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

Mentor: doc. dr. Peter Peer

Ljubljana, 2012

Izjava o avtorstvu diplomskega dela

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 Peera,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela in
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 15. aprila 2012 Podpis avtorja:

Zahvaljujem se mentorju doc. dr. Petru Peer, za pomoč in svetovanje pri izdelavi diplomske naloge.

Zahvaljujem se tudi mag. Janku Kerncu in podjetju Hella Saturnus Slovenija d.o.o., ki sta mi omogočila material, izvedbo in testiranje za diplomsko delo.

Hvaležen sem mojim staršem, 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 (zaslon), kjer so narisane vodilne črte. Tri horizontalne in ena vertikalna. Žaromet se postavi tako, da sveti pravokotno na centralno vertikalno in horizontalno črto. Program iz slike izračuna prehod svetlo-temne meje po celotni širini slike. Uporabnik lahko 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 založnik-naročnik. Preko njega se prenašajo slike in drugi parametri med posameznimi koraki. Vhodna slika se najprej pretvori v črno-belo sliko. Nato se na črno-beli sliki uporabi Sobelov filter, da se iz nje izlušči vodilne črte. Na tej sliki se izvede še Houghova 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, kjer je drugi odvod funkcije enak nič. Iz drugega odvoda se izračuna še začetek (maksimum) in konec (minimum) svetlo-temne meje. Te tri črte narišemo na sliko in izračunamo širino svetlo-temne meje. To na koncu prikažemo 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 kota 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 an image of a car light's beam. Beam is projected on a wall where guidelines are. Three horizontal and one vertical line. Program calculates from the image the cut-off line of the beam. This is a transition line from dark to light. User can select a vertical line on an image 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 program. Input image is first transformed into a black and white image. 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 image in angles and centimetres from the central guidelines. For each vertical on an image 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 a second derivative of this curve equals zero. Then the beginning and the end of the cut-off line are calculated (maximum and minimum of the 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 movements 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 image. Program works fast. It can find a cut-off line on an image (with an interval of 0,25°) in about two seconds.

Poglavje 1

Uvod

Avtomobilska industrija je zelo zahtevno področje. Dosti konkurence. Dosti povpraševanja. Visoki standardi za varnost in kvaliteto. Avtomobili se proizvajajo na tekočem traku. Vsak upad proizvodnje je drag. Izjemno drag. Tukaj govorimo o več 1.000 in celo več 10.000 evrih na minuto. Vsak izdelek, ki pride v tovarno in se montira na avtomobile, tovornjake, motorje ipd., mora biti praktično neoporečen. Dovoljenih je le nekaj slabih kosov na milijon kosov. 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 testiraš samo en žaromet naenkrat. Poleg tega je vsak del naprave potrebno vzdrževati. Cena fotometra se giblje okoli 500.000 evrov. Ampak kljub temu je potrebna, saj se brez nje ne da doseči potrebne kvalitete, ki jo zahtevajo stranke.

Cilj te diplomske naloge je, da razvijemo alternativo te napravi. Ta alternativa bo v obliki računalniškega programa, ki bo obdeloval sliko snopa svetlobe žarometa. Soba bo seveda še vedno potrebna. Vendar jo bo potrebno manj zaščititi pred motnjami. Namesto fotometra potrebujemo samo fotoaparat ali kamero. Naš program mora tako omogočati vsaj isto funkcionalnost. Biti mora hitrejši in dati mora natančne rezultate.

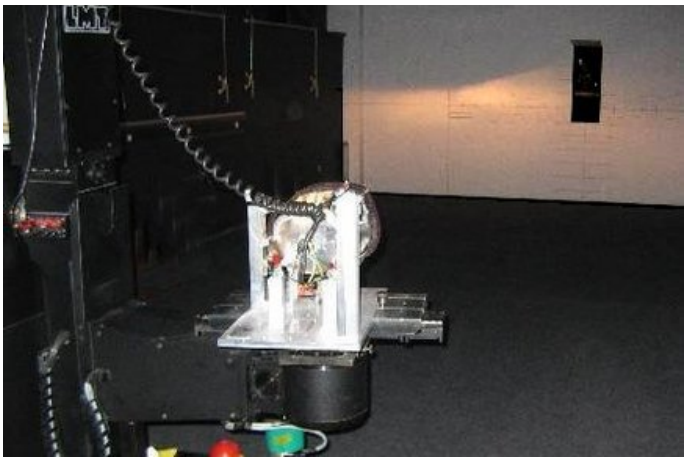
Poglavje 2

Opis problema

Fotometer (slika 2.1) je naprava, ki jo v podjetju Hella Saturnus d.o.o. uporabljajo za določanje svetlobnih lastnosti žarometov [12]. Žaromet se privije na optično os. Ta je nasproti fotocelici na razdalji petindvajsetih metrov. Za svetilke se uporablja razdalja treh metrov. Žarnice, ki se pri testih uporabljajo, so drugačne od tistih v proizvodnji žarometov. Nitka v žarnici mora biti določene dolžine in je ne sme presegati za več kot milimeter. Daljša nitka lahko namreč premakne svetlo-temno mejo. Pri merjenju fotocelica stoji na miru. Premika se samo žaromet in sicer v vertikalni smeri. Na vsake 0,01° se pomeri svetilnost. Tako se ugotovi svetlo-temno mejo. Nahaja se tam, kjer je največja razlika (gradient) svetlobe med dvema točkama. Na razdalji 0,01° svetloba naraste za 20%, pri ksenon žarnicah tudi do 200%. Nato 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.

Slika 2.1 prikazuje fotometer v delovanju. Na sredini je optična os z žarometom. Žaromet sveti na zaslon, kjer so narisane vodilne črte. V odprtini na zaslonu je fotocelica, katera izvaja meritve.

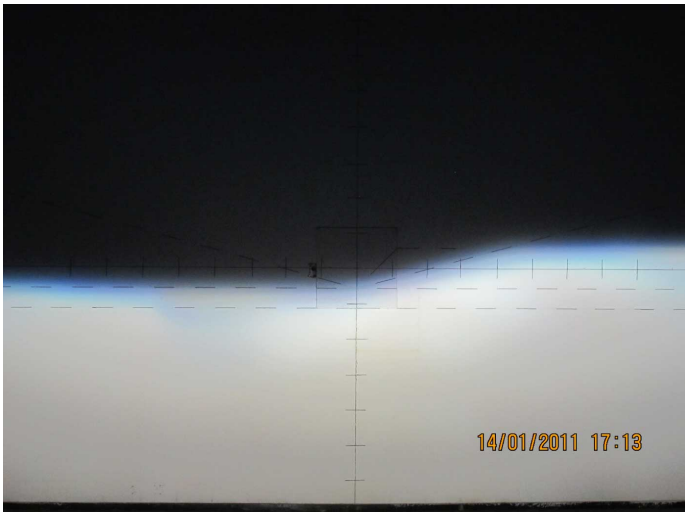
V naslednjem poglavju je opisan razvoj naše aplikacije. Najprej podam zahteve strojne opreme in opis programskih jezikov, ki so na voljo. Nato je opisan uporabniški vmesnik, ter celoten postopek obdelave slike, računanja



