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

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

[1] V dogovorju z mentorjem lahko kandidat diplomsko delo s pripadajočo izvorno kodo izda tudi pod katero izmed alternativnih licenc, ki ponuja določen del pravic vsem: npr. Creative Commons, GNU GPL. V tem primeru na to mesto vstavite opis licence, na primer tekst [5]

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 6 na strani 29.

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 celi 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 črno-beli sliki uporabi Sobelov filter, da se iz nje izlušči vodilne črte. Nad tej sliki se izvede še Hough-ova črtna transformacija, kjer se pridobi natačne koordinate črt.
Te se kasneje uporabi za izračun mer na sliki v kotih in centimetrih glede na centralne vertikalne in horizontalne črte (odmik).
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

This sample document presents an approach to typesetting your B.Sc. thesis using LaTeX. A proper abstract should contain around 100 words which makes this one way too short.

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 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-

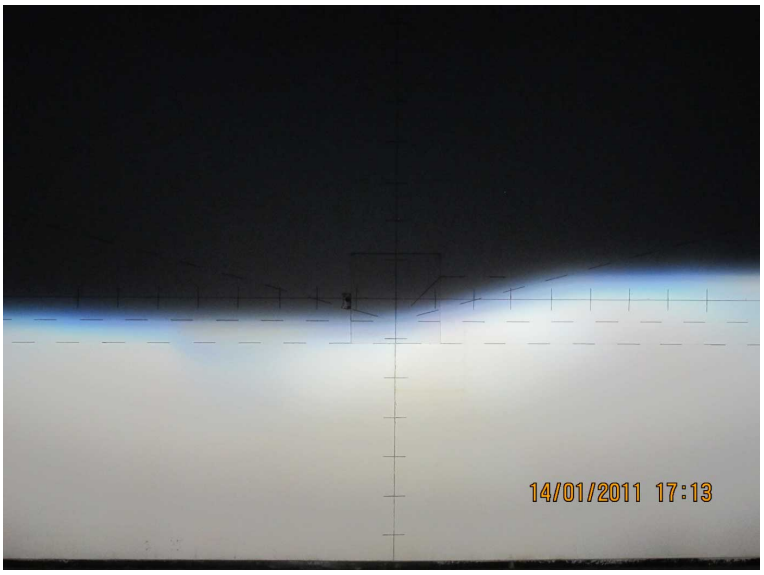
tlobe žarometa. Soba bo seveda še vedno potrebna. Manj jo pa bo potrebno zaščititi pred motnjami. Namesto fotometra pa potrebujemo samo fotoaparat ali kamero. Naš program mora tako omogočati vsaj isto funkcionalnost. Mora biti hitrejši in imeti isto natančnost rezultatov.

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 naših 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.

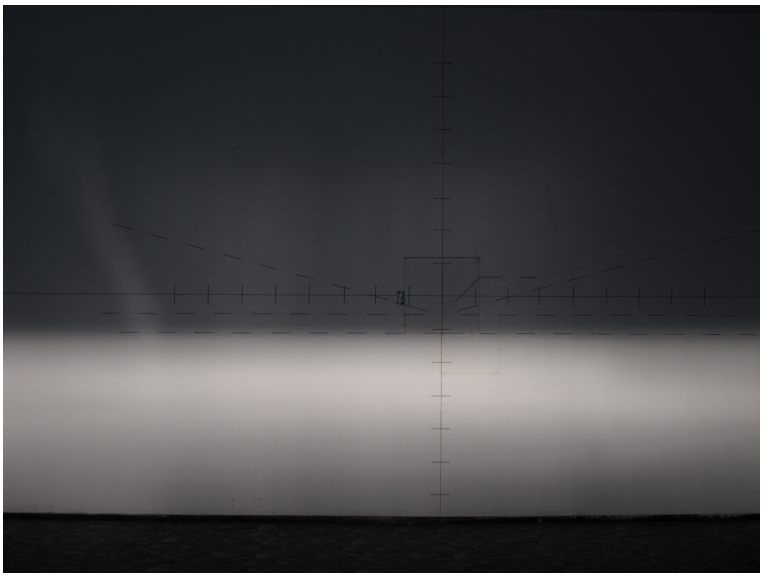
Poglavje 2

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 0 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 tri metre.
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.
- Instaliran .NET framework 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 pomnilnikov... 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.

