i)=XY(i,1), y(i)=XY(i,2)
%
%     Izhodni podatki: Par = [a b R] je dodeljena kroznica:
%                           sredisce (a,b) in radij R
%
%------------------------------------------------------------------------

centroid = mean(XY);   % center vhodnih podatkov

X = XY(:,1) - centroid(1);  %  centriranje podatkov
Y = XY(:,2) - centroid(2);  %  centriranje podatkov
Z = X.*X + Y.*Y;
Zmean = mean(Z);
Z0 = (Z-Zmean)/(2*sqrt(Zmean));
ZXY = [Z0 X Y];
[U,S,V]=svd(ZXY,0);
A = V(:,3);
A(1) = A(1)/(2*sqrt(Zmean));
A = [A ; -Zmean*A(1)];
Par = [-(A(2:3))'/A(1)/2+centroid , sqrt(A(2)*A(2)+A(3)*A(3)-4*A(1)*A(4))/abs(A(1))/2];

end   %  TaubinSVD
```

Literatura

[1] The computer vision homepage. https://www.cs.cmu.edu/~cil/vision.html. Obiskano: 07-2014.

[2] Computer vision online. http://homepages.inf.ed.ac.uk/rbf/CVonline/#wikihierarchy. Obiskano: 07-2014.

[3] N. Chernov. *Circular and linear regression: Fitting circles and lines by least squares*. Chapman and Hall/CRC Monographs on Statistics and Applied Probability (Book 117). CRC Press; 1 edition, 2004.

[4] N. Chernov and C. Lesort. Least squares fitting of circles. *Journal of Mathematical Imaging and Vision*, 23(3):239–252, 2005.

[5] N.I. Chernov and G.A. Ososkov. Effective algorithms for circle fitting. *Computer Physics Communications*, 33(4):329 – 333, 1984.

[6] Nikolai Chernov. Matlab code for circle fitting algorithms. http://people.cas.uab.edu/~mosya/cl/MATLABcircle.html. Obiskano: 07-2014.

[7] E.R. Davies. *Machine Vision: Theory, Algorithms, Practicalities*. Signal Processing and its Applications. Elsevier Science, 2004.

[8] Charles R. Johnson and Susana Furtado. A generalization of sylvester's law of inertia. *Linear Algebra and its Applications*, 338(1–3):287 – 290, 2001.

[9] MathWorks. Computer vision system toolbox. http://www.mathworks.com/products/computer-vision/. Obiskano: 07-2014.

[10] Peter Meer, Doron Mintz, Azriel Rosenfeld, and DongYoon Kim. Robust regression methods for computer vision: A review. *International Journal of Computer Vision*, 6(1), 1991.