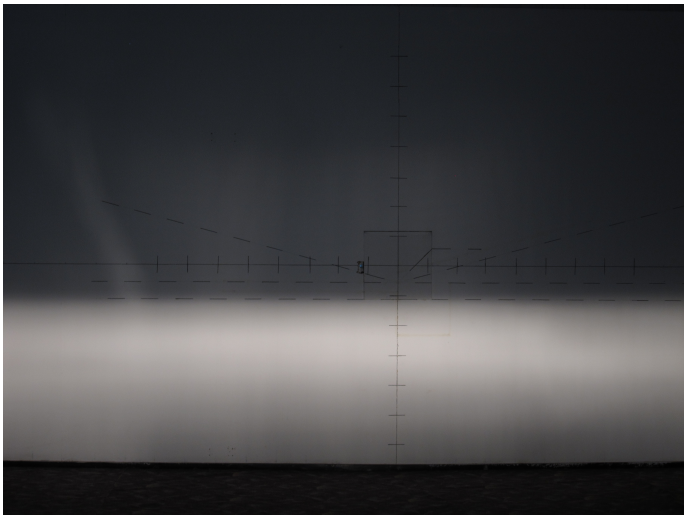
Slika 2.1: Slika fotometra.

svetlo-temne meje. Sledi opis postopka optimizacije. Podani so časi hitrosti algoritmov pred in po optimizaciji. V poglavju 4 je narejena primerjava rezultatov našega programa in fotometra. Nato v poglavju 5 podamo zaključne ugotovitve.

Moramo torej 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. Snop iz žarometa usmerimo na zaslon, 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 $0°$ 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, torej oddaljenost glede na horizontalno centralno črto. Izračunati moramo širino svetlo-temne meje. Sliki 2.2 in 2.3 sta primera vhodnih slik.



Slika 2.2: Vhodna slika, sprednji žaromet.



Slika 2.3: Vhodna slika, meglenka.

Poglavje 3

Razvoj lastne aplikacije

3.1 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 večjedrni.
- 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 nekaj MB.
- Fotoaparat ali kamera (v nastavitvi za slikanje in ne snemanje).
- Priporočena nastavitev fotoaparata za uravnavanje beline ISO je 100, slikanje s števcem ali stojalom.

3.2 Uporabljene programske 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 ipd. Spodaj so našteti dodatni plusi in minusi, ki sem jih ugotovil.

3.2.1 Java

Plusi

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

Minusi

- Ne vsebuje standardnega uporabniškega vmesnika Windows.

3.2.2 C#

Plusi

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

Minusi

- Počasna vgrajena knjižnica za obdelavo slik.

3.2.3 Delphi

Plusi

- Standarden Windows uporabniški vmesnik.

Minusi

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

3.2.4 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).

3.2.5 PHP

Plusi

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

Minusi

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

3.2.6 Izbira

Moja končna izbira je bil C#. Veliko tudi 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 metode. 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 filter in Houghovo črtno transformacijo. 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.

3.3 Zasnova uporabniškega vmesnika

Uporabniški vmesnik je razdeljen v osem glavnih sklopov. Ta 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 3.1).

3.3.1 Vtičniki

Vmesnik je zasnovan na principu vtičnikov [2, 6]. Vsak zavihek je svoj vtičnik in vsak ima svoj uporabniški vmesnik za konfiguracijo nastavitev. Definirani

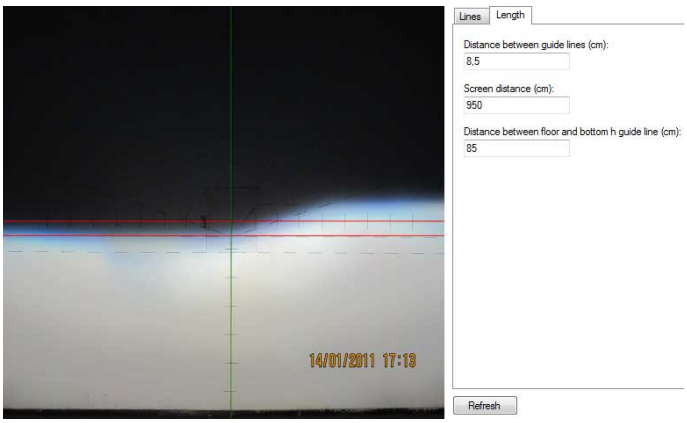
Slika 3.1: Uporabniški vmesnik razvitega programa.

so s pomočjo programskega vmesnika, ki zgleda tako:

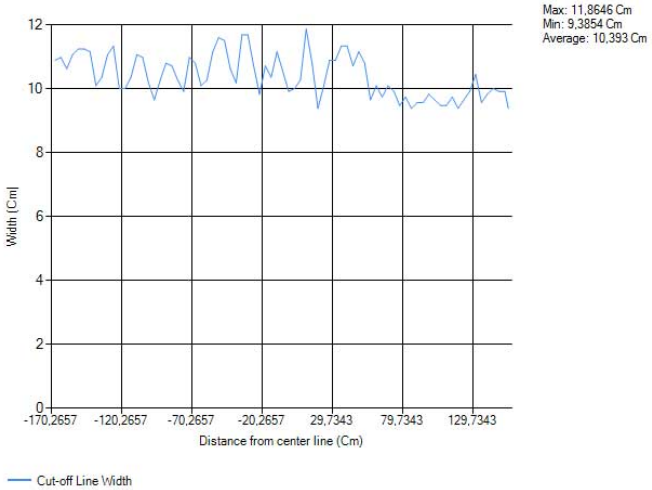
```
interface IImagePlugin {
    UserControl UserInterface { get; }
    string Name { get; }
    bool ValidateInput();
    void ProcessImage();
}
```

Lastnost UserInterface vrne uporabniški vmesnik vtičnika. Ta je lahko standarden s sliko na levi strani in konfiguracijskimi polji in gumbi na desni strani (slika 3.2). Lahko pa vsebuje samo graf (slika 3.3).

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



Slika 3.2: Primer uporabniškega vmesnika za iskanje vodilnih črt.



Slika 3.3: Primer uporabniškega vmesnika z grafom.