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

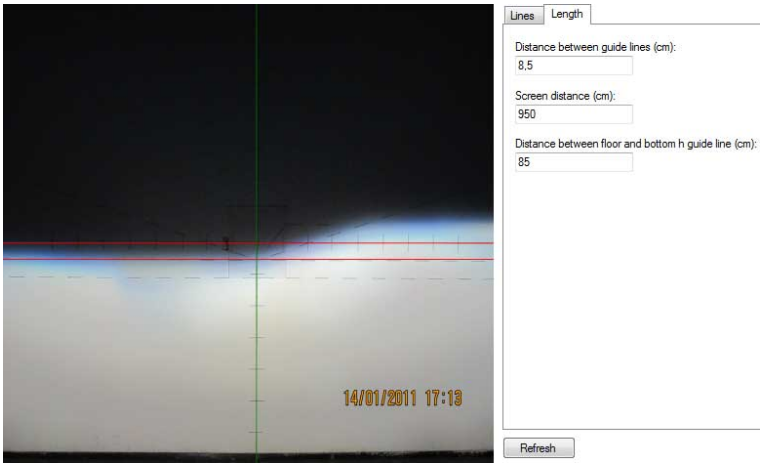
- Aplikacija bi morala biti v dveh delih. Uporabniški vmesnik, ki je napisan v PHP-ju in delu, ki bi procesiral slike.
- Nima podpore za niti/paralelno procesiranje.
- Slaba podpora za računalniški vid.

Slika 4.1: Uporabniški vmesnik programa.

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. Vsebuje tudi dosti pripomočkov za pisanje kode. Uporabil sem knjižnico za računalniški vid Aforge .Net, predvsem zaradi hitrega delovanja, 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).



Slika 4.2: Primer uporabniškega vmesnika za iskanje vodilnih črt.

