Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Vojko Drev

Prenosni merilnik svetlobnih lastnosti žarometov

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika 1

Mentor: doc. dr. Peter Peer

Ljubljana 2012

Namesto te strani **vstavite** original izdane teme diplomskega dela s podpisom mentorja in dekana ter žigom fakultete, ki ga diplomant dvigne v študentskem referatu, preden odda izdelek v vezavo! Glej tudi sam konec Poglavja **??** na strani **??**.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Vojko Drev, z vpisno številko **63050026**, sem avtor diplomskega dela z naslovom:

Prenosni merilnik svetlobnih lastnosti žarometov

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Petra Peer
- 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 v zbirki "Dela FRI".

V Ljubljani, dne 15. april 2012 Podpis avtorja:

Rad bi se zahvalil mentorju doc. dr. Petru Peer, za pomoč in svetovanje pri izdelavi diplomske naloge.
Zahvaljujem se tudi mag. Janku Kernc in podjetju Hella Saturnus Slovenija d.o.o., ki sta mi omogočila material, izvedbo in testiranje diplomskega dela, programa Prenosni merilnik svetlobnih lastnosti žarometov.
Hvaležen sem moji družini, prijateljem in vsem, ki so me podpirali in spodbujali pri študiju.

If I have seen further it is by standing on the shoulders of giants.

-Isaac Newton

Kazalo

Povzetek

Cilj diplomske naloge je razviti program za preverjanje svetilnih lastnosti avtomobilskih žarometov. Vhodni parameter programa je slika snopa svetlobe žarometa. Ta sveti na steno, kjer so narisane vodilne črte. Tri horizontalne in ena vertikalna. Žaromet se postavi tako, da sveti pravokotno na centralno vertikalno in horizontalno sliko.
Program iz slike izračuna prehod svetlo-temne meje po celotni širini slike. Uporabnik lahko nato izbere posamezno vertikalo. Njene intenzitete svetlobe se nato prikažejo na grafu.
Vmesnik je zasnovan na principu vtičnikov. Te se preberejo iz določene mape na disku in naložijo v program dinamično. Vsak v svoj zavihek.
Za komuniciranje med sabo uporabljajo model Publisher-Subscriber. Preko njega se prenašajo slike med posameznimi koraki in drugi parametri.
Vhodna slika se najprej pretvori v črno belo sliko. Nato se nad črno-belo sliko uporabi Sobelov filter, da se iz nje izlušči vodilne črte. Na tej sliki se izvede še Hough-ova črtna transformacija, kjer se pridobi natančne koordinate črt. Te se kasneje uporabi za izračun mer na sliki v kotih in centimetrih glede na odmik od centralne vertikalne in horizontalne črte.
Za vsako vertikalo na sliki se izračunajo intenzitete točk. Ugotovi se svetlo-temna meja. Ta je enaka prevojni točki na funkciji intenzitet točk, tam kjer je drugi odvod funkcije enak nič. Iz drugega odvoda se izračuna še začetek (maksimum) in konec (minimum) svetlo-temne meje.
Te tri črte se nariše na sliko in se izračuna širina svetlo-temne meje. To se na koncu prikaže na grafu.

Program je napisan kot alternativa fotometru. To je stroj, ki s pomočjo fotocelice in mehanskega premikanja žarometa ugotavlja svetlo-temno mejo. Rezultati programa so primerljivi z njim. So pa dosti odvisni od kvalitete in osvetlitve slike.
Program deluje hitro. Svetlo-temno mejo najde po celotni širini slike na intervalu kot 0,25° v dveh sekundah.

Abstract

The goal of this thesis is to develop a program for measuring lightness properties of car lights. Input parameter of this program is a picture of a car light's beam. Beam is projected on a wall where guidelines are. Three horizontal and one vertical.
Program calculates from the picture the cut-off line of the beam. This is a transition line from dark to light. User can select a vertical on a picture to see it's light intensities on a chart.
User interface is composed of a set of plug-ins. They are read from a certain directory and loaded into a program dynamically. Each gets it's own tab. They use a Publisher-Subscriber model to communicate with each other. Through it images and other parameters are passed between steps in a procedure.
Input picture is first transformed into a black and white picture. Then a Sobel's filter and Hough's line transformation are used to extract the guidelines. Guidelines are later used to help calculate real distances on the picture in angles and centimetres from the central guideline.
For each vertical on a picture light intensities of pixels are calculated. Then the light cut-off point is calculated which is equal to the inflection point on a curve of light intensities. This is where second derivative of this curve equals zero. Then the beginning and the end of the cut-off line are calculated (maximum and minimum of second derivative). At the end width of the cut-off line is calculated from this points and shown on a chart.
Program was written as an alternative to a machine called Photometer which

with a help of a light detector and mechanical moving of a car light calculates the cut-off line.
Results of our program are on pair with a Photometer but they are dependant on a quality and lightness of an input picture.

Poglavje 1

Uvod

Avtomobilska industrija je zelo zahtevno področje. Dosti konkurence. Dosti povpraševanja. Visoki standardi za varnost in za kvaliteto. Avtomobili se proizvajajo kot na tekočem traku. Vsak upad proizvodnje je drag. Izjemno drag. Tukaj govorimo o več 1000 in celo več 10000 eurih na minuto. Vsak izdelek, ki pride v tovarno in se montira na avtomobile, tovornjake, motorje... mora biti praktično neoporečen. Dovoljenih je le par slabih kosov na celo pošiljko. Zato je potrebno veliko preverjanje kakovosti raznih delov avtomobila in potrebna so orodja, ki nam to omogočajo.

V tej diplomski nalogi se bomo osredotočili na avtomobilske žaromete. Ocenjevali bomo njihovo kakovost. Ne v smislu ta žaromet je zanič ali ta je dober. Ampak bomo gledali njihove lastnosti. Bolj specifično lastnosti njihovega snopa svetlobe.

Zanimala nas bo svetlo-temna meja. Njen prehod, oblika in širina. Tehnologije za določanje teh karakteristik že obstajajo. Ena izmed takih je fotometer. Naprava ima veliko pomanjkljivosti. Je velika in mehanska. Zanjo potrebuješ posebej pripravljeno sobo, da zmanjšaš količino napake. Potrebuješ posebne stroje, ki žaromet premikajo. Na koncu pa lahko preverjaš samo en žaromet naenkrat. Poleg tega je pa vsak del naprave potrebno vzdrževati.

Cilj te diplomske naloge je, da razvijemo alternativo te napravi. Ta alternativa bo v obliki računalniškega programa, ki bo obdeloval sliko snopa sve-

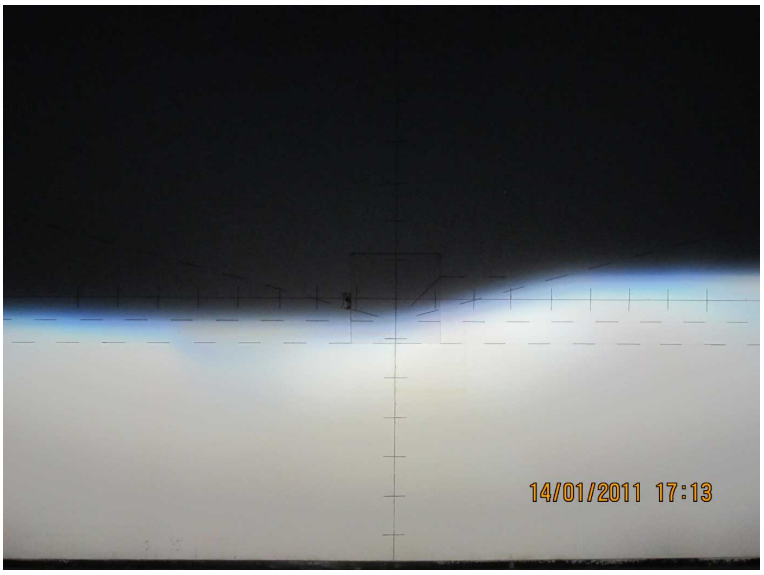
tlobe žarometa. Soba bo seveda še vedno potrebna. Manj pa jo bo potrebno zaščititi pred motnjami. Namesto fotometra pa potrebujemo samo fotoaparat ali kamero. Naš program mora tako omogočati vsaj isto funkcionalnost. Mora biti hitrejši in imeti isto natančnost rezultatov.

Diplomska naloga se začne s poglavjem, kjer opišem naš problem in cilje. Opiše se postopek pridobitve slike in prikazana je vhodna slika.
Nato sledi pregled vseh obstoječih rešitev. Sem spada opis fotometra in postopka kako z njim izmerimo svetilne lastnosti žarometa.
Naslednje poglavje opiše razvoj naše aplikacije. Vsebuje zahteve strojne opreme, opis programskih jezikov, ki so na voljo. Nato je opisan uporabniški vmesnik. Na koncu pa celoten postopek obdelave slike in računanje svetlotemne meje.
Sledi poglavje o optimizaciji uporabljenih algoritmov. Opisan je postopek optimizacije. Podani so časi hitrosti algoritmov pred in po optimizaciji.
Na koncu je dodana še primerjava rezultatov našega programa in fotometra.