3.3.2 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 [6] naložijo vse knjižnice s vtičniki. V vsaki knjižnici se poišče implementacijo vmesnika 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)
```

3.3.3 Model založnik-naročnik

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 model (angl. design pattern) po imenu založnik-naročnik (angl. Publisher-Subscriber) [2, 6]. Deluje na principu dogodkov. Pri tem modelu nekdo pošilja sporočila. Ta lahko vsebujejo slike, mere, velikosti točk ipd. Vsi, ki so prijavljeni na določeno sporočilo, ga nato prejmejo in obdelajo. Vsebuje dve funkciji:

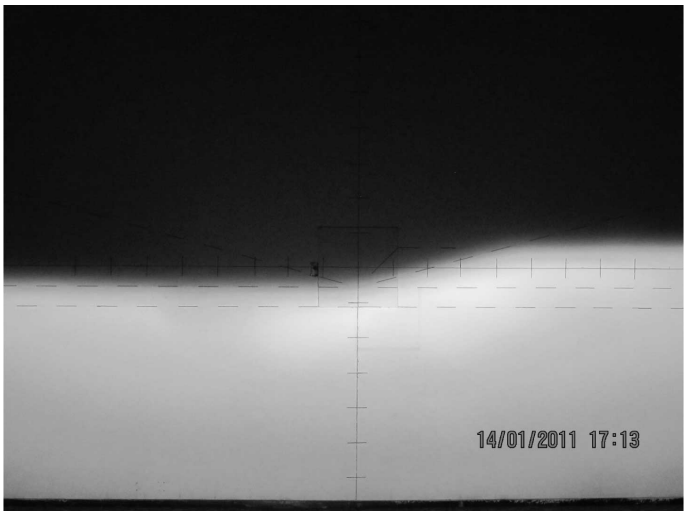
```
class PubSub {
    void Publish<T>(T message);
    void Subscribe<T>(Action<T> callback);
}
```

Subscribe metoda sprejme kot tip sporočila parameter, na katerega se prijavljamo. Callback parameter je referenca na metodo, ki se pokliče, ko nekdo objavi sporočilo s tem tipom. Publish metoda se kliče, ko nekdo želi 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. Potem ponovno kliče metodo Publish s sporočilom tipa BlackAndWhiteImageMessage. Tega nato prejme naslednji vtičnik in tako naprej.

3.4 Postopek določanja svetlobnih lastnosti žarometov

3.4.1 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 enačbi 3.1 [11].



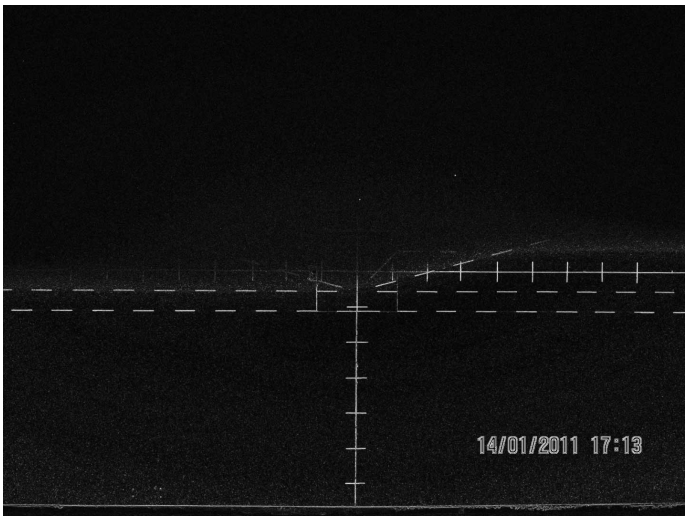
Slika 3.4: Rezultat pretvorbe vhodne barvne slike v črno-belo sliko.

$$L = \frac{1}{2}(M + m) \tag{3.1}$$

Pri tem je M največja vrednost izmed RGB [4] 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 2^{24} 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 3.4.



Slika 3.5: Rezultat Sobelovega filtera.

3.4.2 Sobelov filter

Drugi korak v našem postopku je uporaba Sobelovega filtra [1], ki na naši črno-beli sliki odkrije robove vodilnih črt. Tukaj je bila uporabljena knjižnica za računalniški vid Aforge.NET [7], kjer je ta filter že implementiran. Potreben je klic funkcije:

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

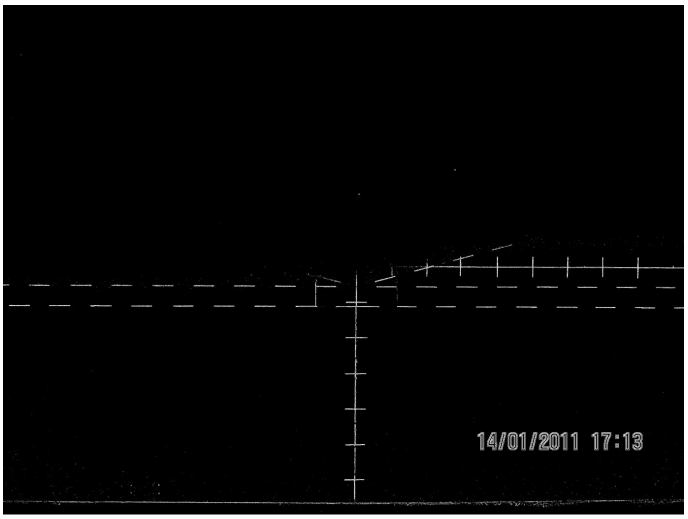
Rezultat filtra je slika 3.5.

3.4.3 Prag

S tretjim korakom izločimo šum z algoritmom za določanje pragu [3]. To storimo tako, da za vsako točko pogledamo njeno intenziteto (po pretvorbi v črno-belo sliko imajo vse tri komponente točke isto vrednost). Intenziteta je enaka vrednosti ene izmed RGB komponent. Če je intenziteta večja od vrednosti, ki si jo uporabnik izbere (intenzitete imajo lahko vrednost od 0 do 255), potem na novo sliko vpišemo točko z maksimalno intenziteto. Če je ta vrednost manjša od izbrane, vpišemo točko z minimalno intenziteto (bela točka ima intenziteto 255, črna pa intenziteto 0). Tako na novi sliki še bolj poudarimo vodilne črte in zmanjšamo količino šuma (torej točke, ki niso del vodilnih črt in 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 sliko in vrednost pragu. Kot rezultat pa vrne sliko 3.6.



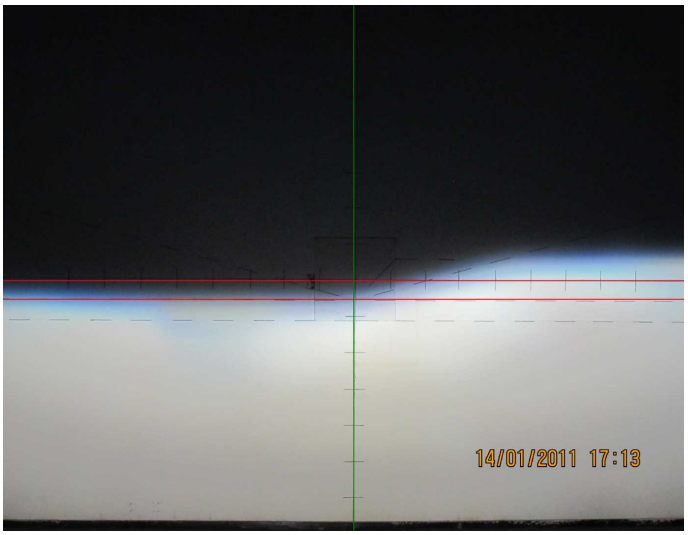
Slika 3.6: Rezultat pragovnega filtra.

3.4.4 Iskanje vodilnih črt

V tem koraku iščemo prvi dve horizontalni vodilni črti (od zgoraj navzdol), ter sredinsko vertikalno vodilno črto. Uporabimo Houghjevo črtno transformacijo [5], s katero najdemo črte na sliki. Tudi ta je že implementirana v Aforge.NET knjižnici [8]. Primer klica za Hough-jevo transformacijo:

```
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 in kot med -5° in 5° (s tem kotnim pogojem najdemo vertikalne črte, za horizontalne črte pa uporabimo kote od 85° do 95°). Relativna jakost črte je odvisna od števila belih pik, ki 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° pa blizu naše iskane vertikalne vodilne črte. [8]



Slika 3.7: Vodilne črte.