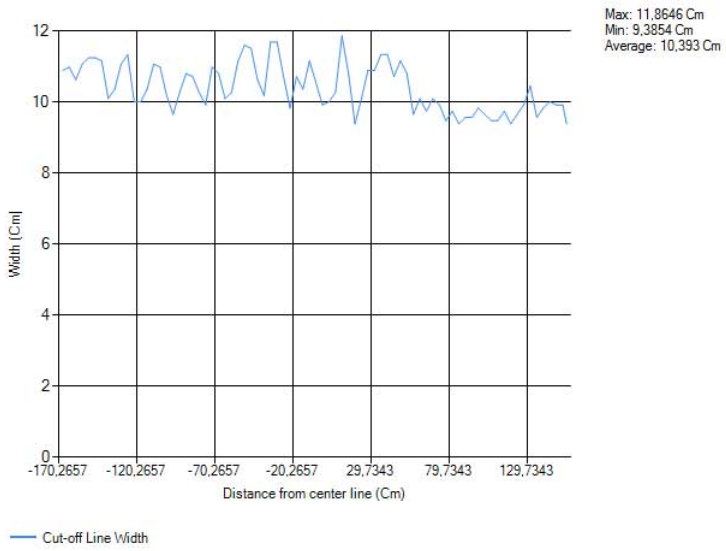
4.3.1 Vtičniki

Vmesnik je zasnovan na principu vtičnikov [4, 5]. 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();
}
```

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. 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



Slika 4.3: Primer uporabniškega vmesnika z grafom.

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 .Net knjižnic za Reflection [5] naložijo vse knjižnice. 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. Nakoncu se poveže še OnClick dogodek refresh gumba s 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 [4, 5]. Deluje na principu dogodkov. Pri tem modelu nekdo pošilja sporočila, ki lahko vsebujejo slike, mere, velikosti točk... Vsi, ki so prijavljeni na določeno sporočilo, ga prejmejo in obdelajo. Vsebuje dve funkciji:

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

Subscribe funkcija sprejme kot parametra tip sporočila 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, ki to sliko vsebuje. Naslednjemu vtičniku, ki je prijavljen na ta sporočila, se bo klicala callback metoda. Sliko bo pa dobil preko parametera v callback metodi. Ta vtičnik je v našem programu za pretvorbo v črno belo sliko. Po tem, ko sliko tudi sam pretvori 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.

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

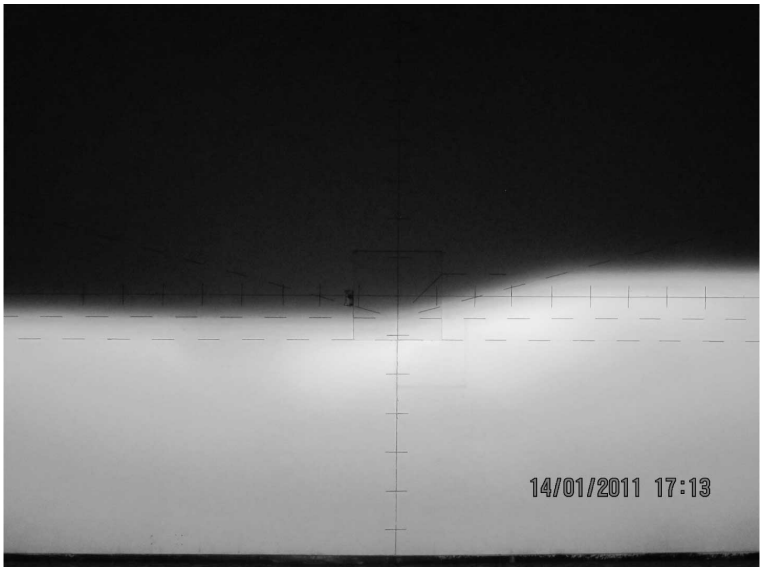