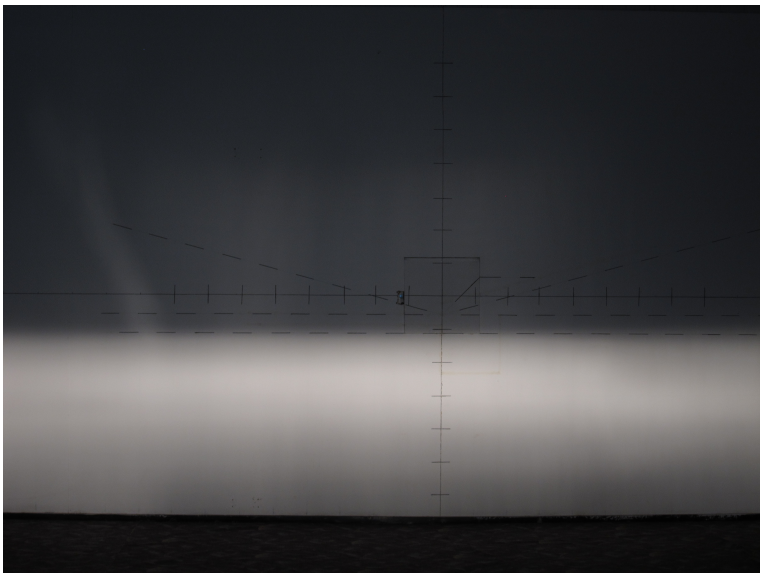
Poglavje 2

Opis problema

Potrebno je narediti program, ki ugotavlja svetlobne lastnosti žarometov. Te lastnosti se potem upoštevajo pri oceni kakovosti žarometa in njegove izdelave. Žarometi so lahko sprednji, s kratkimi ali dolgimi lučmi, zadnji ali pa žarometi za meglenke. Z žarometom se posije na steno, kjer so narisane vodilne črte. Te vsebujejo dve centralni črti (horizontalno in vertikalno). In več drugih pomožnih črtkanih črt. Centralni črti sta vsaki na kotih sijanja nič stopinj in žaromet mora nanju sijati pravokotno. Pri sprednjih žarometih, se tisto stran, ki sije višje nastavi na prvo horizontalno črtkano črto. Pri meglenkah, kjer je cel snop v isti liniji, se žaromet nastavi na zadnjo horizontalno črtkano črto. S programom je potrebno najti prehod svetlo-temne meje v stopinjah in centimetrih, oddaljenih glede na horizontalno centralno črto. Potrebno je izračunati širino svetlo-temne meje. Sliki 2.1 in 2.2 sta primera vhodnih parameterov.



Slika 2.1: Vhodni parameter, sprednji žaromet.



Slika 2.2: Vhodni parameter, meglenka.

Poglavje 3

Sorodne rešitve

3.1 Fotometer

Fotometer je naprava, ki jo v podjetju Hella Saturnus d.o.o. uporabljajo za določanje svetlobnih lastnosti žarometov. Žaromet se privije na optično os. Ta je nasproti fotocelici na razdalji petindvajset metrov. Za svetilke se uporablja razdalja treh metrov.
Žarnice, ki se pri testih uporabljajo so različne od tistih v proizvodnji žarometov. Nitka v žarnici mora biti določene dolžine in je ne sme presegati za več kot milimeter. Daljša nitka lahko premakne svetlo-temno mejo.
Pri merjenju fotocelica stoji na miru. Premika se samo žaromet in sicer v vertikalni smeri. Na vsake 0,01 stopinje se pomeri svetilnost. Tako se ugotovi svetlo-temno mejo. Nahaja se kjer je največji gradient/razlika svetlobe med dvema točkama. Na razdalji 0,01 stopinje svetloba naraste za 20%, pri kseon žarnicah tudi do 200%.
Potem se preveri bleščanje žarometa, kjer se žaromet obrne tako da ne sveti v zaslon. Preveri pa se moč svetlobe (pod različnimi koti), ki pada na fotocelico.
Standardi svetilnosti žarometov so od države do države različni.

Poglavje 4

Razvoj lastne aplikacije

4.1 Določitev strojne opreme

Zahteve za delovanje programa:

- Prenosnik ali osebni računalnik.
- Operacijski sistem Windows XP, Vista ali 7.
- Naložen .NET framework verzije 4.0 ali več.
- Intel ali Athlon procesor. 32 ali 64 biten. Priporočeno, da je več jedrni.
- Priporočena količina pomnilnika je 2GB.
- Velikost diska mora biti dovolj velika za operacijski sistem, .NET framework in slike svetlo-temnih mej. Sam program je velik par MB.
- Fotoaparat ali kamera (v načinu za slikanje ne snemanje).
- Priporočena nastavitev fotoaparata za ISO je 100, slikanje s števcem ali stojalom.

4.2 Uporabljene tehnologije

Ena izmed zahtev naročnika programa je bila, da teče na operacijskem sistemu Windows. Pregledal sem več programskih jezikov, ki so na voljo za ta sistem. Moji glavni kriteriji so bili, da je programski jezik objektno usmerjen, da se v njem enostavno zgradi uporabniški vmesnik, in da zanj obstaja integrirano razvojno okolje. Pregledal sem jezike: Java, C# .Net, Python, Delphi in PHP. Vsi jeziki so objektno usmerjeni, nekateri imajo dobro podporo za računalniški vid, nekateri nimajo avtomatičnega upravljanja s pomnilnikom... Spodaj so našteti dodatni plusi in minusi, ki sem jih ugotovil.

4.2.1 Java

Plusi

- Sam skrbi za čiščenje pomnilnika.
- Gradnja vmesnika v okolju Netbeans.
- Knjižnica OpenCV za računalniški vid.

Minusi

- Ne vsebuje standardnega uporabniškega vmesnika Windows.

4.2.2 C#

Plusi

- Sam skrbi za čiščenje pomnilnika.
- Gradnja vmesnika v okolju Visual Studio.
- Knjižnica Aforge za računalniški vid.
- Knjižnice za paralelno procesiranje.
- Knjižnica LINQ za enostavno delo s seznami in ostalimi podatkovnimi strukturami.

4.2.3 Delphi

Plusi

- Standarden Windows uporabniški vmesnik.

Minusi

- Slaba podpora za računalniški vid.
- Nima avtomatičnega čiščenja pomnilnika.

4.2.4 Python

Plusi

- Sam skrbi za čiščenje pomnilnika.
- Knjižnica OpenCV za računalniški vid.

Minusi

- Slaba podpora za grafični uporabniški vmesnik (starejše verzije GTK+ in QT knjižnic za operacijski sistem Windows kot za Linux).

4.2.5 PHP

Plusi

- HTML vmesnik, ki zgleda isto v večini operacijskih sistemov.

Minusi

- Aplikacija bi morala biti v dveh delih. Uporabniški vmesnik, ki je napisan v PHP-ju in delu, ki bi procesiral slike.
- Nima podpore za niti/paralelno procesiranje.

- Slaba podpora za računalniški vid.

Moja končna izbira je bil C#. Veliko zaradi osebnih preferenc. V njem sem najbolj domač. Visual Studio je zelo dobro integrirano okolje, ki omogoča hiter razvoj aplikacij. V njem se enostavno zgradi uporabniški vmesnik. V njem lahko načrtuješ razrede in funkcije. Zanj se da dobiti veliko dodatkov, ki ti olajšajo pisanje kode. Uporabil sem knjižnico za računalniški vid Aforge .Net, predvsem zaradi hitrega delovanja in že implementiranih algoritmov za Sobelov in Hough-jev filter. Za hranjenje kode sem uporabil sistem GIT, ki beleži zgodovino sprememb. Ima dobro podporo za vejanje in združevanje kode. Je enostaven in hiter za uporabo.

4.3 Zasnova uporabniškega vmesnika

Uporabniški vmesnik je razdeljen v osem glavnih sklopov. Te so originalna slika, črno-bela slika, Sobelov filter, prag, vodilne črte, vertikalne jakosti, svetlo-temna meja in širina svetlo-temne meje. Vsaka izmed teh je dosegljiva preko zavihkov (slika 4.1).

4.3.1 Vtičniki

Vmesnik je zasnovan na principu vtičnikov [5, 4]. Vsak zavihek je svoj vtičnik in vsak ima svoj uporabniški vmesnik za konfiguracijo nastavitev. Definirani so s pomočjo vmesnika, ki zgleda tako:

```
interface IImagePlugin {
    UserControl UserInterface { get; }
    string Name { get; }
    bool ValidateInput();
    void ProcessImage();
}
```

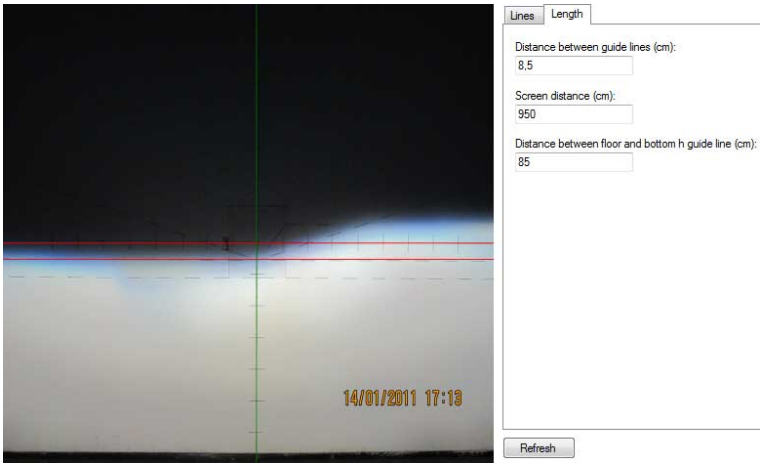
Slika 4.1: Uporabniški vmesnik programa.

Lastnost UserInterface vrne uporabniški vmesnik vtičnika. Ta je lahko standarden s sliko na levi strani in konfiguracijskimi polji in gumbi na desni strani (slika 4.2). Lahko pa vsebuje samo graf (slika 4.3).

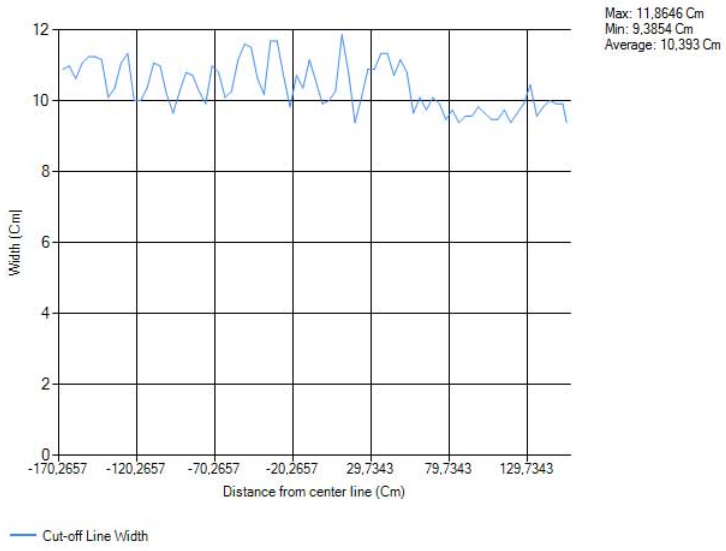
Funkcija ValidateInput se kliče, ko uporabnik pritisne gumb Refresh/Osveži. Ta funkcija preveri vse vrednosti v poljih vmesnika, če imajo pravilno vsebino (če je podana širina število, da ni preveliko ali premajhno...). Funkcija ProcessImage se kliče po funkciji ValidateInput. Namenjena je procesiranju in obdelavi slike. Pri vtičniku črno-bela slika ta funkcija spremeni originalno sliko v črno-belo. V vtičniku vodilne črte pa ta funkcija najde vertikalno in dve horizontalni vodilni črti na sliki. Lastnost Name je ime vtičnika, ki je prikazana v imenu zavihka.