Te črte (slika 3.7) se nato uporabijo za določitev mer na sliki (v kotih in centimetrih). Uporabnik vnese razdaljo med horizontalnima črtama v cm in razdaljo od zaslona do kamere v cm. Iz tega lahko izračunamo, koliko točk na sliki je en centimeter. Formula za izračun:

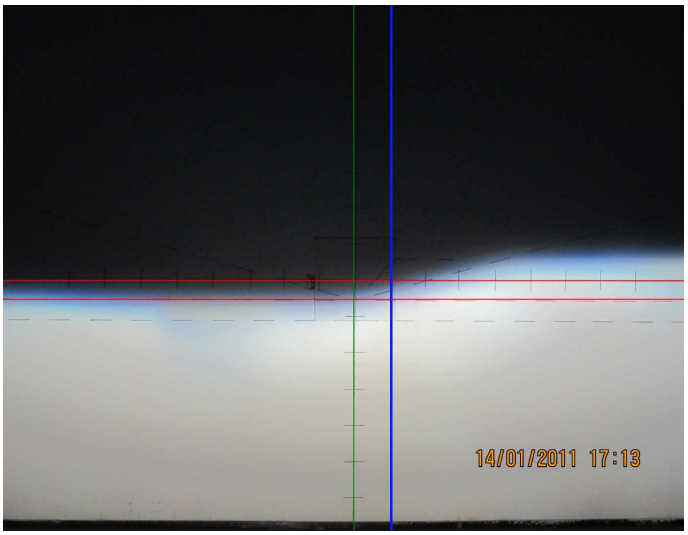
$$v = \frac{a}{|r_1 - r_2|}, \tag{3.2}$$

kjer v predstavlja velikost točke v cm, spremenljivka a je razdalja med horizontalnima vodilnima črtama v cm, razlika med r_1 in r_2 pa je enaka razdalji med horizontalnima vodilnima črtama na sliki v točkah (rdeči črti na sliki 3.8).

Za vsako točko na sliki lahko izračunamo kot glede na centralno vertikalno ali horizontalno vodilno črto. Formula za izračun kota:

$$tan(\alpha) = \frac{a}{b}, \tag{3.3}$$

kjer je a oddaljenost v cm trenutne točke od centralne vodilne črte, b je oddaljenost zaslona od kamere v cm, α je kot, ki ga iščemo. Na sliki 3.8 je prikazan primer oddaljenosti 1° od centralne vertikalne črte (razdalja med zeleno in modro vertikalno črto).



Slika 3.8: Črte za umeritev slike (zelena in rdeči črti) in oddaljenost za 1° od centralne vertikalne vodilne črte (modra črta).

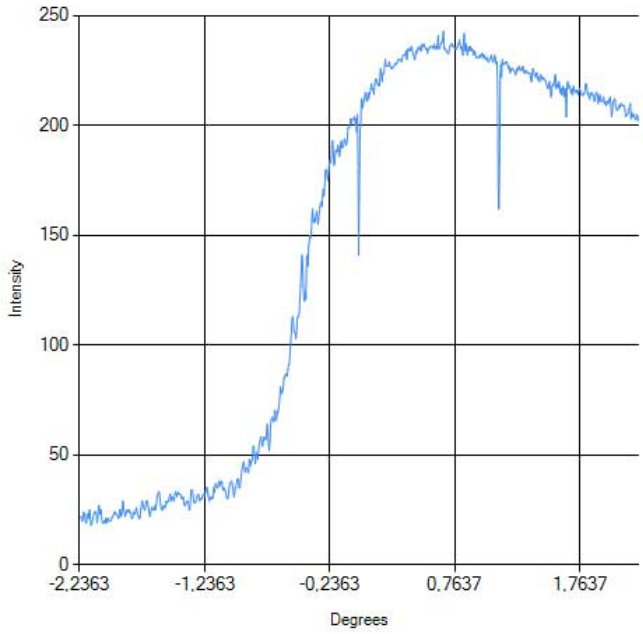
3.4.5 Pregled intenzitet po vertikali

Ta korak je glavni za določitev svetlo-temne meje. To storimo tako, da izberemo vertikalo na sliki in pogledamo intenzitete njenih točk. Za sliko uporabimo črno-belo sliko iz prvega koraka. Intenzitete nato izrišemo na grafu (sliki 3.9 in 3.10).

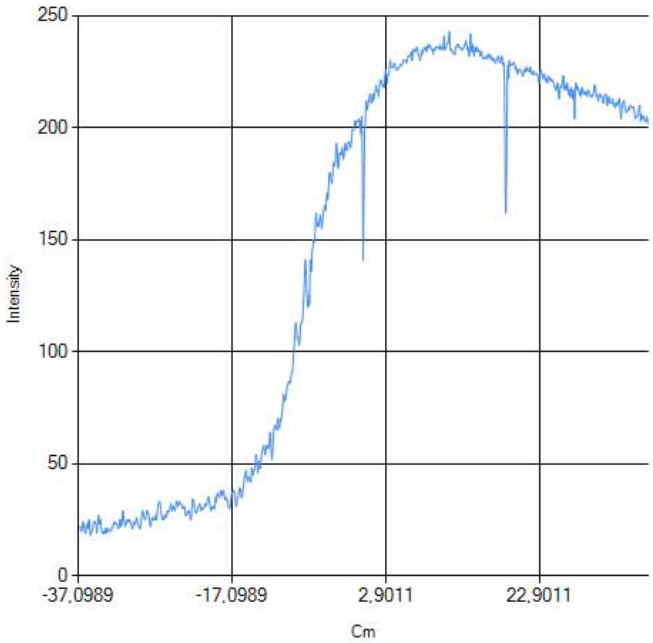
Os x na sliki 3.9 predstavlja oddaljenost v stopinjah, na sliki 3.10 pa v cm, od horizontalne vodilne črte. Na y osi je prikazana intenziteta točke. Primer algoritma v psevdokodi zgleda tako:

```
for (int i = 0; i < visina(slika); i++)
    vrednost = slika[i][x].R
    rezultat.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), nato pretvori koordinate vrstice i v izbrano enoto (stopinje ali centimetri), na koncu pa shrani intenziteto in koordinate vrstice v seznam parov XY.



Slika 3.9: Intenzitete izbrane vertikale v stopinjah.



Slika 3.10: Intenzitete izbrane vertikale v centimetrih.

3.4.6 Iskanje svetlo-temne meje

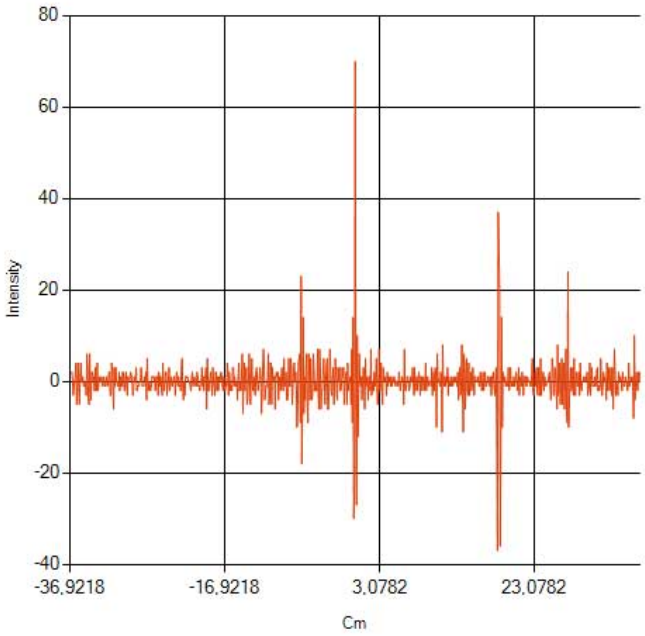
Za svetlo temno mejo vzamemo prevojno točko na grafu. Ta točka se nahaja tam, kjer je drugi odvod funkcije naših intenzitet enak nič. Za izračun drugega odvoda uporabimo [10]:

$$d'' = f(x+1) - 2f(x) + f(x-1), \tag{3.4}$$

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 spodaj). Začnemo na drugem elementu v našem seznamu (saj formula potrebuje prejšnji element), končamo pa na predzadnjem elementu (saj 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 3.11 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 (sliki 3.9 in 3.10) zgladimo z algoritmom za glajenje funkcij.



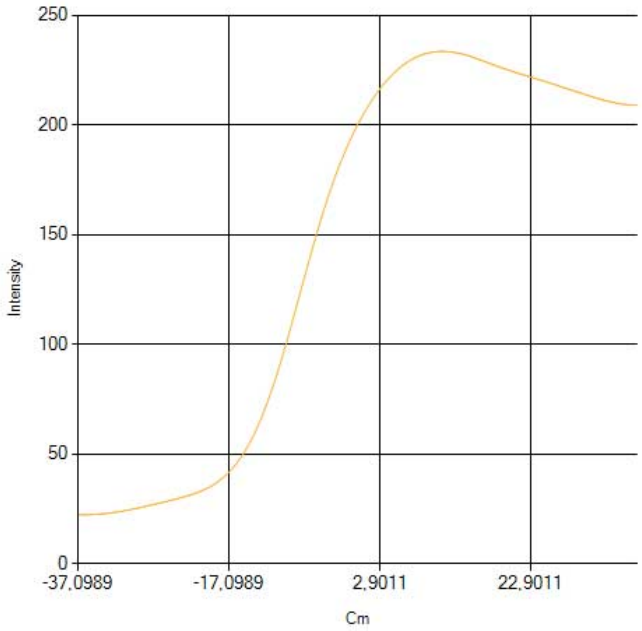
Slika 3.11: Drugi odvod funkcije intenzitet po vertikali.

3.4.7 Glajenje funkcije

V našem programu funkcijo gladimo tako, da jo povprečimo. Velikost okna (število elementov, ki jih vzamemo v povprečje, ko računamo trenutni element) je nastavljiva s strani uporabnika. Obstaja več načinov kako povprečiti funkcijo. V povprečju lahko upoštevamo samo elemente pred trenutnim, lahko samo elemente za trenutnim elementom. Najbolje se obnese, če vzamemo polovico velikosti okna elementov pred trenutnim in polovico elementov za trenutnim. Torej, če je naše okno veliko enajst elementov, začnemo s šestim elementom, k njemu prištejemo pet elementov pred njim in pet za njim. Število, ki ga dobimo, delimo z 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))
```