Kjer je M največja vrednost izmed RGB komponent točke. Vsaka točka je določena s tremi vrednostmi. Vsaka določa količino barve. R je za rdečo, G za zeleno in B za modro barvo. Te vrednosti so lahko od 0 do 255. Na tak način lahko opišemo barv. Spremenljivka m je najmanjša vrednost izmed RGB komponent. L je nova vrednost točke za črno-belo sliko. Trenutni algoritem se da pohitriti s paralelnim računanjem točk. Tako bi, pri 4 jedrnem procesorju, v enem koraku, izračunali 4 ali več točk hkrati. Rezultat tega koraka je slika 4.4.

4.4.2 Sobelov filter

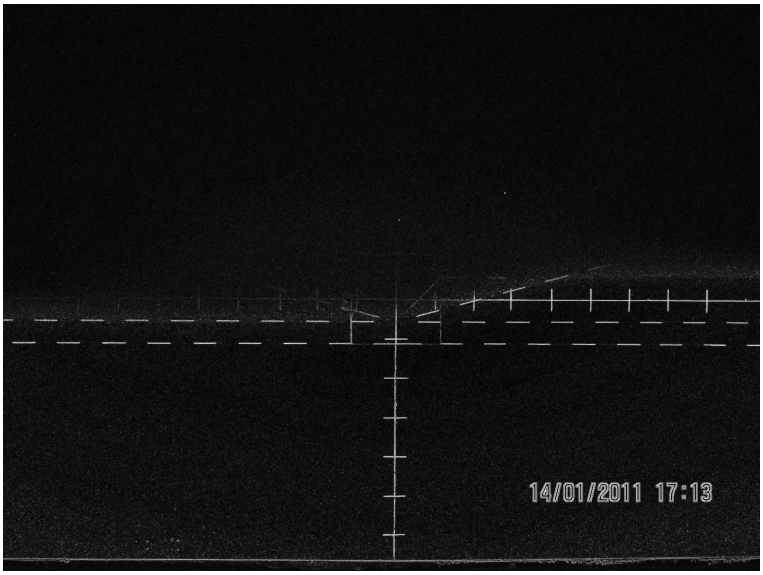
Drugi korak v našem postopku je uporaba Sobelovega filtra. Ta nam pomaga kasneje pri odkrivanju vodilnih črt na sliki. Tukaj je bila uporabljena knjižnica za računalniški vid Aforge.NET [1], kjer je ta filter že implementiran. Potreben je klic funkcije:

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

Rezultat filtra je slika 4.5.



Slika 4.4: Črno-bela slika.



Slika 4.5: Sobelov filter.

4.4.3 Prag

S tretjim korakom počistimo sliko iz poglavja Sobelov filter. To storimo tako, da za vsako točko pogledamo njeno intenziteto (pri 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 (od 0 do 255), potem na novo sliko vpišemo točko z maksimalno intenziteto (bela pika). Če je ta vrednost manjša od izbrane, vpišemo črno piko. 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 izgleda tako:

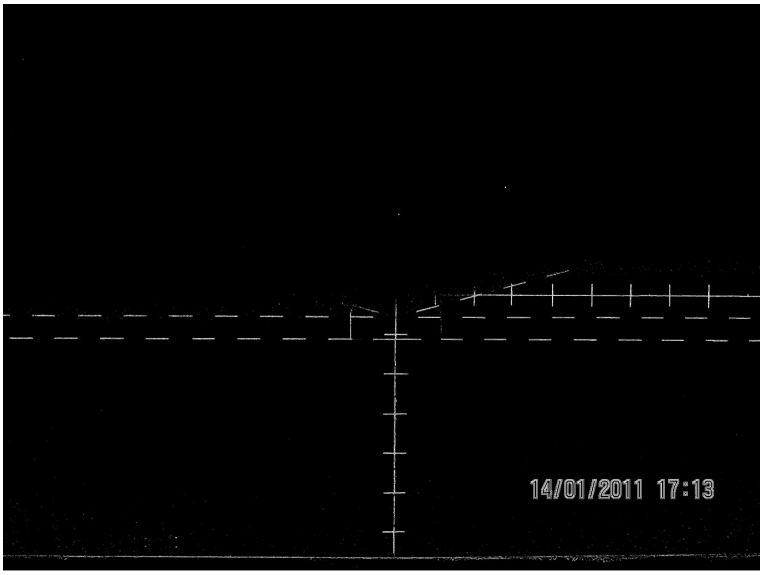
```
for (int i = 0; i < visina(slika); i++)
    for (int j = 0; i < sirina(slika); j++)
        if (slika[i][j].R < prag)
            rezultat[i][j].R, G, B = 0
        else
            rezultat[i][j].R, G, B = 255
```

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

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, 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 =
```