4.3.2 Nalaganje vtičnikov

Vtičniki niso del glavnega programa. Vsak vtičnik se nahaja v svoji knjižnici, ki se ob zagonu programa dinamično naloži. V programu je določena mapa vtičnikov. Iz te mape se preko knjižnice .Net za Reflection [5] naložijo vse knjižnice s vtičniki. V vsaki knjižnici se poišče implementacijo vmesnika



Slika 4.2: Primer uporabniškega vmesnika za iskanje vodilnih črt.



Slika 4.3: Primer uporabniškega vmesnika z grafom.

IImagePlugin. Naredi se objekt, iz katerega se prebere uporabniški vmesnik. Tega se doda v nov zavihek z imenom našega vtičnika. Na koncu se poveže še OnClick dogodek refresh gumba z metodama Validate in ProcessImage. V psevdokodi to zgleda tako:

```
foreach (dll in files_from_dir(plugins))
    foreach (class in dll.get_classes())
        if (class.implements(IImagePlugin))
            o = createObject(class)
            t = new Tab(o.Name)
            t.refreshButton.onclick += (sender, e) => {
                if (o.Validate)
                    o.ProcessImage()
            }
            t.controls.add(o.UserInterface)
            tabs.add(t)
```

4.3.3 Publisher-Subscriber model

Vtičniki morajo nekako komunicirati med sabo. Slabo pa je, če vedo eden za drugega. Pravilo je, da so čim bolj neodvisni. Za tak primer prav pride vzorec po imenu Publisher-Subscriber [5, 4]. Deluje na principu dogodkov. Pri tem modelu nekdo pošilja sporočila. Ta lahko vsebujejo slike, mere, velikosti točk... Vsi, ki so prijavljeni na določeno sporočilo, ga nato prejmejo in obdelajo. Vsebuje dve funkciji:

```
class PubSub {
    void Publish<T>(T message);
    void Subscribe<T>(Action<T> callback);
}
```

Subscribe funkcija sprejme kot tip sporočila parameter, na katerega se prijavljamo. Callback parameter je pa referenca na funkcijo, ki se pokliče,

ko nekdo objavi sporočilo s tem tipom. Publish metoda se kliče, ko nekdo hoče objaviti sporočilo tipa T.
Recimo, da vtičnik za originalno sliko naloži sliko v procesiranje. Ta potem kliče metodo Publish z objektom tipa OriginalPictureLoadedMessage. Vtičniku za pretvorbo v črno-belo sliko, ki je prijavljen na to sporočilo se pokliče metoda callback. Preko parametra sprejme sliko iz prvega vtičnika in jo pretvori. Po tem ponovno kliče metodo publish s sporočilom tipa BlackAndWhiteImageMessage. Tega nato prejme naslednji vtičnik in tako naprej.

4.4 Postopek

4.4.1 Črno-bela slika

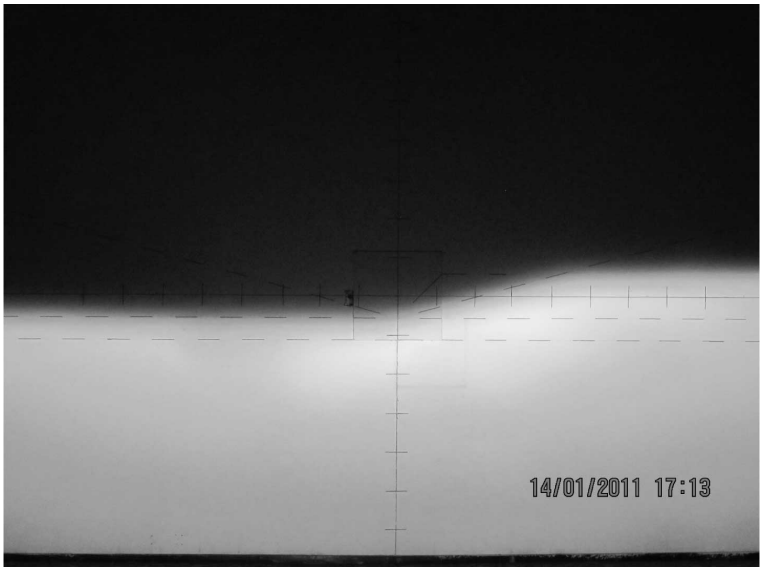
Prvi korak v postopku je pretvorba slike v črno belo. To storimo tako, da za vsako piko na sliki izračunamo novo vrednost po formuli 4.1 [11].

$$L = \frac{1}{2}(M + m) \tag{4.1}$$

Kjer je M največja vrednost izmed RGB [10] komponent točke. Vsaka točka je določena s tremi vrednostmi. Vsaka vrednost določa količino barve. R je za rdečo, G za zeleno in B za modro barvo. Te vrednosti so lahko od 0 do 255. Na tak način lahko opišemo barv. Spremenljivka m je najmanjša vrednost izmed RGB komponent. L je nova vrednost točke za črno-belo sliko. Algoritem v psevdokodi zgleda tako:

```
for (int i = 0; i < visina(slika); i++)
    for (int j = 0; i < sirina(slika); j++)
    tocka = slika[i][j]
    rezultat[i][j].R, G, B = (max_komponenta(tocka)
        + min_komponenta(tocka)) / 2
```

Rezultat tega koraka je slika 4.4.



Slika 4.4: Črno-bela slika.

4.4.2 Sobelov filter

Drugi korak v našem postopku je uporaba Sobelovega filtra [7]. Ta nam pomaga kasneje pri odkrivanju vodilnih črt na sliki. Tukaj je bila uporabljena knjižnica za računalniški vid Aforge.NET [1], kjer je ta filter že implementiran. Potreben je klic funkcije:

```
sobel = new SobelEdgeDetector().Apply(crnobelaslika);
```

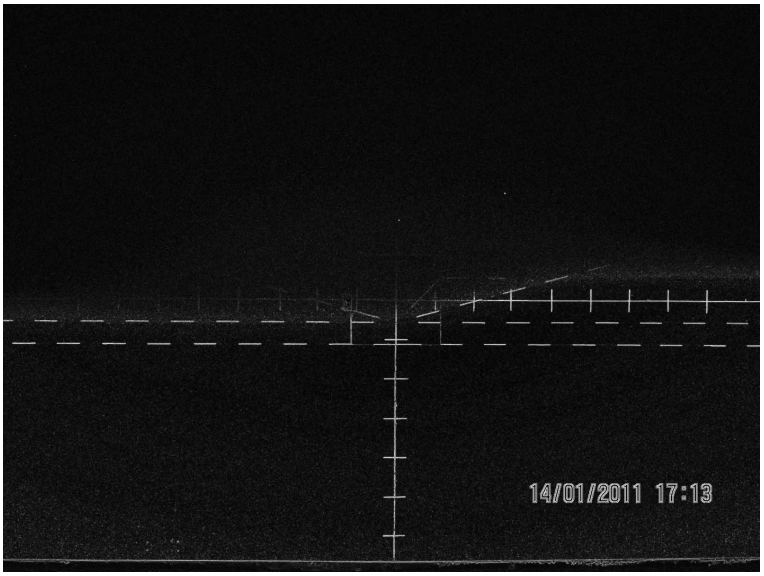
Rezultat filtra je slika 4.5.

4.4.3 Prag

S tretjim korakom počistimo sliko iz poglavja 4.4.2 z algoritmom za določanje pragu [8]. To storimo tako, da za vsako točko pogledamo njeno intenziteto [1] Intenziteta je enaka vrednosti ene izmed RGB komponent. Če je intenziteta večja od vrednosti, ki si jo uporabnik izbere [2], potem na novo sliko

[1]Pri pretvorbi v črno belo sliko imajo vse tri komponente točke isto vrednost.

[2]Intenzitete imajo lahko vrednost od 0 do 255.



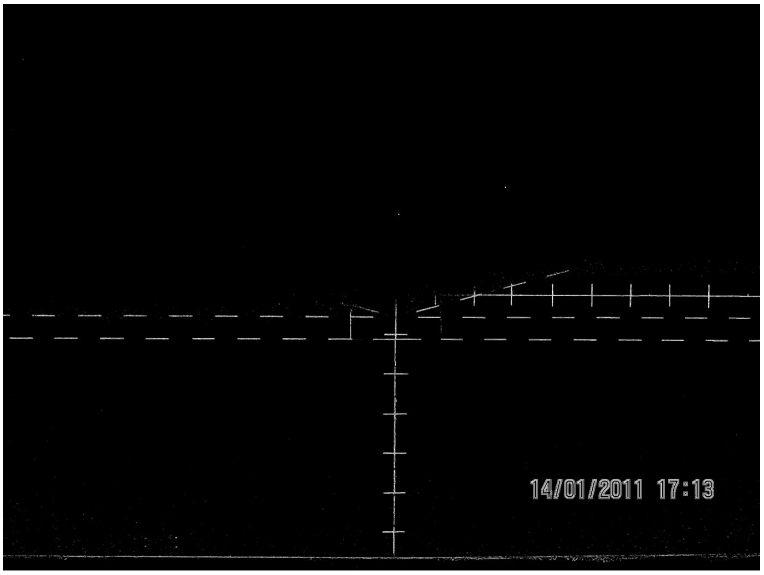
Slika 4.5: Sobelov filter.

vpišemo točko z maksimalno intenziteto. Če je ta vrednost manjša od izbrane, vpišemo točko z minimalno intenziteto [3]. Tako na novi sliki še bolj poudarimo vodilne črte. In zmanjšamo količino šuma (točke, ki niso vodilne črte in ki bi ovirale njihovo detekcijo). Ta algoritem v psevdokodi zgleda tako:

```
for (int i = 0; i < visina(slika); i++)
    for (int j = 0; i < sirina(slika); j++)
        if (slika[i][j].R < prag)
            rezultat[i][j].R, G, B = 0
        else
            rezultat[i][j].R, G, B = 255
```

Algoritem kot vhodni parameter sprejme bitno sliko in vrednost pragu. Kot rezultat pa vrne sliko 4.6.