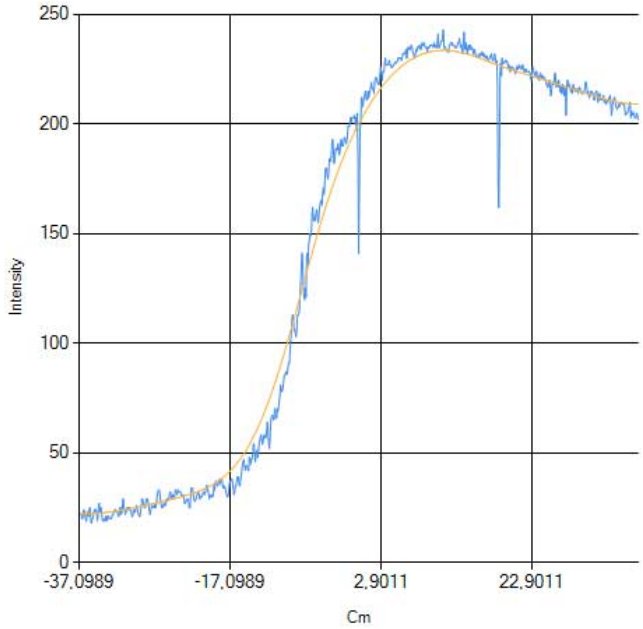
Spremenljivka r je rezultat in zglajena funkcija našega algoritma (slika 3.12 in slika 3.13). Slika 3.12 prikazuje našo zglajeno funkcijo. Slika 3.13 pa



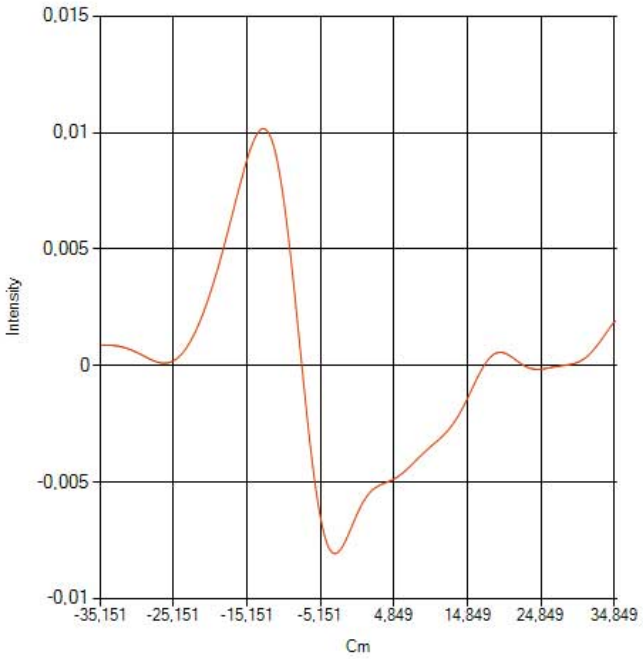
Slika 3.12: Zglajena funkcija intenzitet po vertikali.

prikazuje primerjavo med originalno in zglajeno funkcijo.

Če sedaj to zglajeno funkcijo (slika 3.12) odvajamo, dobimo lepšo funkcijo drugega odvoda (slika 3.14), kjer lahko določimo točko svetlo-temne meje.



Slika 3.13: Primerjava zglajene in originalne funkcije.



Slika 3.14: Drugi odvod zglajene funkcije.

3.4.8 Določanje svetlo-temne meje

Ničla na drugem odvodu naše funkcije (slika 3.14), ki poda svetlo-temno mejo, se poišče tako, da se najprej najde njegov maksimum. Potem se 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:

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

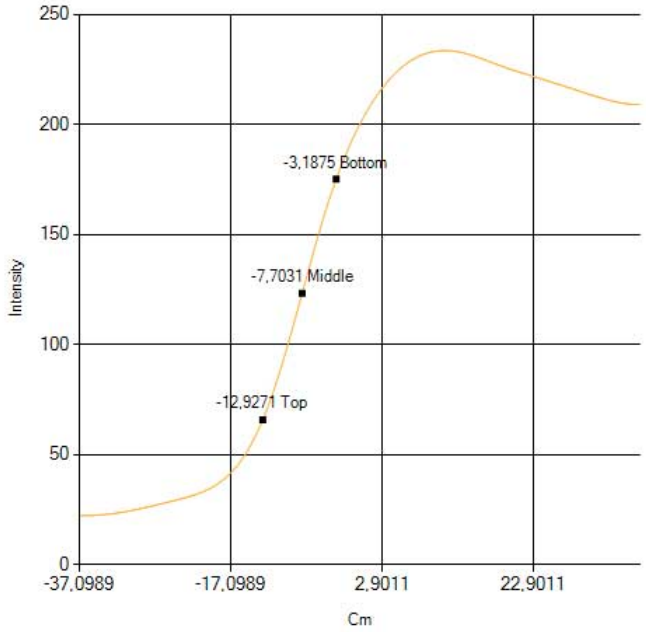
Če je ničla med najdenima elementoma, potem se jo določi preko linearne interpolacije:

$$\begin{aligned} k &= \frac{y_2 - y_1}{x_2 - x_1} \\ n &= y_1 - k * x_1 \\ z &= (-n/k) \end{aligned} \tag{3.5}$$