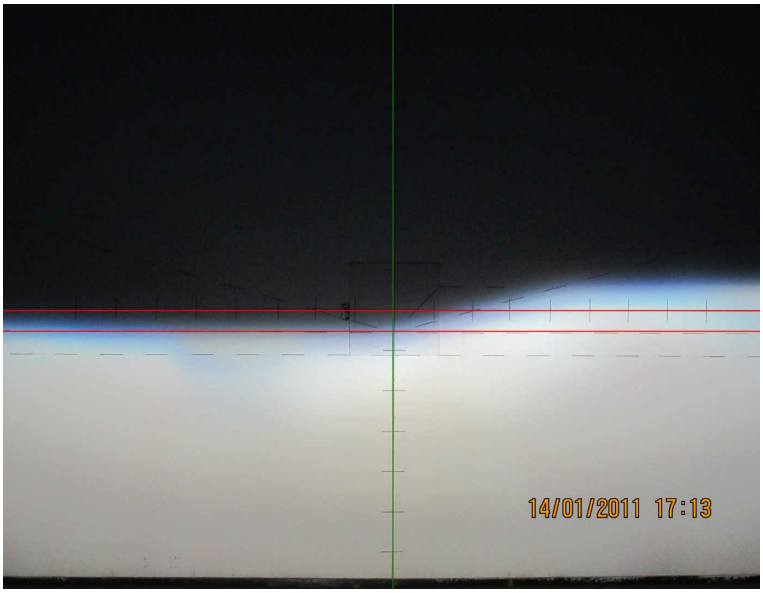
Slika 4.6: Prag.

```
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 (s tem kotnim pogojem najdemo vertikalne črte, za horizontalne črte 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. Tako je recimo črta na oddaljenosti 0 in kotom 90 zelo blizu naše prve horizontalne črte, ki jo iščemo. Črta z oddaljenostjo 0 in kotom 0 stopinj pa blizu naše 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.

$$v = \frac{a}{|r1 - r2|} \tag{4.2}$$



Slika 4.7: Vodilne črte.

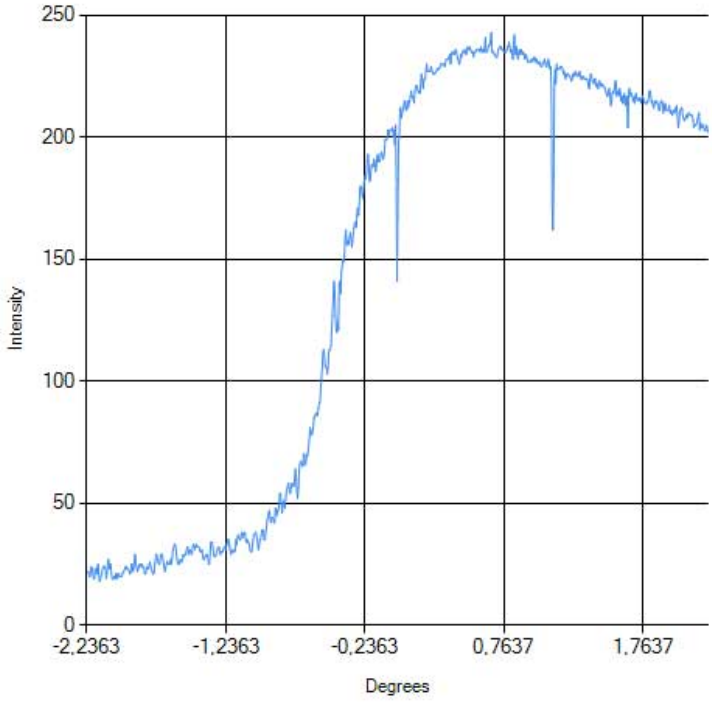
Kjer je v velikost točke v cm. Spremenljivka a je razdalja v cm med horizontalnima vodilnima črtama. R1 in r2 sta pa razdalji od centralne točke na sliki (v točkah). Vsako točko na sliki lahko podamo tudi v kotu glede na centralno vertikalno ali horizontalno vodilno črto. Formula za izračun kota je formula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

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

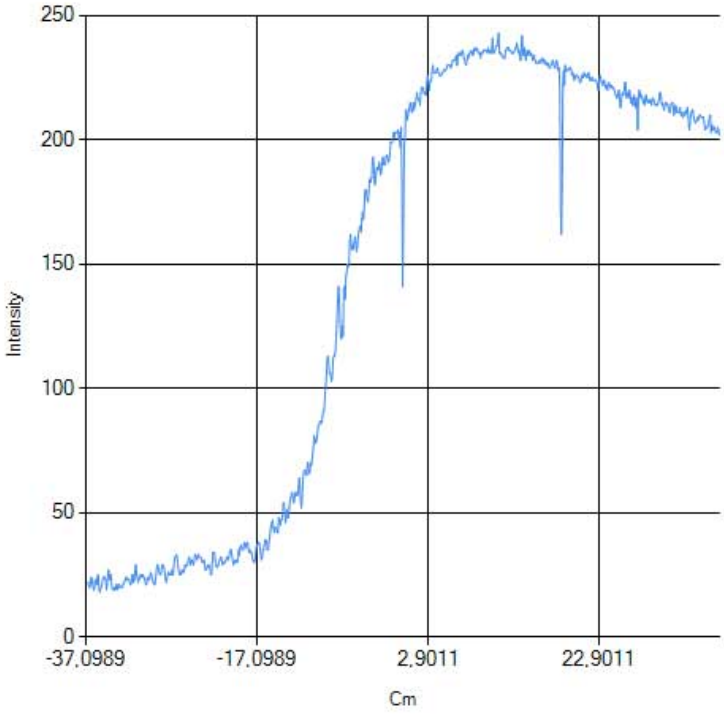
A je oddaljenost, v cm, trenutne točke od centralne vodilne črte. B je oddaljenost zaslona od kamere. Alfa je kot, ki ga iščemo.

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).



Slika 4.8: Intenzitete v stopinjah.



Slika 4.9: Intenzitete v centimetrih.

Os x grafu slika 4.8 predstavlja oddaljenost v stopinjah (4.9 v cm), od horizontalne vodilne črte. Na y osi je prikazana intenziteta točke. Primer tega 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 (intenziteta) v stolpcu x (naša vertikala). Pretvori koordinate vrstice i v izbrano enoto (stopinje ali centimetri). Na koncu shrani intenziteto in koordinate vrstice v seznam XY parov.

4.4.6 Iskanje svetlo-temne meje

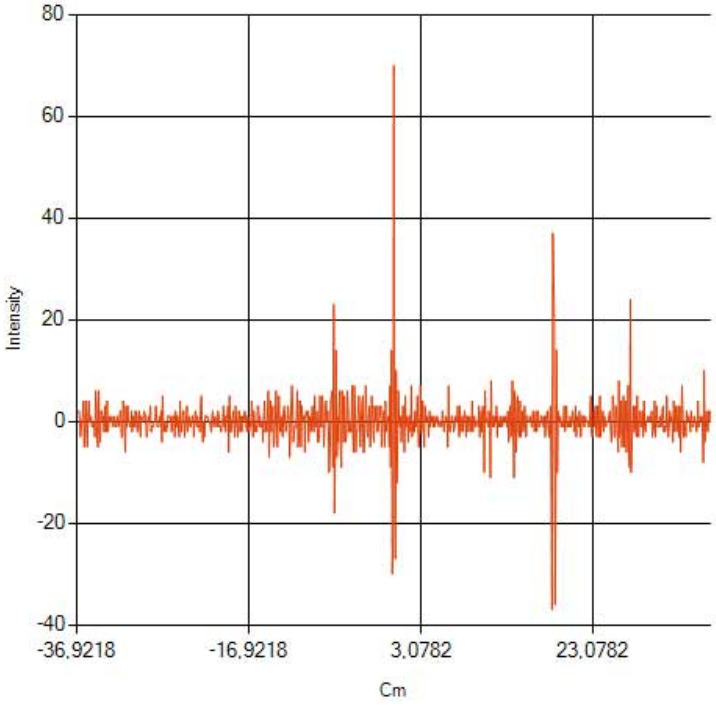