[3]Bela točka ima intenziteto 255, črna pa intenziteto 0.



Slika 4.6: Prag.

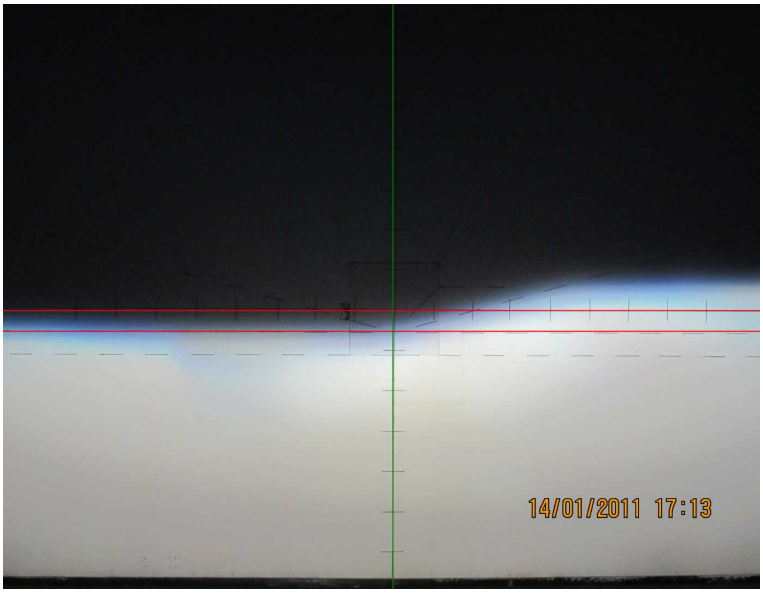
4.4.4 Iskanje vodilnih črt

V tem koraku iščemo prvi dve horizontalni vodilni črti (od zgoraj navzdol). In sredinsko vertikalno vodilno črto. Tukaj je uporabljena Hough-jeva črtna transformacija [6], ki se uporablja za detekcijo črt na sliki. Tudi ta je že implementirana v Aforge.NET knjižnici [2]. Primer klica za Hough-jevo transformacijo je:

```
HoughLineTransformation hlt = new HoughLineTransformation();
hlt.ProcessImage(sobel);
lines = hlt.GetLinesByRelativeIntensity(relativeIntensity);
IEnumerable<HoughLine> verticalLines =
lines.Where(l => -5 <= l.Theta && l.Theta <= 5);
```

Algoritem sprejme sliko pragu Sobelovega filtra. Vrne nam seznam črt, ki imajo večjo relativno jakost od parametra relativeIntensity [2] in kot med -5 in 5 stopinjami [4]. Relativna jakost črte je odvisna od števila belih pik, ki

[4]S tem kotnim pogojem najdemo vertikalne črte, za horizontalne črte uporabimo kote od 85° do 95°



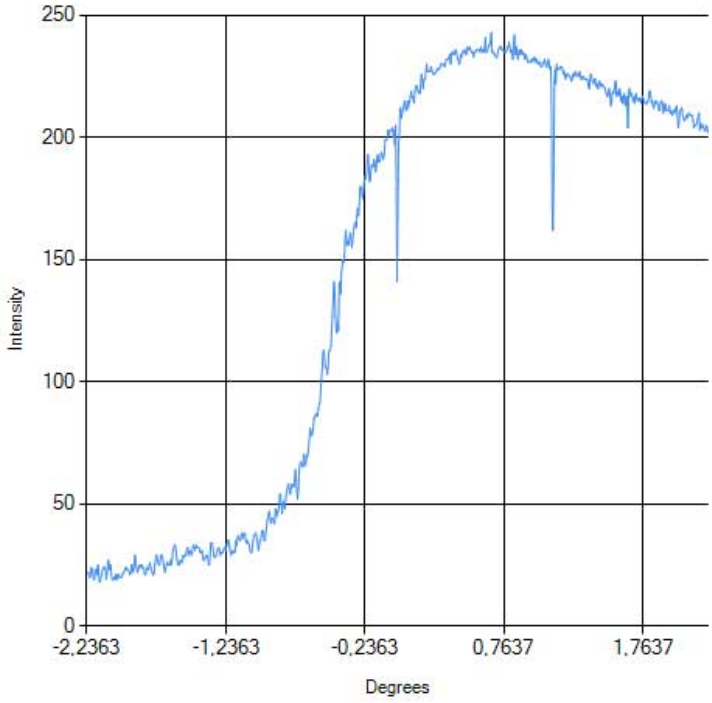
Slika 4.7: Vodilne črte.

ležijo na črti. Vsaka črta je predstavljena z oddaljenostjo od centralne točke na sliki in kotom naklona. Tako je črta na oddaljenosti 0 in kotom 90 zelo blizu naše iskane prve horizontalne črte. Črta z oddaljenostjo 0 in kotom 0 stopinj pa blizu naše iskane vertikalne vodilne črte [2].

Te črte (slika 4.7) se nato uporabijo za določitev mer na sliki (v kotih in centimetrih). Uporabnik vnese razdaljo med horizontalnima črtama in razdaljo od zaslona do kamere (v centimetrih). Iz tega lahko izračunamo koliko točk na sliki je en centimeter. Formula za izračun je formula 4.2.

$$v = \frac{a}{|r_1 - r_2|} \tag{4.2}$$

Kjer je v velikost točke v cm. Spremenljivka a je razdalja v cm med horizontalnima vodilnima črtama. R1 in r2 sta pa razdalji od centralne točke na sliki (v točkah). Vsako točko na sliki lahko podamo tudi v kotu glede na centralno vertikalno ali horizontalno vodilno črto. Formula za izračun kota je formula 4.3.



Slika 4.8: Intenzitete v stopinjah.

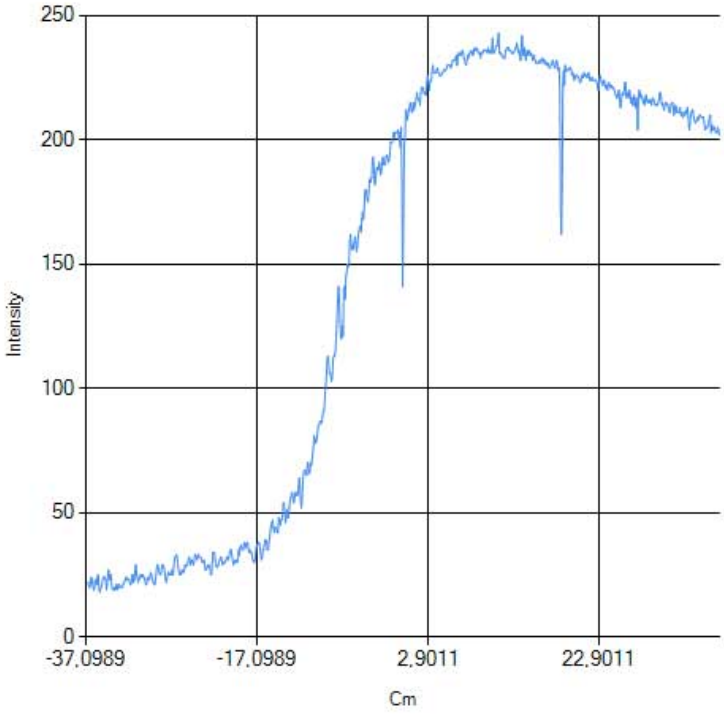
$$tan(\alpha) = \frac{a}{b} \tag{4.3}$$

A je oddaljenost, v cm, trenutne točke od centralne vodilne črte. B je oddaljenost zaslona od kamere. Alfa je kot, ki ga iščemo.

4.4.5 Pregled intenzitet po vertikali

Ta korak je glavni za določitev svetlo-temne meje. To storimo tako, da izberemo vertikalo na sliki in pogledamo intenzitete njenih točk (0 do višine naše slike). Za sliko uporabimo črno-belo sliko iz prvega koraka. Intenzitete nato izrišemo na grafu (slika 4.8 in slika 4.9).

Os x grafu iz slike 4.8 predstavlja oddaljenost v stopinjah (graf iz slike 4.9 v cm), od horizontalne vodilne črte. Na y osi je prikazana intenziteta



Slika 4.9: Intenzitete v centimetrih.

točke.
Primer algoritma v psevdokodi zgleda tako:

```
for (int i = 0; i < visina(slika); i++)
    vrednost = slika[i][x].R
    rezulat.dodaj(new XY(pretvori(i, enota), vrednost))
```

Algoritem sprejme kot vhodni parameter sliko svetlo-temne meje. Prebere iz trenutne vrstice, z indeksom i, vrednost (intenziteto) iz stolpca x (naša vertikala). Pretvori koordinate vrstice i v izbrano enoto (stopinje ali centimetri). Na koncu shrani intenziteto in koordinate vrstice v seznam parov XY.