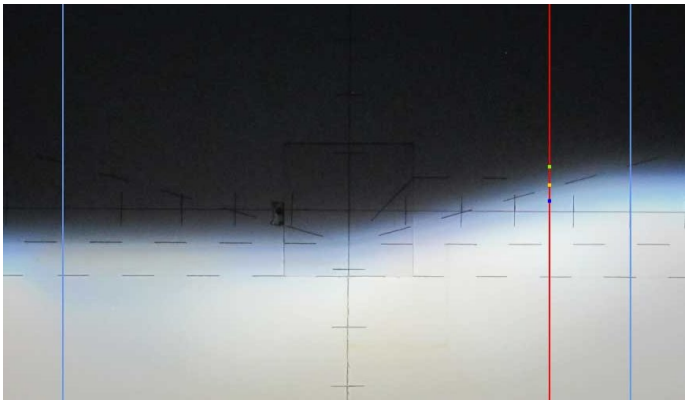
z predstavlja x koordinato, kjer se nahaja ničla na naši funkciji drugega odvoda in posledično točka svetlo-temne meje na naši originalni funkciji. To točko nato izrišemo na grafu (slika 3.15) in na naši originalni sliki (3.16).

Na sliki 3.15 je točka Middle izračunana svetlo-temna meja. Točki Top and Bottom sta maksimum in minimum funkcije drugega odvoda.

Rumena točka na sliki 3.16 predstavlja svetlo-temno mejo, zelena začetek oziroma masksimum drugega odvoda funkcije, modra pa konec svetlo-temne meje oziroma 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. To območje je enako območju osvetljenosti vozišča, ki ga vidi voznik avtomobila.



Slika 3.15: Začetna (Top), končna točka (Bottom) in točka svetlo-temne meje na grafu (Middle).



Slika 3.16: Začetna točka (zelena), končna točka (modra) in točka svetlo-temne meje na sliki (rumena).

3.4.9 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 pride zelo prav in olajša delo. Zanki podamo seznam elementov, ki jih hočemo obdelati. Ta pa jih nato razdeli na vsa procesorska jedra. Sintaksa je preprosta. Cel naš algoritem izgleda tako:

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

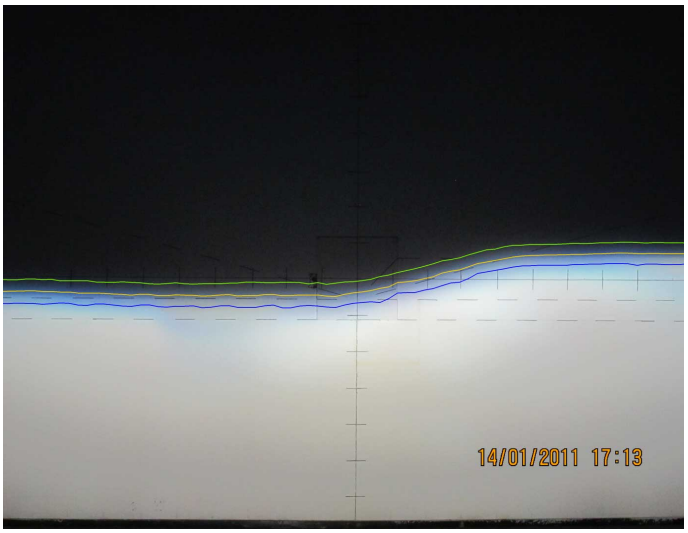
Algoritem nam najde vse točke po celotni širini slike za svetlo temno mejo. Vsa jedra procesorja so obremenjena. Več jeder imamo, hitreje deluje algoritem.

Alternativna implementacija je lahko z vnaprej kreiranimi nitmi in delavnimi (angl. worker) razredi. Razredu se poda kot parameter sliko in koordinato vertikale. Na koncu pa preko založnik-naročnik objekta sporoči rezultate. Lahko se uporabi distribuirano procesiranje na več računalnikih, kjer se izvajajo delavni procesi. Ti procesi sprejmejo iste parametre kot predhodni primer. Razlika je le v tem, da se parametri pošljejo preko vtičnic (angl. sockets) po lokalni mreži ali preko interneta. To implementacijo bi se dalo izvesti s prosto rešitvijo BeanstalkD [9], ki omogoča preprosto implementacijo vodje in delavnih procesov.

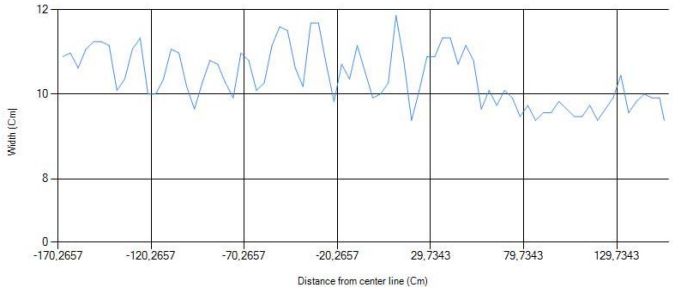
Na koncu nam preostane samo še, da te črte (algoritem vrne svetlo-temno mejo, njen začetek in konec) izrišemo na sliki (3.17).

3.4.10 Graf širine svetlo-temne meje

Za vsako izračunano vertikalo sedaj odštejemo višino črte začetka svetlo-temne meje na sliki in višino črte konca svetlo-temne meje na sliki. Te vre-



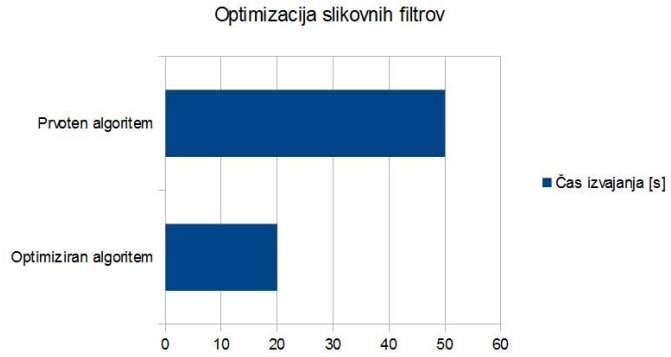
Slika 3.17: Črte začetka, konca in prehoda svetlo-temne meje.



Slika 3.18: Graf širine svetlo-temne meje.

dnosti nato prikažemo na grafu, glede na izbrano enoto (lahko v centimetrih ali kotih v stopinjah).

Na sliki 3.18 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. Meje te širine so določene s strani stranke in jih žarometi ne smejo presegati. Ta test nam tako pove, če žaromet zadostuje zahtevam ali je potrebno spremeniti njegove karakteristike.



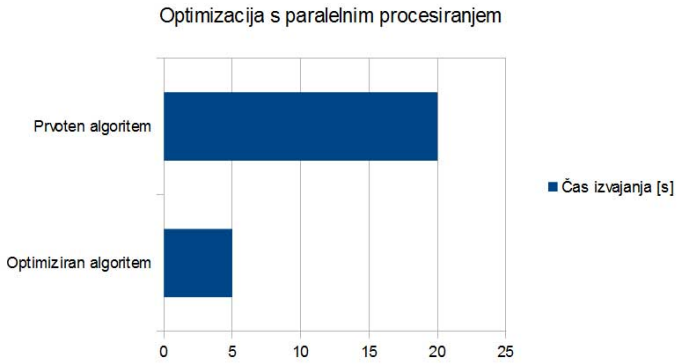
Slika 3.19: Pospešitev algoritmov za slikovne filtre.

3.5 Optimizacija

Optimizacija je potekala na algoritmu za izračun in izris svetlo-temne meje na celotni širini slike (3648 točk, celotna slika jih vsebuje 10 milijonov). Upoštevane so samo vertikale na intervalu kota 0,25°. 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.

3.5.1 Slikovni filtri

Prve verzije teh algoritmov so uporabljale Microsoftov .NET razred System.-Drawing.Bitmap za branje in pisanje točk s/na sliko. 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 (komponenta A določa transparentnost). Takšen direkten dostop do katerekoli točke na sliki je skrajšal čas algoritma za več kot polovico, na 20 sekund (slika 3.19).



Slika 3.20: Pospešitev algoritmov s paralelnim procesiranjem.

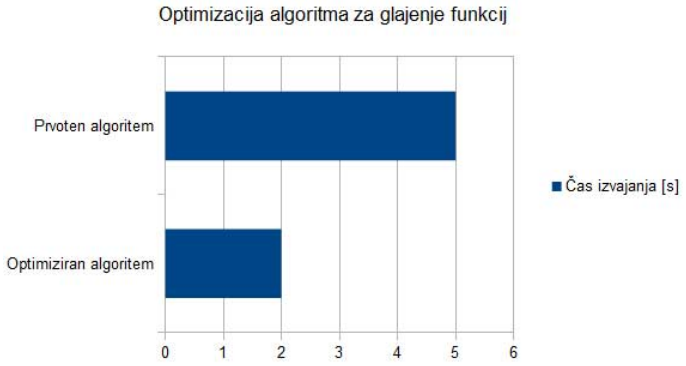
3.5.2 Paralelno procesiranje

Kot sem že omenil v poglavju 3.4.9, 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 (poleg tega pa knjižnica .NET ne omogoča pisanja v sliko iz več niti naenkrat, tako da paralelno procesiranje z njo ni bilo mogoče). S 4-jedrnim procesorjem in paralelnim izvajanjem se je ta čas zmanjšal na 5 sekund (slika 3.20). Tukaj se vidi, da je tak način zelo primeren za naš problem. Prav tako hitrost algoritma seveda narašča s hitrostjo računalnika.

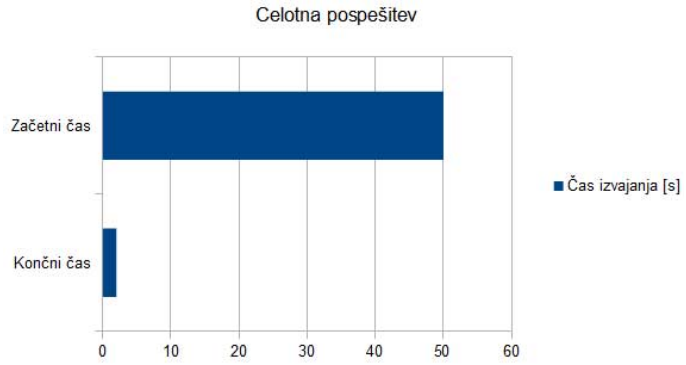
3.5.3 Algoritem za glajenje funkcij

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 element, ki zapusti okno. Tako je časovna zahtevnost algoritma padla iz O(velikost okna $\times$ n) na O(n) (n je število elementov v naši funkciji), hitrost algoritma pa na 2 sekundi (slika 3.21).

Na sliki 3.22 je prikazana celotna pospešitev algoritmov po optimizaciji. Hitrost izvajanja se je znižal iz 50 sekund na 2 sekundi.



Slika 3.21: Pospešitev algoritma za glajenje funkcije.



Slika 3.22: Pospešitev algoritmov po optimizaciji.

Poglavje 4

Primerjava rezultatov s fotometrom

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

4.1 Levi žaromet

Pri tem žarometu je naš program izračunal rezultate, ki so skoraj isti kot rezultati fotometra. 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 4.1). Podrobni rezultati fotometra se nahajajo v prilogah 6.1.

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

Tabela 4.1: 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 4.2: Primerjava rezultatov fotometra in prenosnega merilnika za desni žaromet.

4.2 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 4.2). 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 skrči ali raztegne materiale in so zato drugačni rezultati med enimi ali drugimi meritvami. Podrobni rezultati fotometra se nahajajo v prilogah 6.1.

Poglavje 5

Zaključne ugotovitve

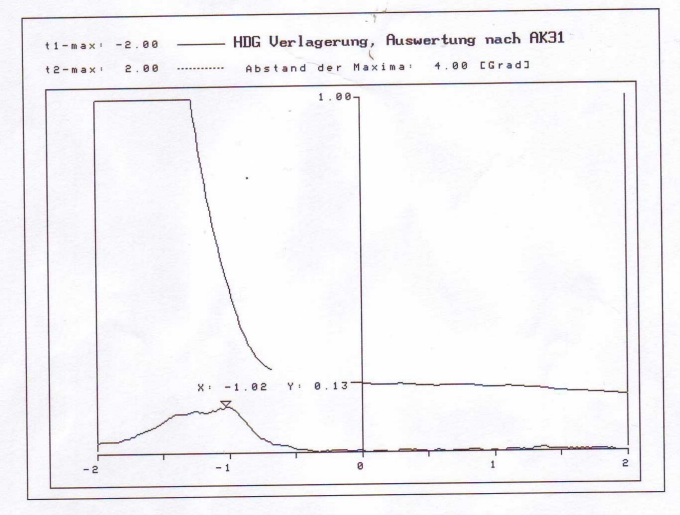
Program vključuje vso funkcionalnost, ki je bila potrebna. Deluje hitro. Je nastavljiv in zna obdelati veliko vrst slik snopov svetlobe. Dobljeni rezultati so na nekaterih slikah še različni od fotometra. Za nadaljnji razvoj bi bilo potrebno nekatere algoritme še malo spremeniti in določiti najbolj optimalne nastavitve fotoaparata ali kamere. Med reševanjem problema mi je največjo težavo povzročal izračun drugega 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 bile 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 6