Za svetlo temno mejo vzamemo prevojno točko na grafu. Ta točka se nahaja kjer je drugi odvod funkcije 0. Za izračun drugega odvoda uporabimo formulo 4.4.

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

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) vrednosti tipa double. 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 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

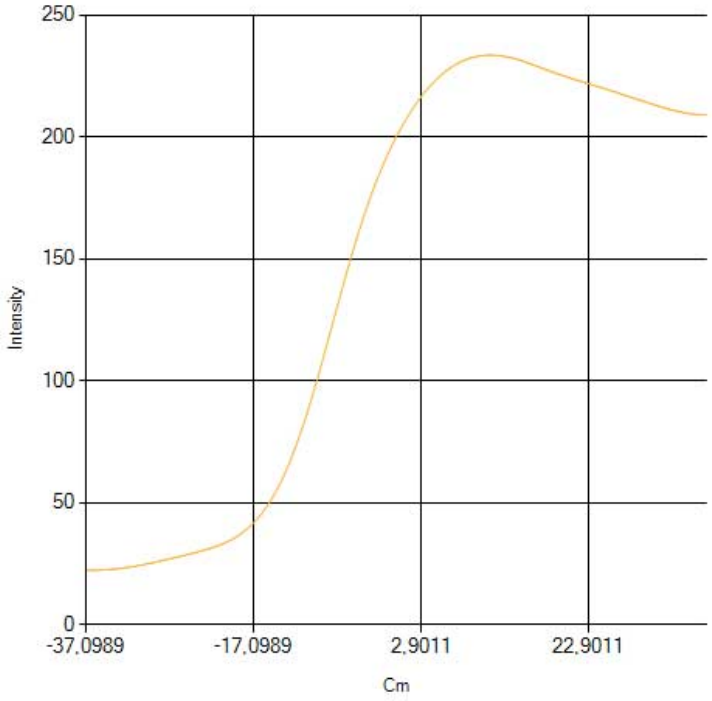


Slika 4.10: Drugi odvod funkcije.

smer in tako vsebuje veliko prevojnih točk. Ta problem rešimo tako, da našo originalno funkcijo zgladimo z algoritmom za glajenje funkcij.

4.4.7 Glajenje funkcije

V našem programu funkcijo gladimo tako, da jo povprečimo. Velikost okna (okno je število elementov, ki jih vzamemo v povprečje, ko računamo trenutni element) 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. 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



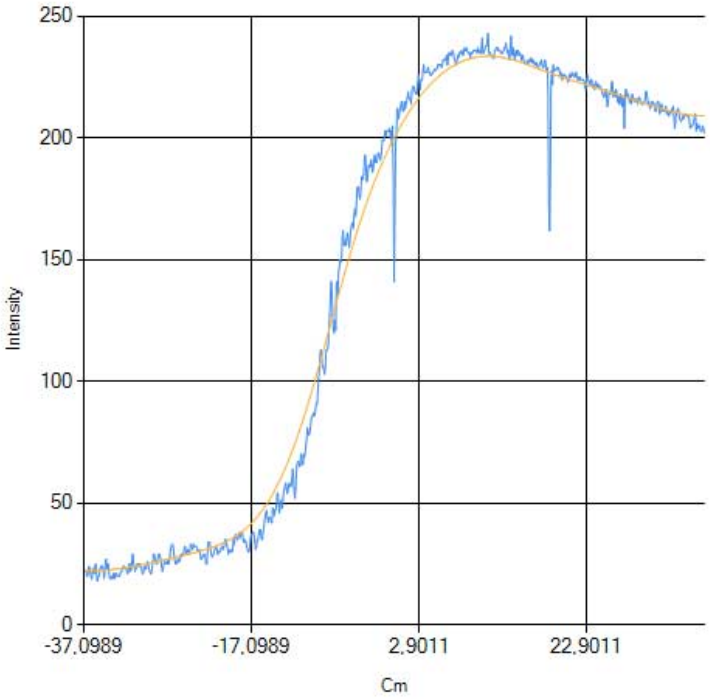
Slika 4.11: Zglajena funkcija.

nam elementov/točk v naši originalni funkciji ne zmanjka.

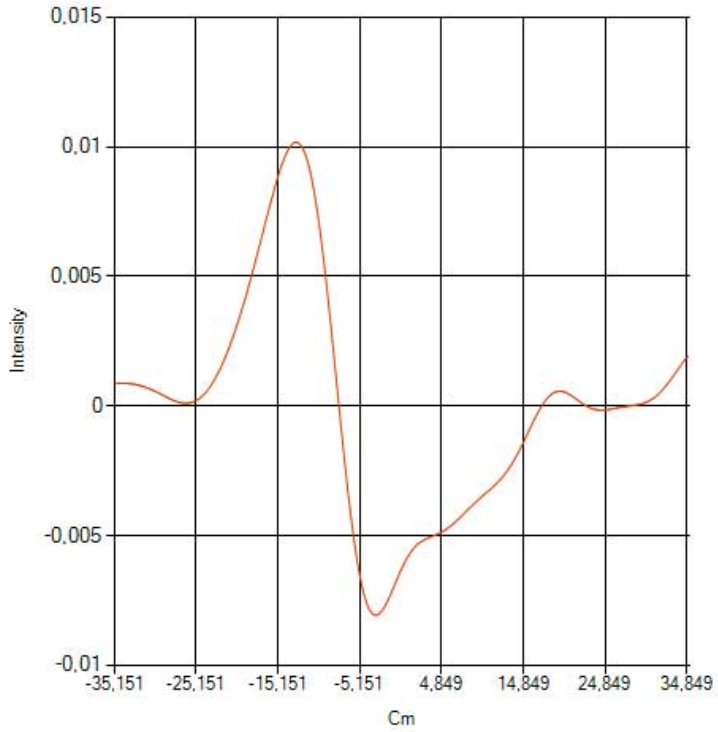
```
a = dolzina(f) / 2
for (i = a; i < dolzina(f) { a; i++)
    sum = 0
    for (j = i { a; j < i + a; j++)
        sum += f[j].X
    r.dodaj(new XY(sum / (a * 2 + 1), f[i].Y))
```

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-



Slika 4.12: Primerjava zglajene in originalne funkcije.



Slika 4.13: Drugi odvod zglajene funkcije.

temne meje.

Poglavje 5

Sklicevanje na besedilne konstrukte

Matematična ali popolna indukcija je eno prvih orodij, ki jih spoznamo za dokazovanje trditev pri matematičnih predmetih.

Izrek 5.1 *Za vsako naravno število n velja*

$$n < 2^n. \tag{5.1}$$

Dokaz. Dokazovanje z indukcijo zahteva, da neenakost (5.1) najprej preverimo za najmanjše naravno število — 0. Res, ker je $0 < 1 = 2^0$, je neenačba (5.1) za $n = 0$ izpolnjena.