4.4.6 Iskanje svetlo-temne meje

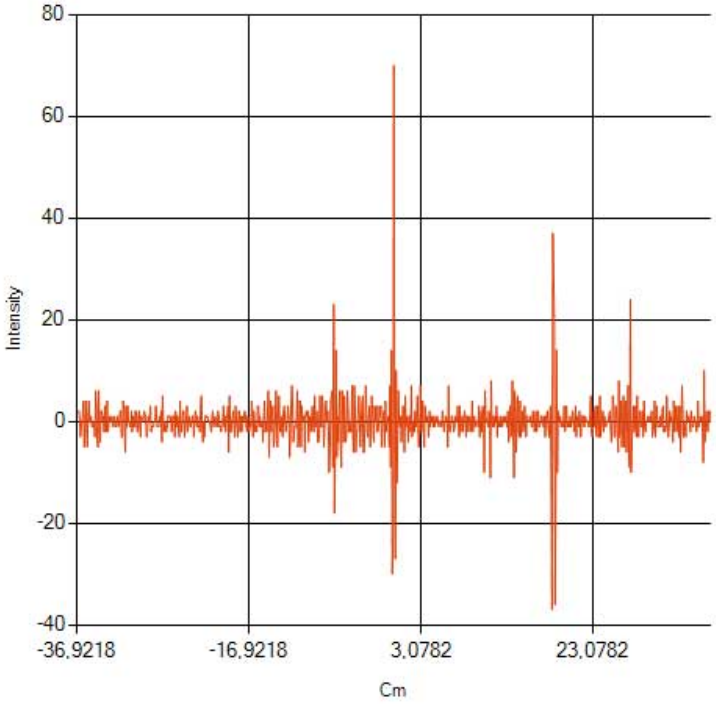
Za svetlo temno mejo vzamemo prevojno točko na grafu. Ta točka se nahaja kjer je drugi odvod funkcije naših intenzitet enak nič. Za izračun drugega odvoda uporabimo formulo 4.4 [9].

$$d2 = f(x+1) - 2f(x) + f(x-1) \tag{4.4}$$

Kjer je f(x+1) naslednja točka na funkciji, f(x) trenutna in f(x-1) prejšnja točka. Funkcijo predstavimo s seznamom parov (X in Y). Pari so urejeni naraščajoče po vrednosti Y. Algoritem za izračun sprejme kot parameter našo funkcijo (spremenljivka f spodaj), vrne pa funkcijo drugega odvoda (d). Začnemo na drugem elementu v našem seznamu (zato, ker formula potrebuje prejšnji element). Končamo pa na predzadnjem elementu (ker formula potrebuje zadnji element).

```
For (i = 1; i < dolzina(f) { 1; i++)
    d.dodaj(new XY(f[i + 1].X { 2 * f[i].X + f[i-1].X, f[i].Y))
```

Slika 4.10 prikazuje rezultat algoritma. Iz nje težko določimo katera ničla na grafu je prava. Originalna funkcija vsebuje veliko šuma. Dosti spreminja smer in tako vsebuje veliko prevojnih točk. Ta problem rešimo tako, da našo originalno funkcijo zgladimo z algoritmom za glajenje funkcij.



Slika 4.10: Drugi odvod funkcije.

4.4.7 Glajenje funkcije

V našem programu funkcijo gladimo tako, da jo povprečimo. Velikost okna[5] je nastavljiva s strani uporabnika. Obstaja več načinov kako povprečiti funkcijo. V povprečje lahko upoštevamo samo elemente pred trenutnim, lahko samo elemente za trenutnim. Najbolje se odnese, če vzamemo polovico velikosti okna elementov pred trenutnim in polovico elementov za trenutnim. Torej, če je naše okno veliko deset, začnemo s šestim elementom, k njemu prištejemo pet elementov pred njim in pet za njim. Število, ki ga dobimo delimo s enajst in to je naš rezultat za trenutni element. Nadaljujemo dokler nam elementov v naši originalni funkciji ne zmanjka.

```
a = dolzina(f) / 2
for (i = a; i < dolzina(f) { a; i++)
    sum = 0
    for (j = i { a; j < i + a; j++)
        sum += f[j].X
    r.dodaj(new XY(sum / (a * 2 + 1), f[i].Y))
```

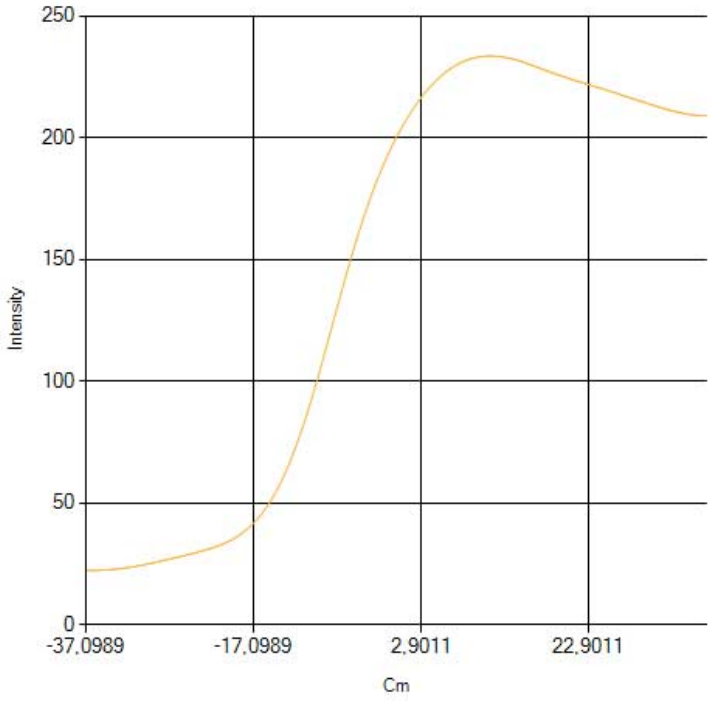
Spremenljivka r je rezultat in zglajena funkcija našega algoritma (slika 4.11 in slika 4.12). Slika 4.11 prikazuje našo zglajeno funkcijo. Slika 4.12 pa prikazuje primerjavo med originalno in zglajeno funkcijo.

Če sedaj to zglajeno funkcijo (slika 4.11) še enkrat odvajamo, dobimo lepšo funkcijo drugega odvoda (slika 4.13), kjer lahko določimo točko svetlo-temne meje.

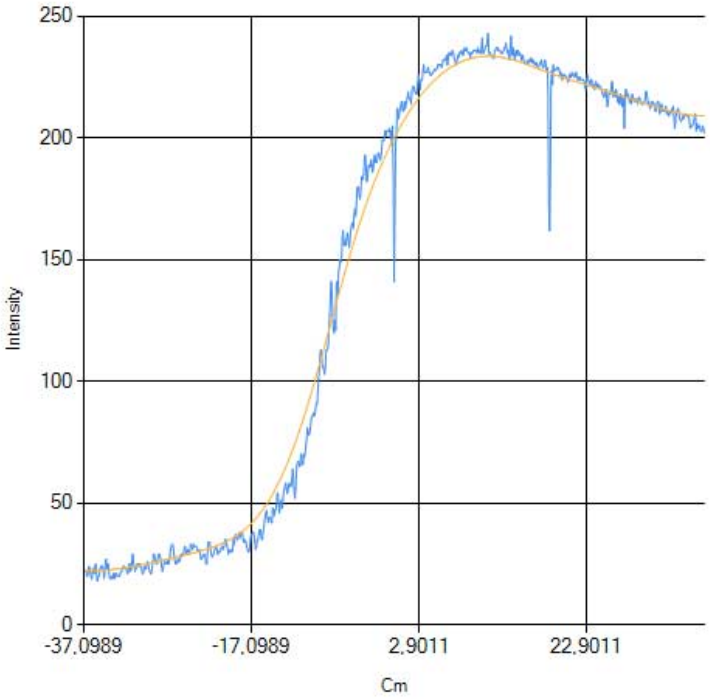
4.4.8 Določanje ničle/svetlo-temne meje

Ničla na drugem odvodu naše funkcije (slika 4.13) se poišče tako, da se najprej najde njegov maksimum. Potem se pa poišče dve zaporedni točki, kjer se jima spremeni predznak. Spodnji algoritem prikazuje ta postopek. Algoritem sprejme indeks maksimalnega elementa kot vhodni parameter in vrne indeks prvega elementa od dveh kjer se spremeni predznak.

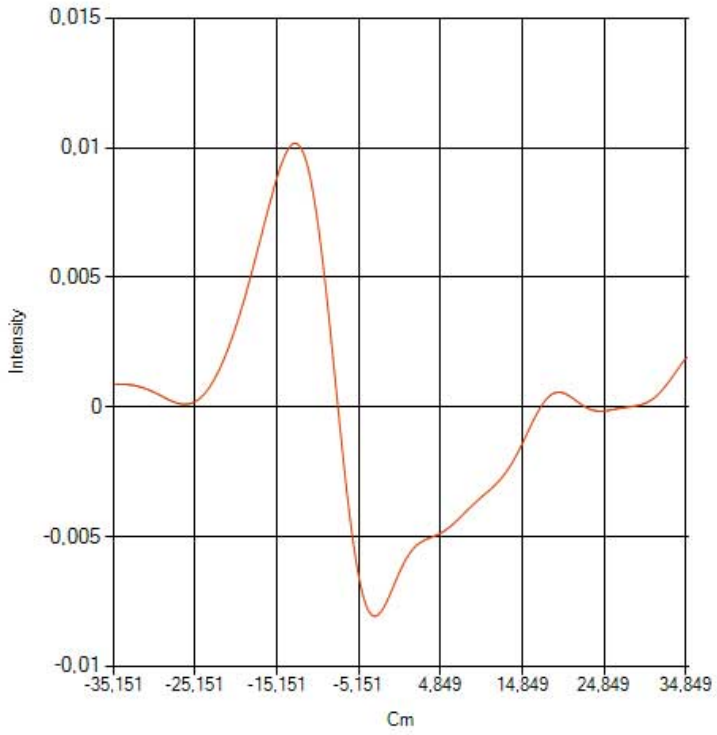
[5]Okno je število elementov, ki jih vzamemo v povprečje, ko računamo trenutni element



Slika 4.11: Zglajena funkcija.



Slika 4.12: Primerjava zglajene in originalne funkcije.



Slika 4.13: Drugi odvod zglajene funkcije.

```
For (i = s; i < dolzina(f); i++)
    if (f[i].X == 0) return i;
    if (f[i].X < 0) return i { 1;
```

Če je ničla med najdenima elementoma, potem se jo določi preko linearne interpolacije (formula 4.5).

$$\begin{aligned} k &= \frac{y_2 - y_1}{x_2 - x_1} \\ n &= y_1 - k * x_1 \\ z &= (-n/k) \end{aligned} \tag{4.5}$$

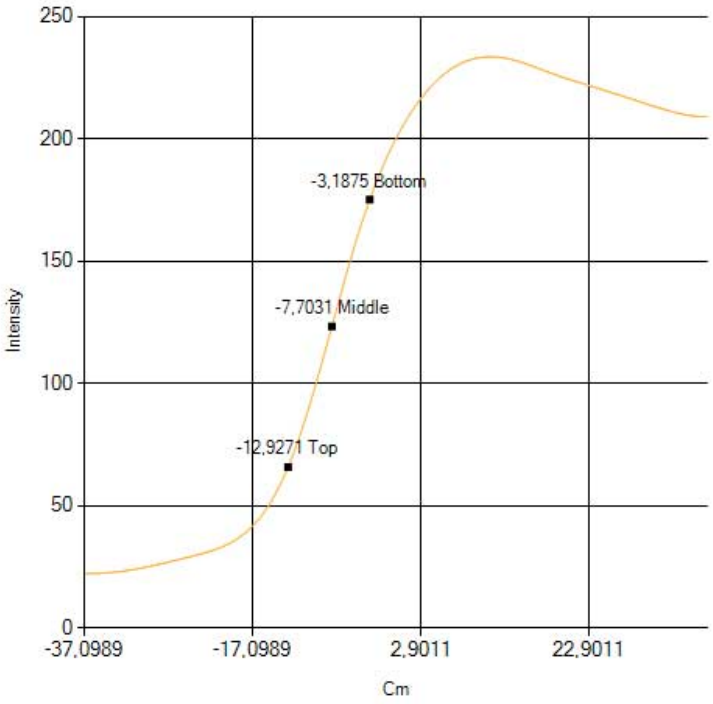
Z predstavlja x koordinato kjer se nahaja ničla na naši funkciji drugega odvoda in posledično točka svetlo-temne meje na naši originalni funkciji. To točko nato izrišemo na grafu (slika 4.14) in na naši originalni sliki (4.15).

Na sliki 4.14 je točka »Middle« izračunana svetlo-temna meja. Točki »top« and »bottom« sta pa maksimum in minimum funkcije drugega odvoda.

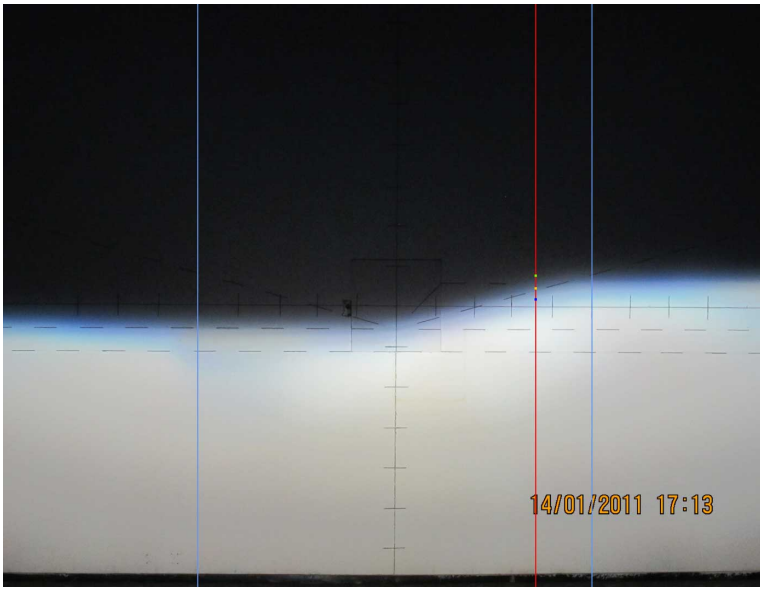
Rumena točka na sliki 4.15 predstavlja svetlo-temno mejo. Zelena začetek nje ali masksimum drugega odvoda funkcije. Modra pa konec svetlo-temne meje ali minimum drugega odvoda funkcije. Rdeča črta prikazuje trenutno izbrano vertikalo. Modri črti sta pa vodilni črti, izračunani z našim programom. Vsaka je postavljena na kotu petih stopinj glede na centralno vertikalno vodilno črto.

4.4.9 Iskanje svetlo-temne meje

Svetlo temno mejo najdemo tako, da zgornje algoritme ponovimo na celotni širini slike. Tukaj nam prav pride paralelno procesiranje. Procesiramo lahko več vertikal hkrati, vsako na svojem jedru procesorja. Microsoft je v .Net 4.0 dodal podporo za paralelno for zanko, ki nam tukaj zelo prav pride in olajša delo. Zanki podamo seznam elementov, ki jih hočemo obdelati. Ta pa



Slika 4.14: Začetna, končna in točka svetlo-temne meje na grafu.



Slika 4.15: Začetna, končna in točka svetlo temne meje na sliki.

jih nato razdeli na vsak procesorska jedra. Sintaksa je preprosta. Cel naš algoritem izgleda tako:

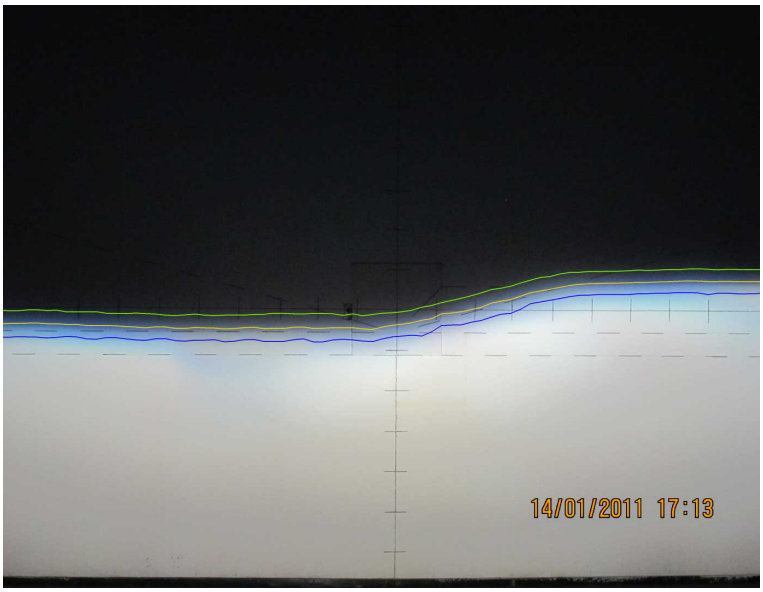
```
rezultat = []
Parallel.ForEach<int>(sirinaSlike, x =>
{
    rezultat.dodaj(najdiSvetloTemnoMejo(slika, x));
});
```

Algoritem nam najde vse točke po celotni širini slike za svetlo temno mejo. Vsa jedra procesorja so obremenjena. Več jeder ko imamo (ali boljši računalnik kot je) hitreje deluje algoritem.
Alternativna implementacija je lahko s v naprej kreiranimi nitmi in delavnimi razredi. Razredu se poda kot parameter sliko in koordinate vertikale. Na koncu pa preko Pub-Sub objekta sporoči rezultate. Lahko se uporabi tudi distribuirano procesiranje na več računalnikih, kjer se izvajajo delavni procesi. Te procesi sprejmejo iste parametre kot predhodni primer. Razlika je le v tem, da se parametri pošljejo preko vtičnic po lokalni mreži ali preko interneta. To implementacijo bi se dalo izvesti s prosto rešitvijo BeanstalkD [3], ki omogoča preprosto implementacijo vodje in delavnih procesov.
Na koncu nam preostane samo še, da te črte (algoritem vrne svetlo-temno mejo, začetek in konec nje) izrišemo na sliki (4.16).

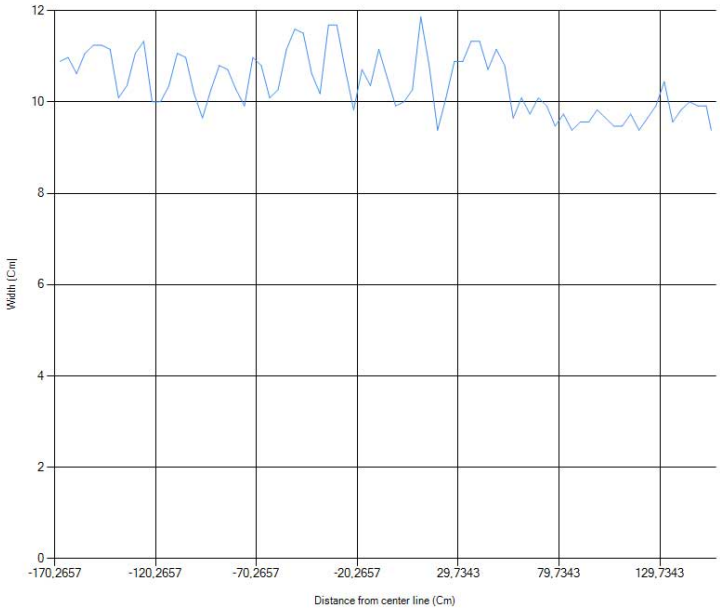
4.4.10 Graf širine svetlo-temne meje

Algoritem je sledeč. Za vsako izračunano vertikalo odštejemo višimo črte začetka svetlo-temne meje na sliki in višimo črte konca svetlo-temne meje na sliki. Te vrednosti nato prikažemo na grafu, glede na izbrano enoto (lahko v centimetrih ali kotih v stopinjah).

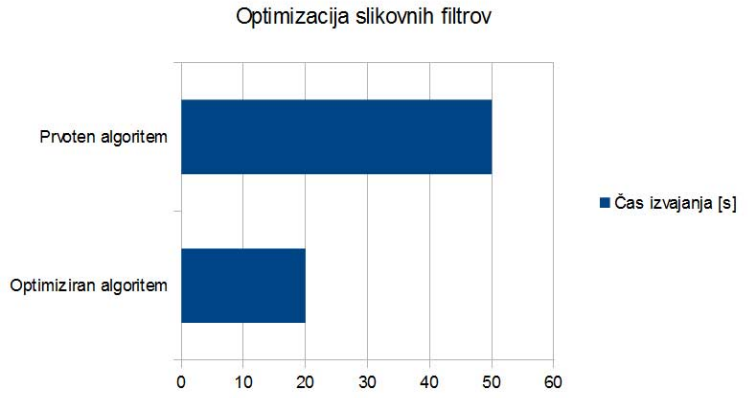
Na grafu slika 4.17 je na x osi prikazana razdalja v centimetrih od vertikalne vodilne črte. Na y osi pa širina svetlo-temne meje, tudi v centimetrih. V tem primeru širina sega od devet do dvanajst centimetrov.



Slika 4.16: Črte začetka, konca in prehoda svetlo-temne meje.



Slika 4.17: Graf širine svetlo-temne meje.



Slika 4.18: Vizualna predstavitev pospešitve algoritmov za slikovne filtre.

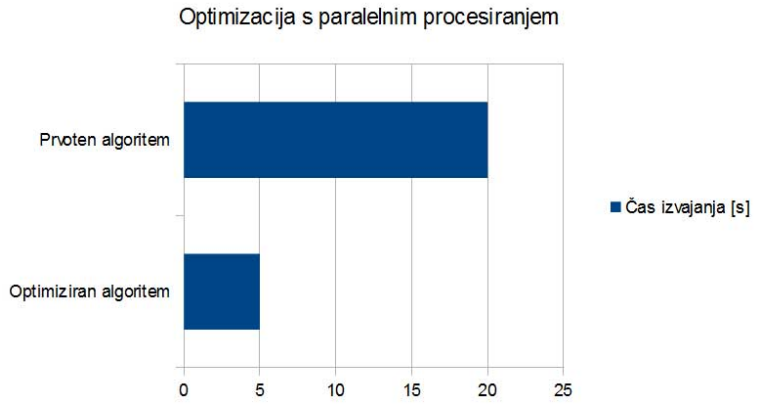
4.5 Optimizacija

Optimizacija je potekala na algoritmu za izračun in izris svetlo-temne meje na celotni širini slike (3648 točk). Upoštevane so samo vertikale na intervalu kota 0,25 stopinje. Oddaljenost kamere od zaslona je 920 cm. Računalnik na katerem so testi potekali ima 4 jedrni procesor. Izmerjen čas prvotnih algoritmov se je gibal okoli 50 sekund.

4.5.1 Slikovni filtri

Prve verzije teh algoritmov so uporabljale Microsoftov .Net razred System.-Drawing.Bitmap za branje in pisanje točk na/s slike. Ta razred vsebuje dve metodi prva je GetPixel(x,y), ki vrne barvo točke. Druga je SetPixel(x,y,color), ki nastavi barvo točki. Namesto tega sem uporabil direkten dostop do slike, kjer so točke shranjene v zaporedjih štirih komponent R, G, B in A[6]. Takšen direkten dostop do katerekoli točke na sliki je skrajšal čas algoritma za več kot polovico, na 20 sekund (slika 4.18).

[6]Komponenta A določa transparentnost.



Slika 4.19: Vizualna predstavitev pospešitve algoritmov s paralelnim procesiranjem.

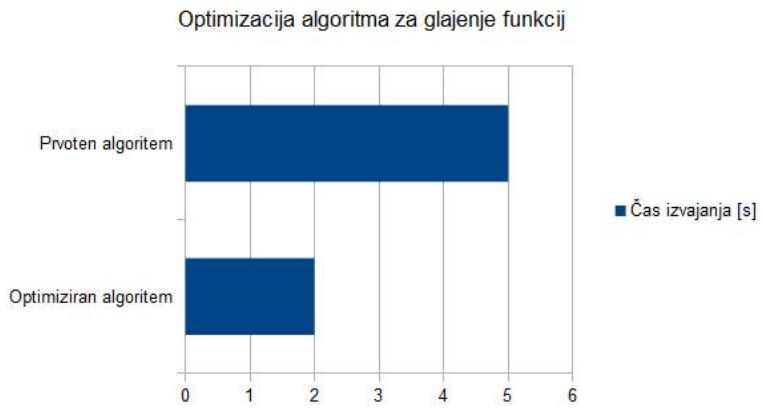
4.5.2 Paralelno procesiranje

Kot sem že omenil v poglavju 4.4.9 sem pri iskanju svetlo-temne meje na vertikalah uporabil paralelno for zanko. Algoritem brez paralelnega procesiranja z izboljšanim branjem in pisanjem slike je potreboval 20 sekund[7]. Z 4 jedrnim procesorjem in paralelnim izvajanjem se je ta čas dvakrat razpolovil na 5 sekund. Tukaj se vidi, da je tak način zelo primeren za naš problem. Prav tako hitrost algoritma narašča z hitrostjo računalnika (če podvojimo enote, ki obdelujejo naš problem, se tudi hitrost algoritma podvoji) (slika 4.19).

4.5.3 Algoritem za glajenje funkcij

Po podrobni analizi kode sem ugotovil, da glavni algoritem največ časa porabi pri glajenju funkcije. Začetni algoritem je za vsak element sešteval polovico velikosti okna elementov pred njim in za njim. Ta algoritem sem nato spremenil, da hrani vsoto okna skozi celotno vertikalo. Vsak nov element, ki pride v okno se prišteje k vsoti. Odšteje se pa element, ki zapusti okno. Tako je

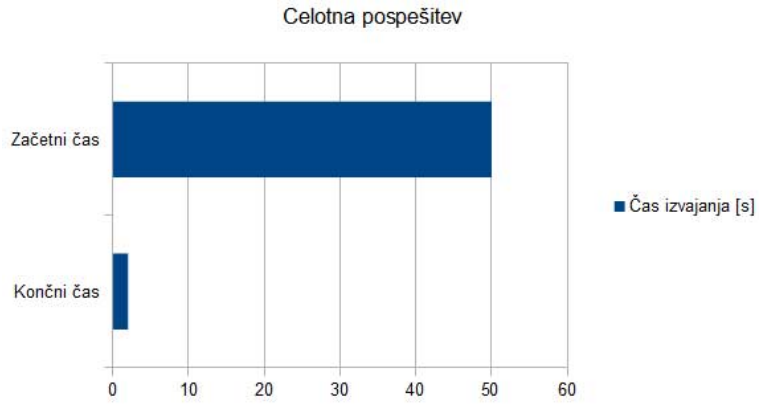
[7]Poleg tega pa knjižnica .Net ne omogoča pisanja v sliko iz več niti naenkrat, tako da paralelno procesiranje z njo ni bilo mogoče.



Slika 4.20: Vizualna predstavitev pospešitve algoritma za glajenje funkcije.

časovna zahtevnost algoritma padla iz O(velikost okna * n) na O(n). Hitrost algoritma pa na 2 sekundi (slika 6.6).

Na sliki 4.21 je prikazana celotna pospešitev algoritmov po optimizaciji. Hitrost izvajanja se je znižal iz 50 sekund na 2 sekundi.



Slika 4.21: Vizualna predstavitev pospešitve algoritmov po optimizaciji.

Poglavje 5

Primerjava s fotometrom

Test je bil opravljen na levem in desnem žarometu za meglenke. Na pozicijah 0, -2 in +2 stopinj glede na vertikalno vodilno črto. Izmerjene vrednosti so v stopinjah glede na horizontalno vodilno črto.

5.1 Levi žaromet

Pri tem žarometu je program (Prenosni merilnik svetlobnih lastnosti žarometov) izračunal rezultate, ki so skoraj isti kot fotometer. Pri 0° je program izračunal isto vrednost kot fotometer. Pri -2°in +2° pa je program izračunal vrednost, ki je za 0,02° različna od fotometra, kar je enako 0,3 cm (tabela 5.1). Podrobni rezultati fotometra se nahajajo v prilogah (6).

Pozicija	**Fotometer**	**Prenosni merilnik**
0°	-1,26°	-1,26°
-2°	-1,26°	-1,24°
+2°	-1,26°	-1,24°

Tabela 5.1: Primerjava rezultatov fotometra in prenosnega merilnika za levi žaromet.

Pozicija	Fotometer	Prenosni merilnik
0°	-1,02°	-1,23°
-2°	-0,98°	-1,19°
+2°	-1,08°	-1,22°

Tabela 5.2: Primerjava rezultatov fotometra in prenosnega merilnika za desni žaromet.

5.2 Desni žaromet

Tukaj je program vrnil različne rezultate pri vseh pozicijah. Razlike so pri poziciji 0° 0,11° (1,63 cm) pri -2° 0,21° (3,11 cm) in pri 2° 0,14° (2,07 cm) (tabela 5.2). Na različne rezultate lahko vpliva več dejavnikov. Lahko je to slaba kvaliteta slike. Lahko so moteči elementi na sliki, kot so vodilne črte. Lahko je drugačna osvetlitev slike. Lahko pa tudi temperatura žarometa krči ali raztegne materiale in so zato drugačni rezultati med enimi ali drugimi meritvami. Podrobni rezultati fotometra se nahajajo v prilogah (6).

Poglavje 6

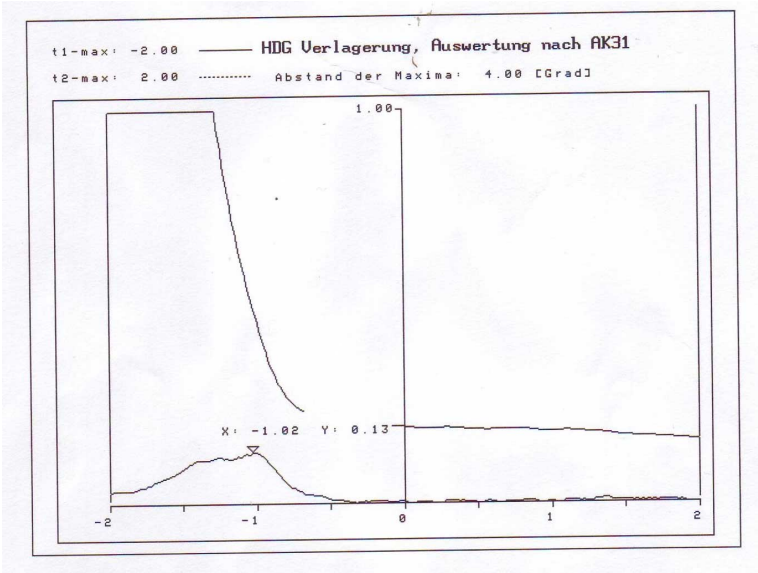
Zaključne ugotovitve

Program vključuje vso funkcionalnost, ki je bila potrebna. Deluje hitro. Je nastavljiv in zna obdelat veliko vrst slik snopov svetlobe.
Dobljeni rezultati so na nekaterih slikah še različni od fotometra. Tako, da za nadaljnji razvoj bi bilo potrebno nekatere algoritme še malo spremeniti in določiti najbolj optimalne nastavitve fotoaparata ali kamere.
Med reševanjem problema mi je največjo težavo povzročal izračun druga odvoda. Poizkusil sem veliko možnih rešitev kako čim bolj natančno odvajati funkcijo. Na koncu sem problem rešil tako, da sem funkcijo zgladil in jo šele potem odvajal. Vendar pa so bili tudi tukaj težave. Potrebno je bilo najti postopek kako najti najboljši približek naši originalni funkciji snopa, ki se ga da lepo odvajati. Na tem delu je največja možnost napake ampak tudi najboljša priložnost, da se izboljša natančnost.
Včasih je problem pri obdelavi slike in sicer pri detekciji horizontalnih vodilnih črt. To se še posebej opazi pri žarometih za meglenke, kjer se mejo postavi na zadnjo vodilno črto. Tako sta prvi dve v temi in se ju včasih ne zazna. Moja ideja za izboljšavo je, da fotografija vsebuje en del tal sobe. Tretjo horizontalno črto in tla se nato uporabi kot referenčni točki in se izračuna pozicijo prve in druge vodilne črte.

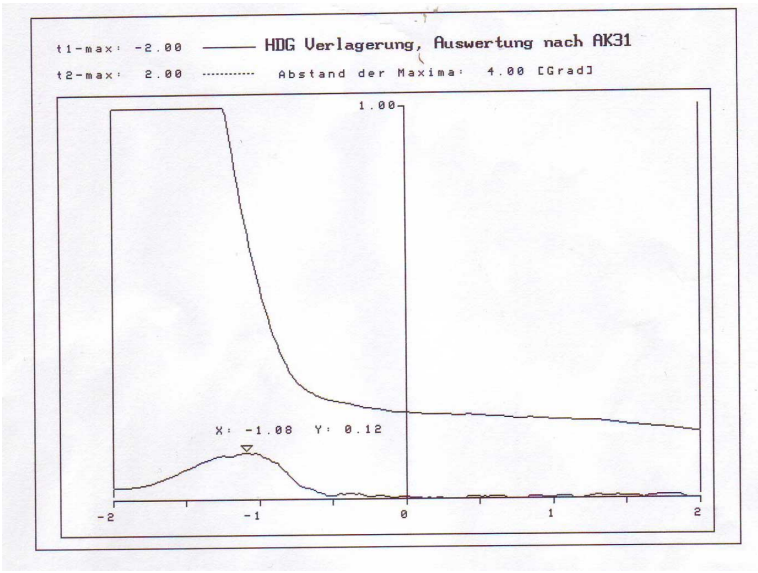
Poglavje 7

Priloge

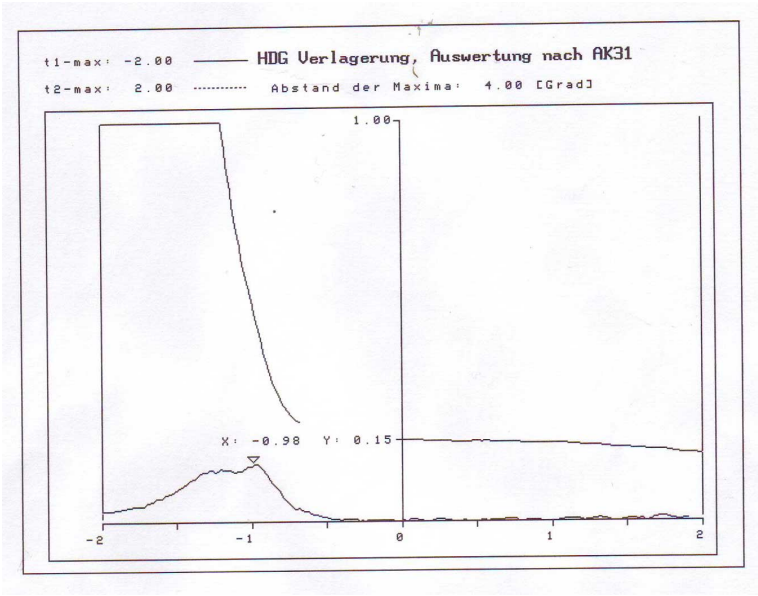
7.1 Rezultati fotometra



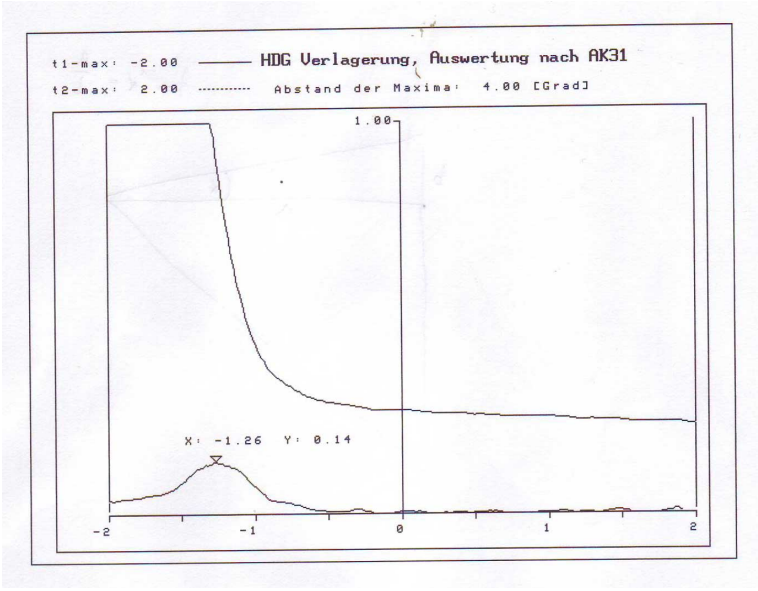
Slika 7.1: Desni žaromet, pozicija 0°.



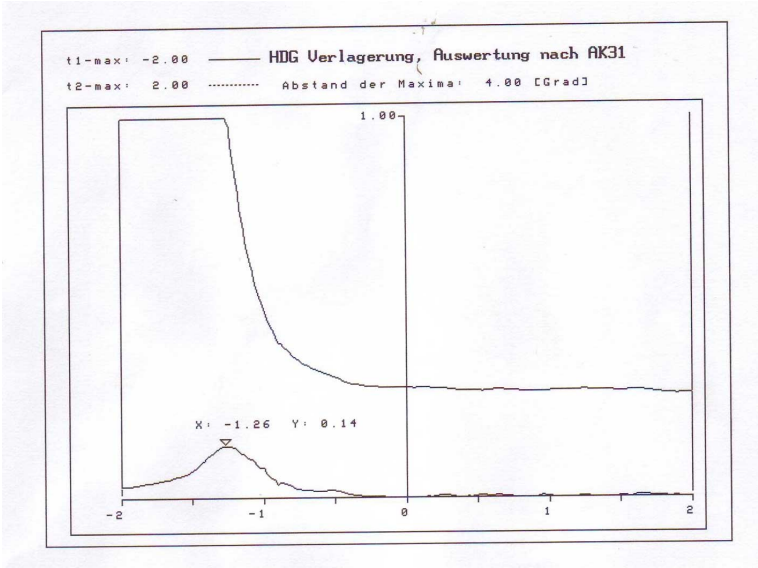
Slika 7.2: Desni žaromet, pozicija +2°.



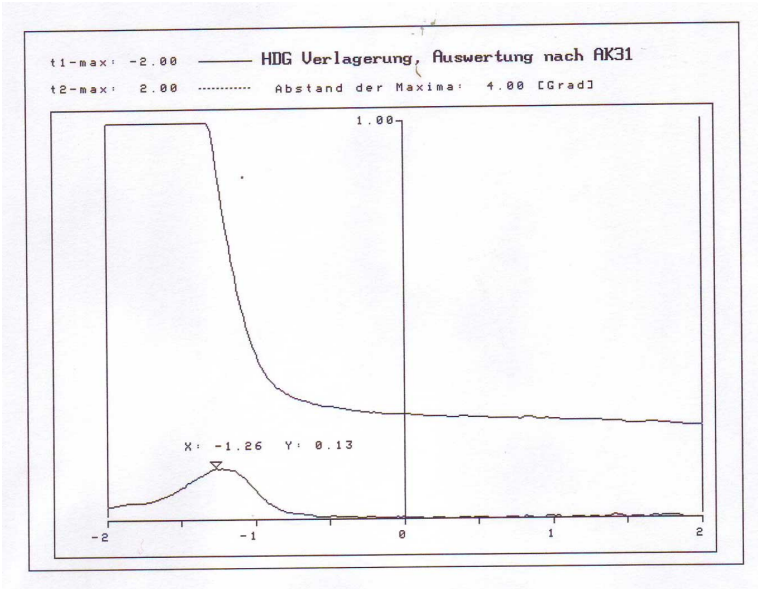
Slika 7.3: Desni žaromet, pozicija -2°.



Slika 7.4: Levi žaromet, pozicija 0°.



Slika 7.5: Levi žaromet, pozicija +2°.



Slika 7.6: Levi žaromet, pozicija -2°.

Literatura

[1] (2012) Aforge.Net Sobelov filter. Dostopno na: http://www.aforgenet.com/framework/docs/html/2c8218cc-921c-34d8-5c88-39c652488490.htm http://en.wikipedia.org/wiki/Sobel_operator

[2] (2012) Aforge.Net Hough-jeva transformacija. Dostopno na: http://www.aforgenet.com/framework/features/hough_transformation.html http://en.wikipedia.org/wiki/Hough_transform

[3] (2012) BeanstalkD. Dostopno na: https://github.com/kr/beanstalkd

[4] Eric Freeman, Elisabeth Robson, Bert Bates, Kathy Sierra, Head First Design Patterns, O'Reilly, 2004, ISBN:978-0-596-00712-6 — ISBN 10:0-596-00712-4

[5] Andrew Stellman, Jennifer Greene, Head First C#, O'Reilly, 2007, ISBN:978-0-596-51482-2 — ISBN 10:0-596-51482-4

[6] Shapiro, Linda and Stockman, George. Čomputer Vision", Prentice-Hall, Inc. 2001

[7] K. Engel (2006). Real-time volume graphics,. pp. 112–114.

[8] Gonzalez, Rafael C. & Woods, Richard E. (2002). Thresholding. In Digital Image Processing, pp. 595–611. Pearson Education. ISBN 81-7808-629-8

[9] (2012) Finite difference, Wikipedia. Dostopno na: http://en.wikipedia.org/wiki/Finite_difference

[10] Charles A. Poynton (2003). Digital Video and HDTV: Algorithms and Interfaces. Morgan Kaufmann. ISBN 1558607927.

[11] (2012) Lightness, Hue, Saturation, Wikipedia. Dostopno na: http://en.wikipedia.org/wiki/HSL_color_space