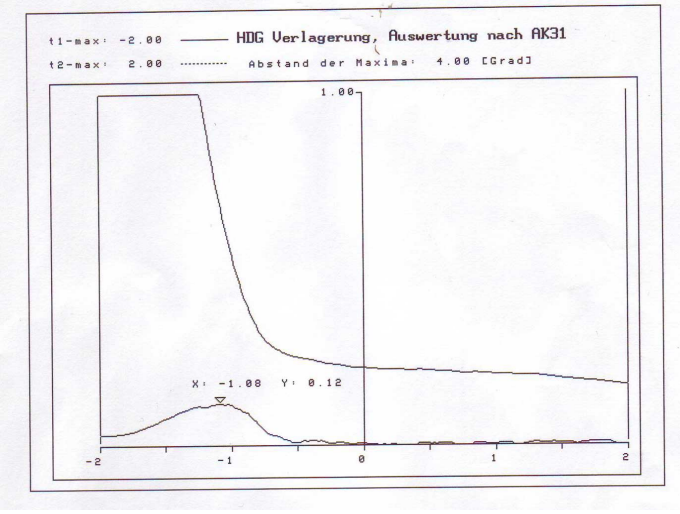
Priloge

6.1 Rezultati fotometra

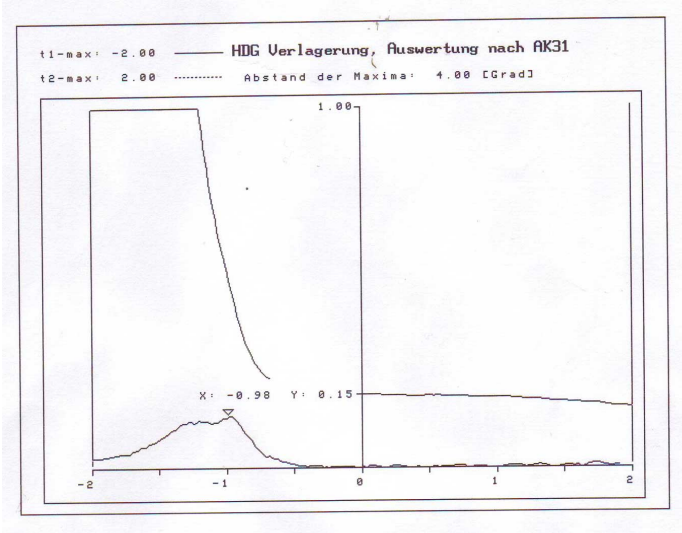
Meritve fotometra so bili izvedene na žarometih za levo in desno meglenko. Izmerjene so bile na pozicijah kotov +2°, -2° in 0°, glede na centralno vertikalno vodilno črto. Na slikah 6.1, 6.2, 6.3, 6.4, 6.5 in 6.6 sta prikazana grafa intenzitet (zgornja krivulja) in prvega odvoda intenzitet svetlobe (spodnja krivulja na grafu). Na x osi je prikazan odmik od centralne horizontalne vodilne črte v stopinjah, na y osi pa intenziteta svetlobe. Za točko prehoda svetlo-temne meje se upošteva maksimum prvega odvoda intenzitet svetlobe. Začetek in konec svetlo-temne meje na slikah nista prikazana.



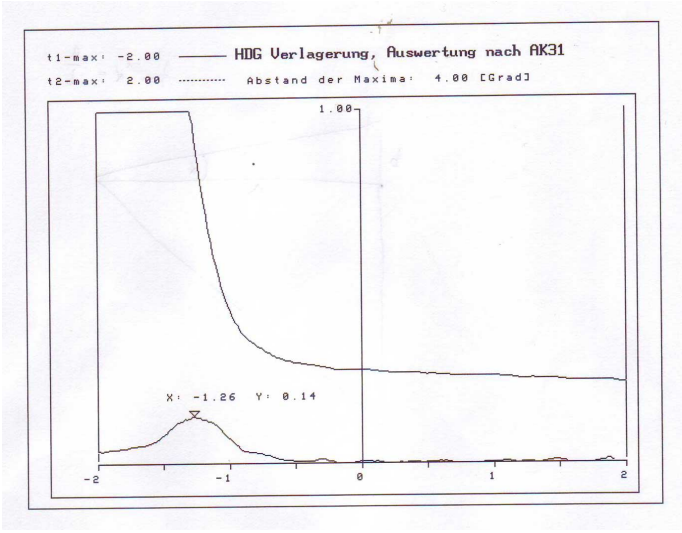
Slika 6.1: Meritev fotometra, desni žaromet, pozicija 0°.



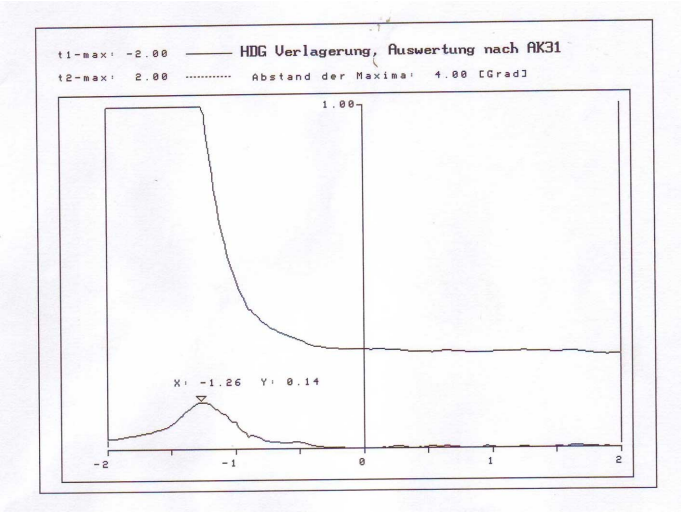
Slika 6.2: Meritev fotometra, desni žaromet, pozicija +2°.



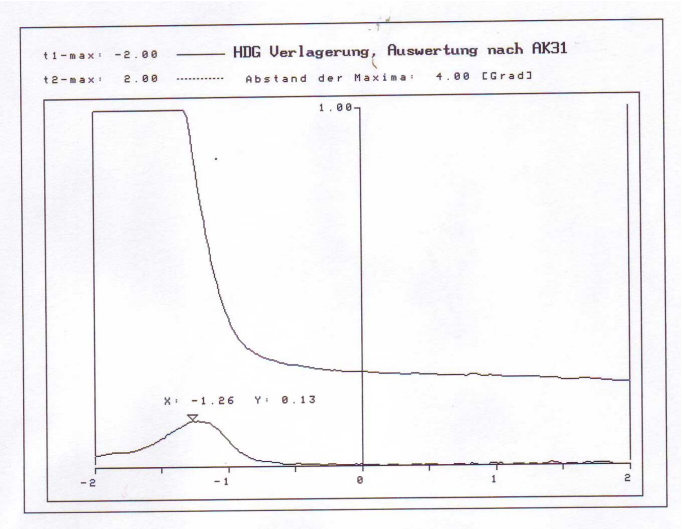
Slika 6.3: Meritev fotometra, desni žaromet, pozicija -2°.



Slika 6.4: Meritev fotometra, levi žaromet, pozicija 0°.



Slika 6.5: Meritev fotometra, levi žaromet, pozicija +2°.



Slika 6.6: Meritev fotometra, levi žaromet, pozicija -2°.

Literatura

[1] Klaus Engel, *Real-time volume graphics*, str. 112–114, A. K. Peters, 2006.

[2] Eric Freeman, Elisabeth Robson, Bert Bates, Kathy Sierra, *Head First Design Patterns*, O'Reilly, 2004.

[3] Rafael C. Gonzalez, Richard E. Woods, *Thresholding. In Digital Image Processing*, str. 595–611, Pearson Education, 2002.

[4] Charles A. Poynton, *Digital Video and HDTV: Algorithms and Interfaces*, Morgan Kaufmann, 2003.

[5] Linda Shapiro, George Stockman, *Computer Vision*, Prentice-Hall Inc., 2001.

[6] Andrew Stellman, Jennifer Greene, *Head First C#*, O'Reilly, 2007.

[7] (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

[8] (2012) Aforge.NET Houghova transformacija. Dostopno na:
http://www.aforgenet.com/framework/features/hough_transformation.html
http://en.wikipedia.org/wiki/Hough_transform

[9] (2012) BeanstalkD. Dostopno na:
https://github.com/kr/beanstalkd

[10] (2012) Finite difference. Dostopno na:
http://en.wikipedia.org/wiki/Finite_difference

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

[12] (2012) Fotometer podjetja Hella Saturnus d.o.o.. Dostopno na:
http://www.hella-saturnus.si/hella-si-si/278.html