Sledi indukcijski korak. S predpostavko, da je neenakost (5.1) veljavna pri nekem naravnem številu n, je potrebno pokazati, da je ista neenakost v veljavi tudi pri njegovem nasledniku — naravnem številu $n+1$. Računajmo.

$$\begin{aligned} n + 1 &< 2^n + 1 && (5.2)\\ &\leq 2^n + 2^n && (5.3)\\ &= 2^{n+1} \end{aligned}$$

Neenakost (5.2) je posledica indukcijske predpostavke, neenakost (5.3) pa enostavno dejstvo, da je za vsako naravno število n izraz 2^n vsaj tako velik kot 1. S tem je dokaz Izreka 5.1 zaključen. □

Opazimo, da je LaTeX številko izreka podredil številki poglavja.

Poglavje 6

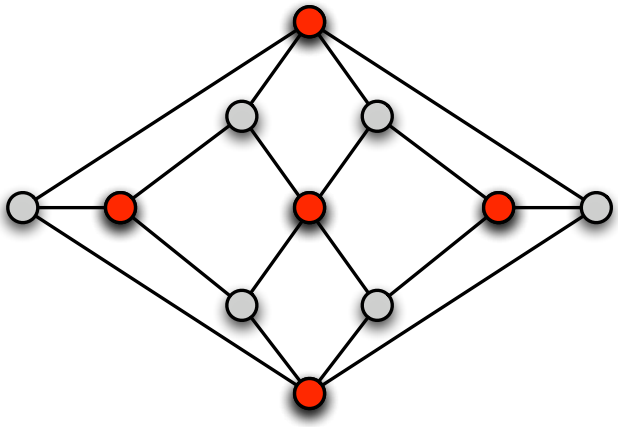
Plovke: slike in tabele

Slike in daljše tabele praviloma vključujemo v dokument kot plovke. Pozicija plovke v končnem izdelku ni pogojena s tekom besedila, temveč z izgledom strani. LaTeX bo skušal plovko postaviti samostojno, praviloma na vrh strani, na kateri se na takšno plovko prvič sklicujemo. Pri tem pa bo na vsako stran končnega izdelka želel postaviti tudi sorazmerno velik del besedila. V skrajnem primeru, če imamo res preveč plovk, se bo odločil za stran popolnoma zapolnjeno s plovkami.

6.1 Formati slik

Bitne slike, vektorske slike, kakršnekoli slike, z LaTeXom lahko vključimo vse. Slika 6.1 je v `.pdf` formatu. Pa res lahko vključimo slike katerihkoli formatov? Žal ne. Programski paket LaTeX lahko uporabljamo v več dialektih. Ukaz `latex` ne mara vključenih slik v formatu Portable Document Format `.pdf`, ukaz `pdflatex` pa ne prebavi slik v Encapsulated Postscript Formatu `.eps`. Strnjeno v Tabeli 6.1.

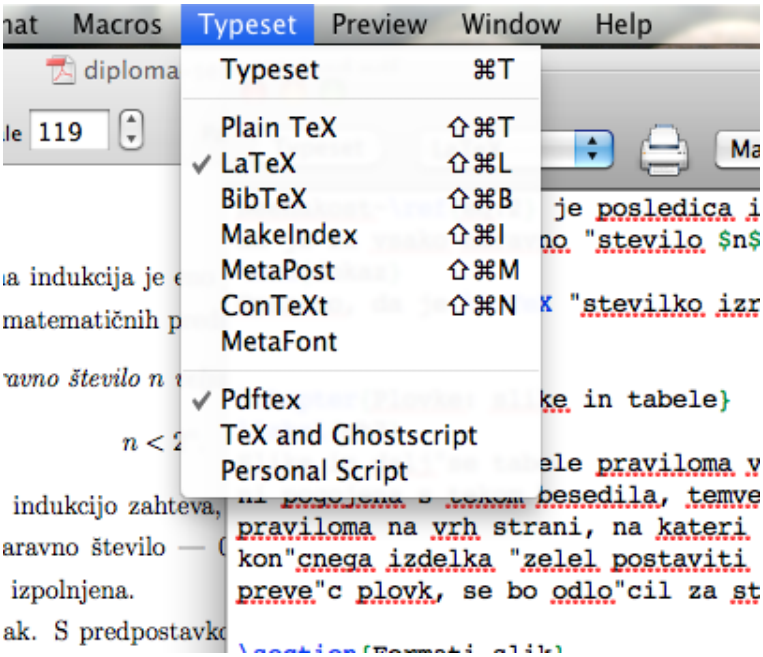
Nasvet? Odločite se za uporabo ukaza `pdflatex`. Vaš izdelek bo brez vmesnih stopenj na voljo v .pdf formatu in ga lahko odnesete v vsako tiskarno. Če morate na vsak način vključiti sliko, ki jo imate v `.eps` formatu, jo vnaprej pretvorite v alternativni format, denimo `.pdf`.



Slika 6.1: Herschelov graf, vektorska grafika.

ukaz/format	`.pdf`	`.eps`	ostali formati
`pdflatex`	da	ne	da
`latex`	ne	da	da

Tabela 6.1:



Slika 6.2: Kateri dialekt uporabljati?

Včasih se da v okolju za uporabo programskega paketa LaTeX nastaviti na kakšen način bomo prebavljali vhodne dokumente. Spustni meni na Sliki 6.2 odkriva uporabo LaTeXa v njegovi pdf inkarnaciji — `pdflatex`.

Vključena Slika 6.2 je seveda bitna.

Kaj pa stran iz študentskega referata? Tudi njo lahko vključimo v dokument. Toda ne kot plovko.

Poglavje 7

Kaj pa literatura

Kot smo omenili že v uvodu, je pravi način za citiranje literature uporaba BibTeXa [4]. Programski paket LaTeXje prvotno predstavljen v priročniku [3] in je v resnici nadgradnja sistema TeX avtorja Donalda Knutha, znanega po denimo, če izpustim njegovo umetnost programiranja, Knuth-Bendixovem algoritmu [2].

Vsem raziskovalcem s področja računalništva pa svetujem v branje mnenje L. Fortnowa [1].

Poglavje 8

Zaključek

Izbira LaTeX ali ne LaTeX je seveda prepuščena vam samim. Res je, da so prvi koraki v LaTeXu težavni. Ta dokument naj vam služi kot začetna opora pri hoji.

Literatura

[1] L. Fortnow, "Viewpoint: Time for computer science to grow up", *Communications of the ACM*, št. 52, zv. 8, str. 33–35, 2009.

[2] D. E. Knuth, P. Bendix. "Simple word problems in universal algebras", v zborniku: Computational Problems in Abstract Algebra (ur. J. Leech), 1970, str. 263–297.

[3] L. Lamport. *LaTEX: A Document Preparation System*. Addison-Wesley, 1986.

[4] O. Patashnik (1998) BibTEXing. Dostopno na: http://ftp.univie.ac.at/packages/tex/biblio/bibtex/contrib/doc/btxdoc.pdf

[5] licence-cc.pdf. Dostopno na: