

---

# **test\_docs Documentation**

***Release 23.2.2021***

**iovius**

**Feb 24, 2021**



## CONTENTS

<b>1 Prostorsko fitriranje in razširjevalnik laserskega snopa</b>	<b>3</b>
<b>2 Preračun optičnega sistema s programom OSLO LT</b>	<b>5</b>
<b>3 Fizeauev interferometer</b>	<b>19</b>
<b>4 Twyman-Greenov interferometer</b>	<b>21</b>
<b>5 Uvod v računalniško obdelavo slik</b>	<b>23</b>
<b>6 Računalniška analiza interferogramov</b>	<b>35</b>
<b>7 Razvoj programske opreme laserskega 3D merilnika</b>	<b>37</b>
<b>8 Umeritev laserskega 3D merilnika</b>	<b>43</b>
<b>9 Obdelava 3D meritev</b>	<b>47</b>
<b>10 Vlakenski senzorji</b>	<b>53</b>



V tem dokumentu je zbrano gradivo laboratorijske vaje pri predmetu LMS. Dokument je namenjen branju v spletnem brskalniku, kjer tudi delujejo vse povezave. V osnovi je 9 vaj, od tega je ena (Računalniška analiza interferogramov) poteka 2 tedna.



## PROSTORSKO FITRIRANJE IN RAZŠIRJEVALNIK LASERSKEGA SNOPA

Navodila za laboratorijsko vajo so zaenkrat dostopna le v *.pdf* verziji na povezavi Prostorsko filtriranje in razširjevalnik žarek

Pri tej vaji boste uporabljali polariziran *He-Ne* laser z valovno dolžino 633 nm in premerom žarka 0,8 mm.

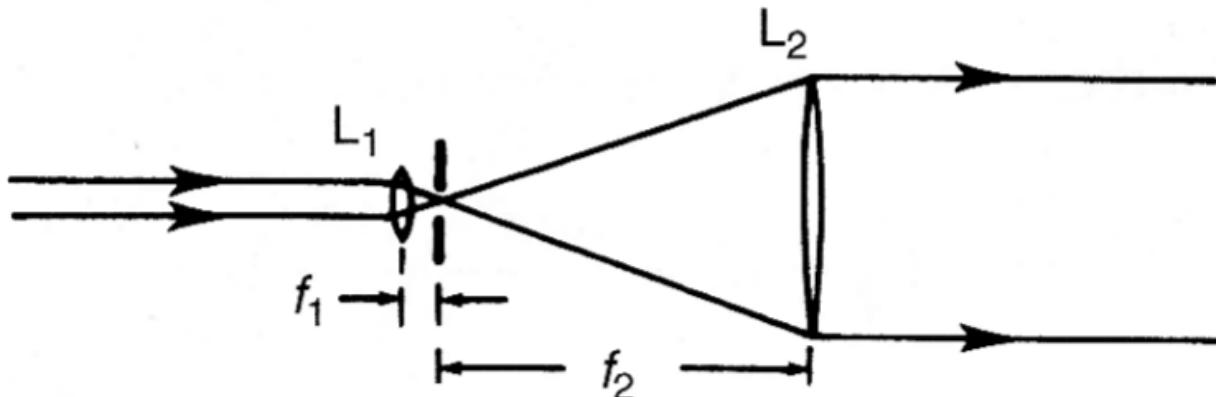


Fig. 1: Slika 1. Razširjevalnik žarka.

### 1.1 Naloge

1. Naravnajte *Ne-Ne* laser tako, da bo vzporeden z optično letvijo. Za preverbo vzporednosti uporabite zaslонko, ki ima odprtino približno 1/2 premera laserskega žarka. Zaslonko najprej postavite tik za laser in naravnajte izhodni konec laserja tako, da bo skozi zaslonko šlo čim več svetlobe. Nato zaslonko prestavite na drug konec letve in po istem postopku naravnajte drug konec laserja. Proceduro ponavljajte dokler ni laserski žarek naravnан tako, da sveti skozi zaslonko na katerikoli oddaljenosti od laserja.
2. Med zaslonko in laser postavite lečo z goriščno razdaljo 7,5 mm tako, da njeno gorišče sovpada z zaslonko. Pri tem pustite zaslonko in laser pri miru in v vseh treh oseh premikajte le lečo tako, da bo zopet šlo skozi zaslonko čim več svetlobe.
3. Sedaj odstranite zaslonko in fotografirajte intenzitetno porazdelitev laserskega žarka na zaslonu, ki je oddaljen nekaj metrov od leče. Opišite intenzitetni vzorec. Kakšen bi moral biti v idealnem primeru? Zakaj ni takšen? Primerjajte tudi teoretični in dejanski premer oziroma divergenco žarka? Za to uporabite fotografijo, ki ste jo posneli.

4. V gorišče leče tokrat postavite zaslонko, ki ima premer približno 1,5-krat večji od premera žarka v gorišču. Zopet fotografirajte žarek in opišite novo stanje.
5. Preizkusite še ostale zaslone, ki so na voljo in pri vsaki izmerite premer prvega temnega Airy-jevega diska. Na osnovi teh izmerkov izračunajte premere zaslonov in jih primerjajte z dejanskimi.
6. Sestavite razširjevalnik žarka z ustrezno zaslono ( $d_{zas}=1,5 d_f$ ), tako da dodate drugo lečo z goriščnico  $f = 150\text{ mm}$ . Kje mora stati, da bo na izhodu žarek kolimiran? Preverite intenzitetni profil žarka  $z$  in *brez* zaslone.

## PRERAČUN OPTIČNEGA SISTEMA S PROGRAMOM OSLO LT

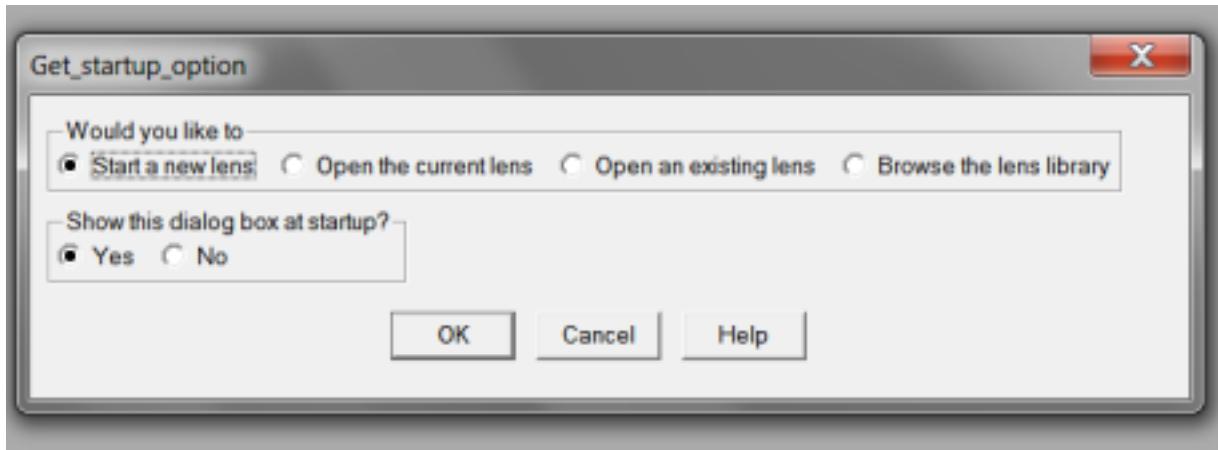
Navodila za laboratorijsko vajo so zaenkrat dostopna le v *.pdf* verziji na povezavi Preračun optičnega sistema s programom OSLO LT

### 2.1 Naloge

1. Zmodelirajte optični sistem z dvema lečama.
2. Vizualizirajte optični sistem in izrišite potek žarkovnega stožca skozi optični sistem.
3. Okarakterizirajte kvaliteto preslikave točkovnega svetlobnega izvora.
4. Naredite primerjavo treh tipov leč za primer preslikave iz neskončnosti v končnost.
5. Zmodelirajte razširjevalnik žarka.

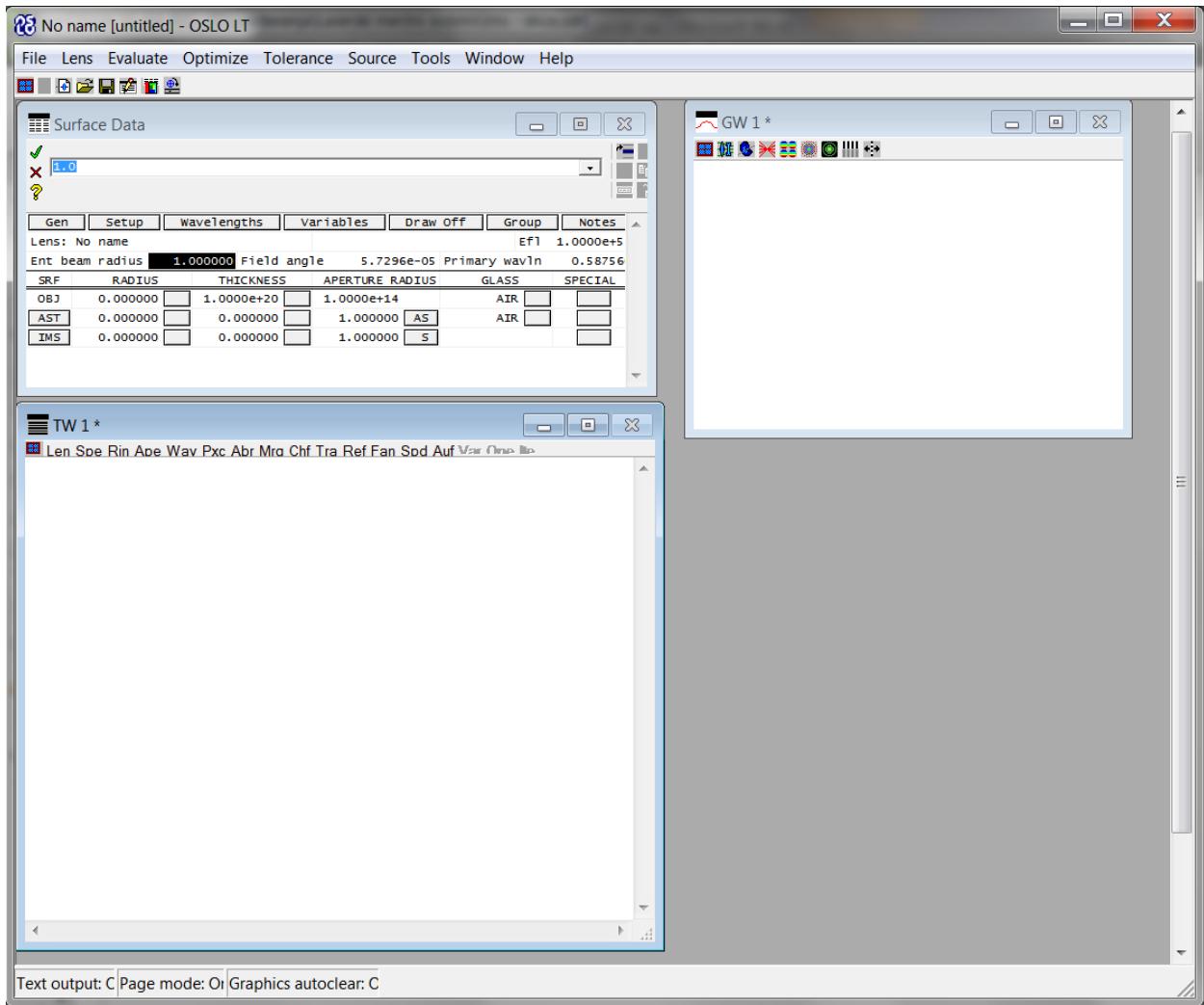
### 2.2 Zagon programa

1. Ob zagonu izberite:



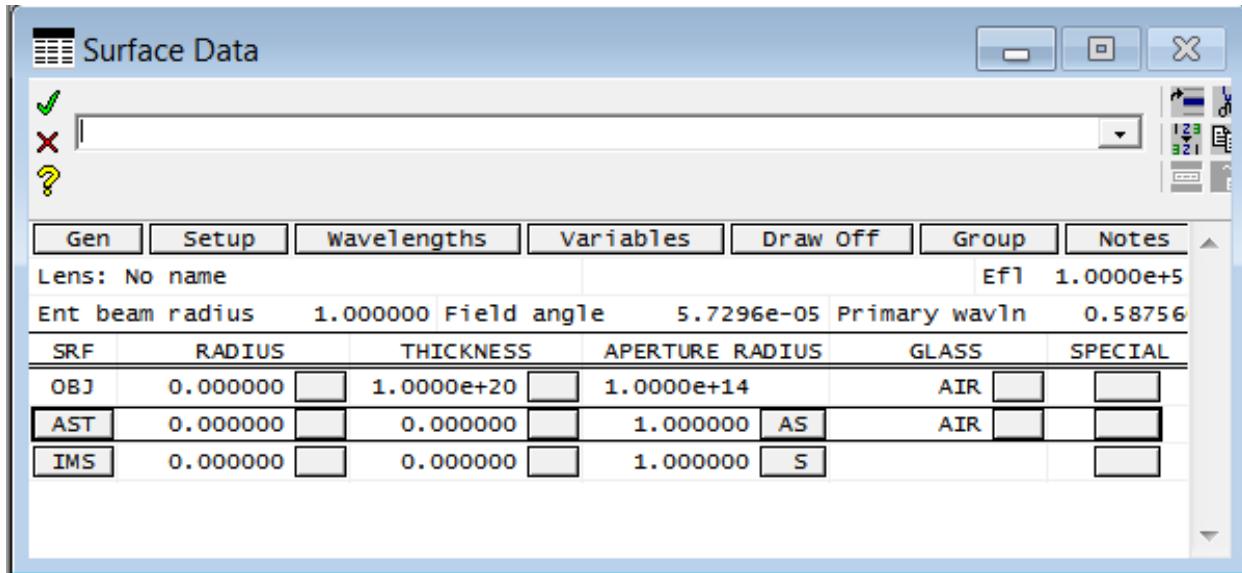
2. Po potrditvi (*OK*) se v programu pojavijo tri podokna (*Surface Data*, *TW* in *GW*):

Zadnji dve služita tekstovnemu (*TW*) in grafičnemu prikazu (*GW*) rezultatov. Okno *Surface Data* je v prvi fazi modeliranja najpomembnejše, saj vanj vnašamo geometrijske in optične podatke proučevanega lečja.



## 2.3 Vnos geometrijskih podatkov

1. V oknu *Surface Data* so torej shranjeni podatki optičnega sistema: ime optičnega sistema:



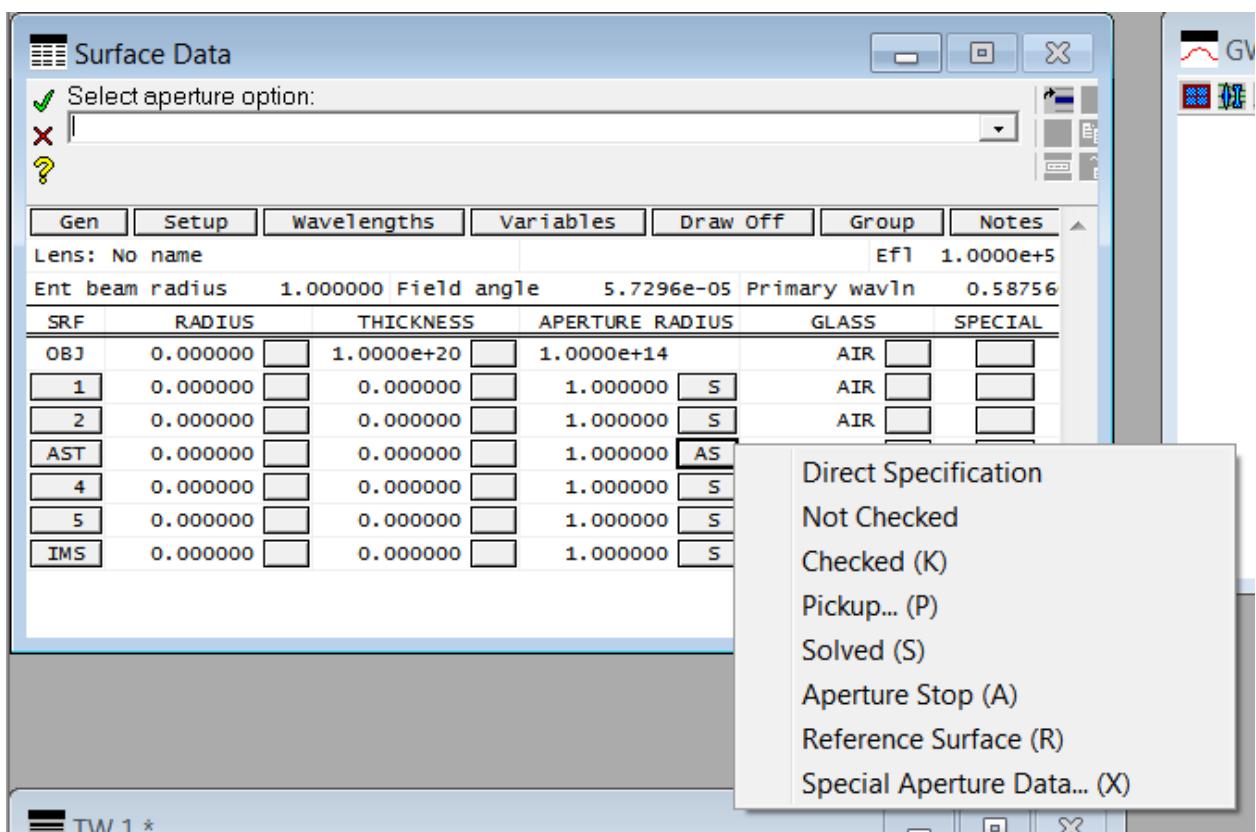
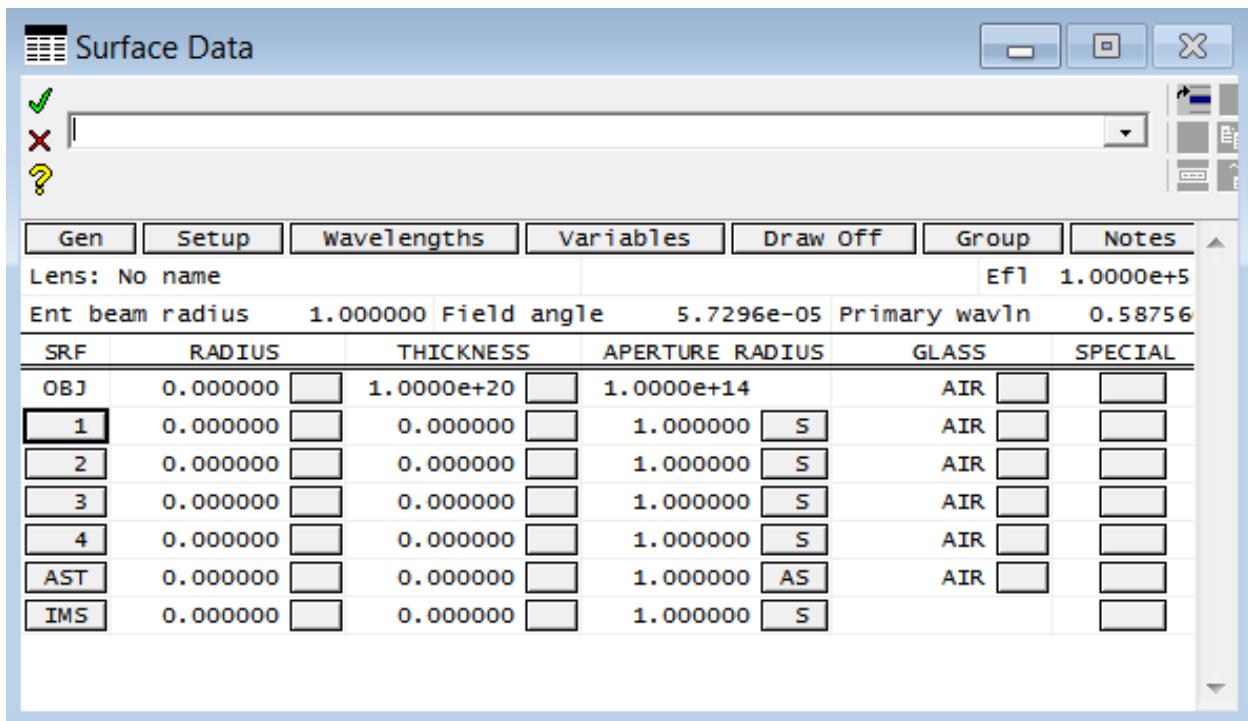
V tabeli površin so ob pričetku že tri površine in sicer: objektna površina (*OBJ*), zaslonka (*AST*) in slikovna ravnina (*IMS*).

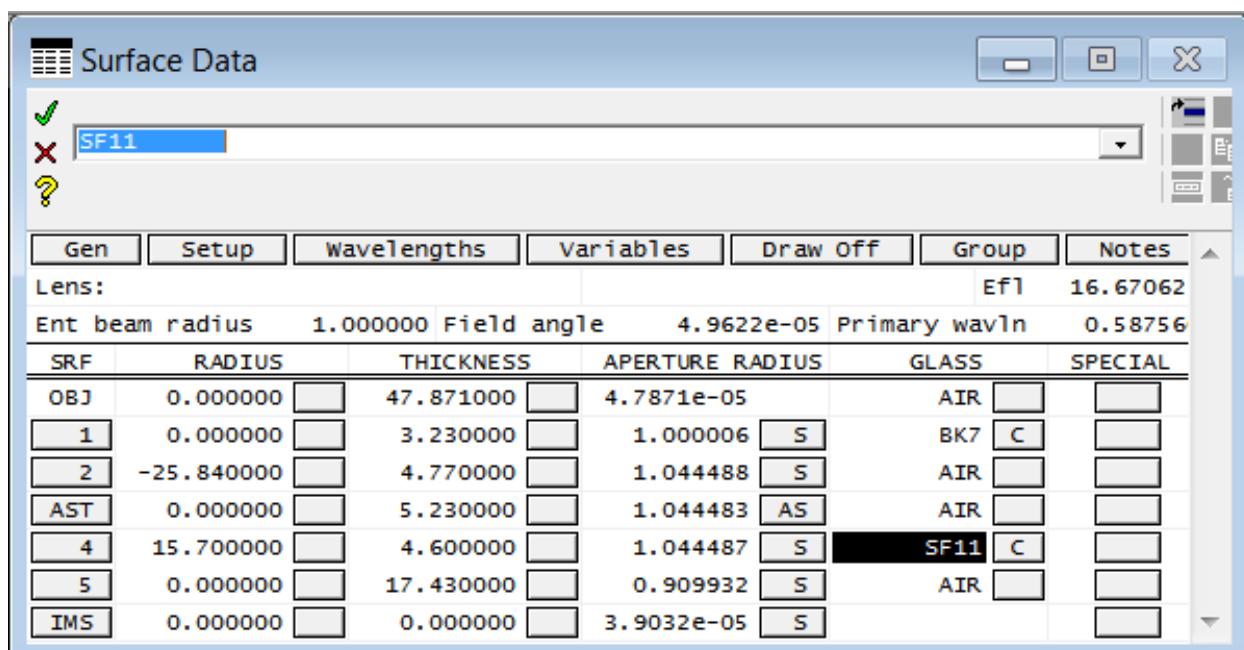
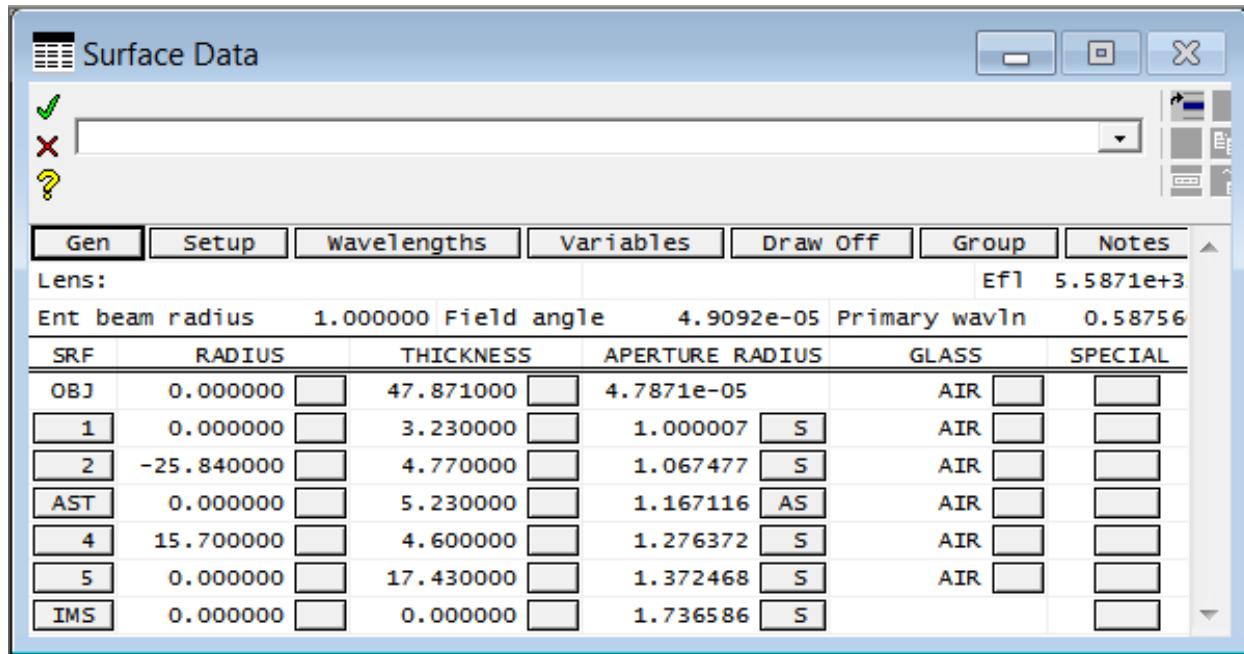
- Nove površine vnesemo z desnim klikom in ukazom **Insert Before** oziroma **Insert After**. Tako vnesite štiri nove površine med *OBJ* in *AST*. Tabela mora ob koncu imeti sledeč izgled:
- Določite 3. površino kot zaslonko. To storite z desnim klikom na gumb v stolpcu **APERTURE RADIUS**:
- Vnesite razdalje med posameznimi površinami tako, da spremenite vrednosti v stolpcu **THICKNESS**. Razdaljo med objektno ravnino in 1. lečo vnesete v prvo vrstico (*OBJ*). Debelino 1. leče vnesete kot debelino 1. površine. Tako nadaljujete z vsemi debelinami, vključno z debelino 5. površine, ki popisuje razdaljo med zadnjo optično površino in slikovno ravnino.
- Vnesite tudi krivinske radije posameznih površin. Zato spremenite vrednosti v stolpcu **RADIUS**... note:: Pri tem upoštevajte dogovor o predznaku: pozitiven, če je center sfere na desni strani površine!

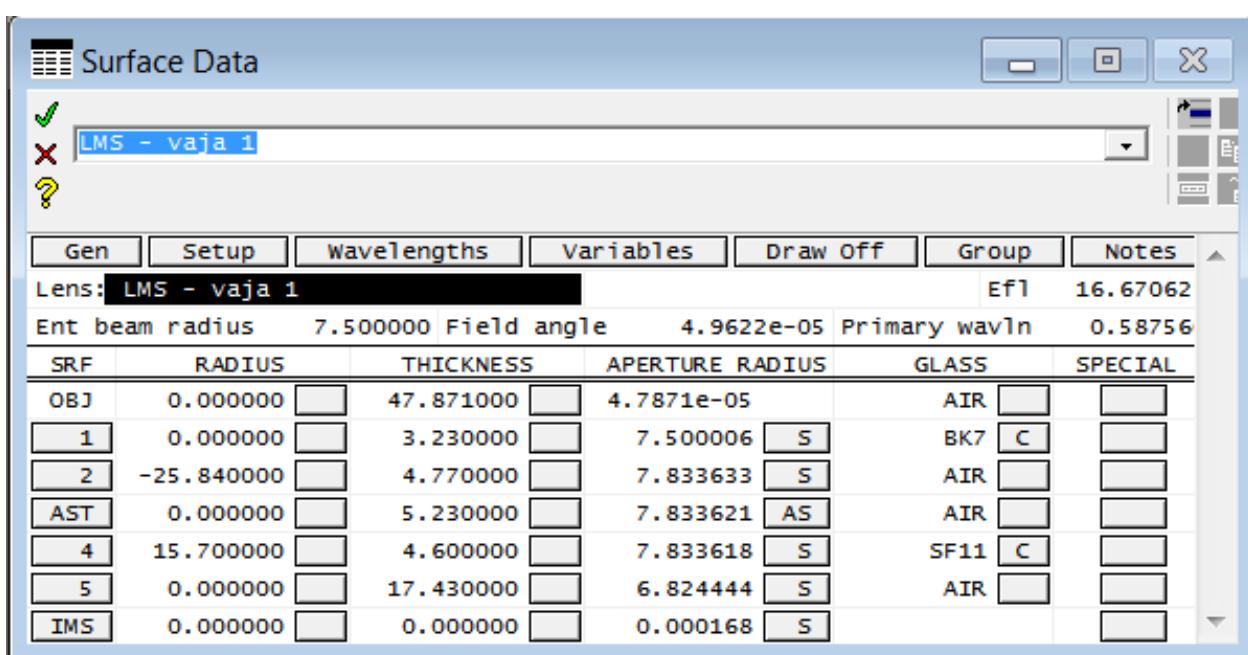
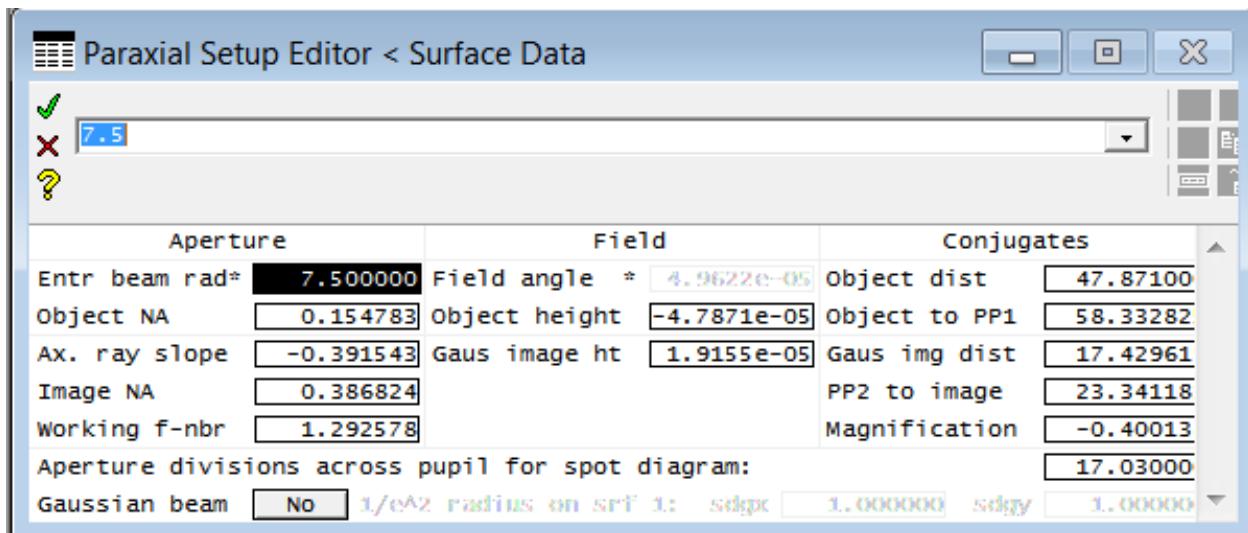
Ob koncu mora imeti tabela sledeč izgled:

- Dodajte še materialne podatke in sicer v stolpec **GLASS**. Prva leča je iz stekla z oznako *BK7*, zato v celico 1. površine vpišite *BK7*. Ob tem opazimo, da se je na gumbu desno pojavila oznaka **C**, kar pomeni, da je program oznako prepoznal in podatke črpa iz knjižnice o različnih steklih. Enako storimo tudi za drugo lečo, ki je iz stekla *SF11*:
- Nastavite polmer žarka na vstopu v prvo lečo. Pritisnite na gumb **SETUP** in spremenite parameter **Entr beam rad** na vrednost 7.5:
- Poimenujte optični sistem, tako da v okno *Surface Data*, v polje **Lens** vpišete *LMS – vaja 2*.

Vidimo, da tabela poleg stolpcev **RADIUS**, **THICKNESS** in **GLASS** vsebuje tudi stolpca **APERTURE RADIUS** in **SPECIAL**. Prvi stolpec, torej **APERTURE RADIUS**, določa polmer posamezne optične površine, ki se ob privzetem načinu samodejno izračuna glede na polmer žarkovnega stožca. Zato je na gumbih znak **S**, ki pomeni *Solved*. Stolpec **SPECIAL** pa služi za dodatne nastavitev posamezne površine, kot so na primer asferičnost, reflektivnost, ekscentričnost in, kot bomo videli v nadaljevanju, način grafičnega prikaza.



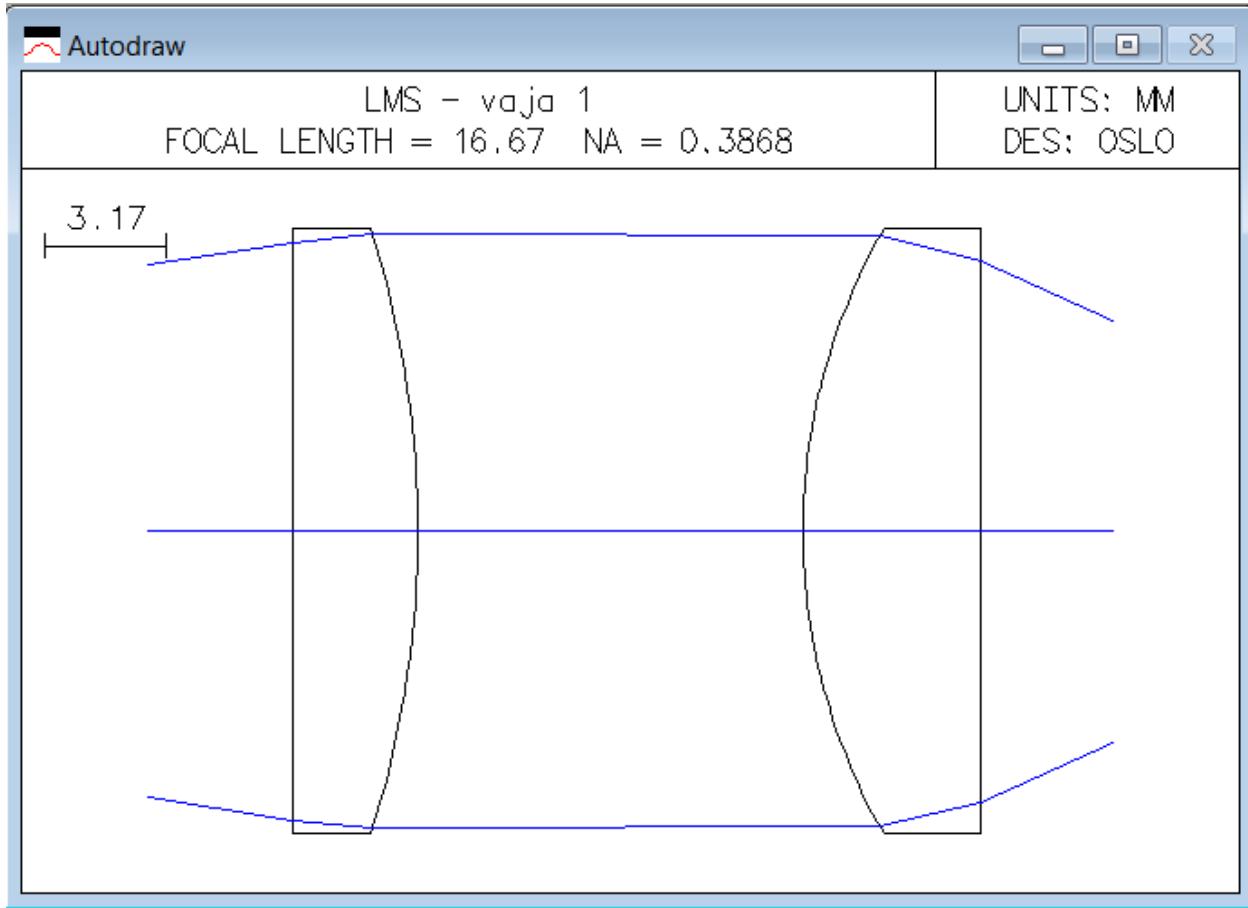




## 2.4 Izris optičnega sistema

Tako smo vnesli večino geometrijskih in materialnih podatkov. Čas je, da sistem izrišemo in vizualno preverimo ustreznost podatkov.

- V ta namen pritisnemo gumb **Draw Off**, ki se nahaja v oknu *Surface Data*. Po pritisku se gumb preimenuje v **Draw On**, izriše pa se tudi okno *Autodraw*:



V oknu se v osrednjem delu izriše stranski pogled na optični sistem ter centralni in marginalna žarka. V glavi okna (zgoraj) je izpisano ime optičnega sistema ter goriščna razdalja in numerična odprtina slikovnega prostora (NA). Pri tem se numerična odprtina izračuna po enačbi:

$$NA' = n' \cdot \sin\Phi'$$

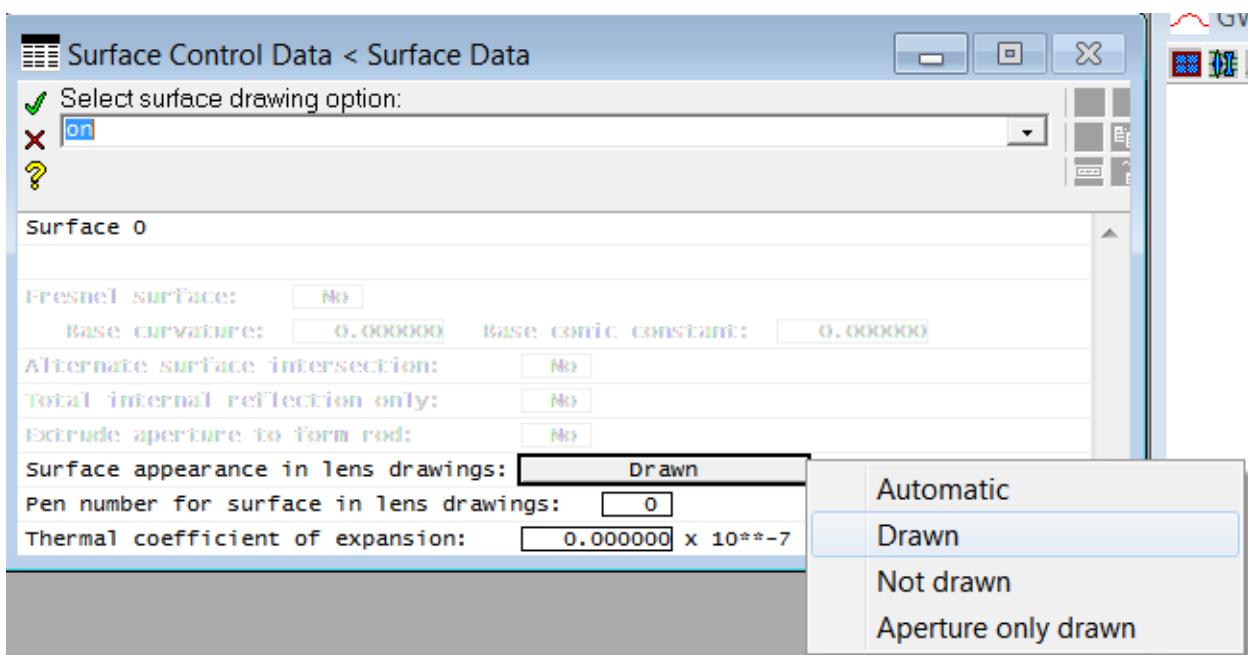
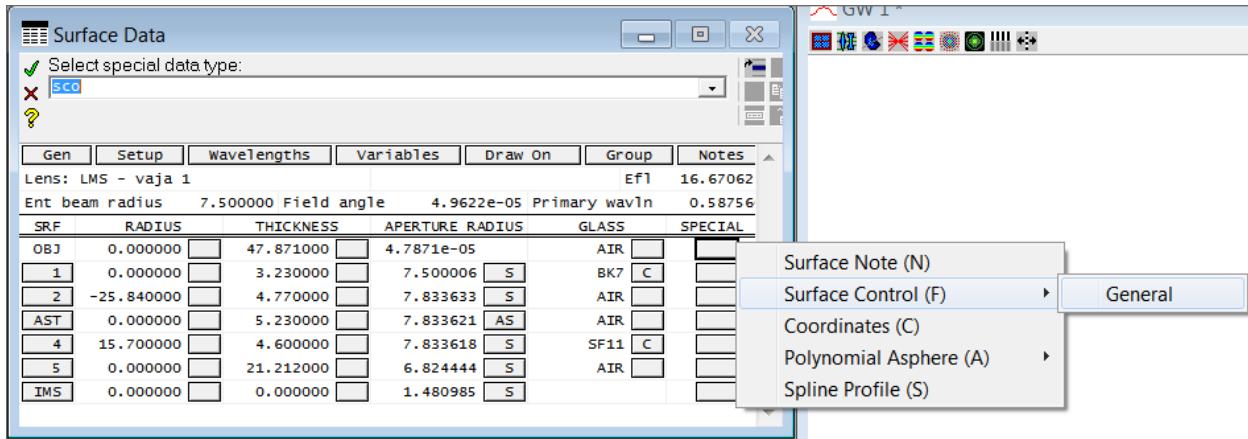
$n'$  ... lomni količnik medija med zadnjo lečo in slikovno ravnino,  $\Phi'$  ... prostorski kot žarkovnega snopa na izhodu iz optičnega sistema

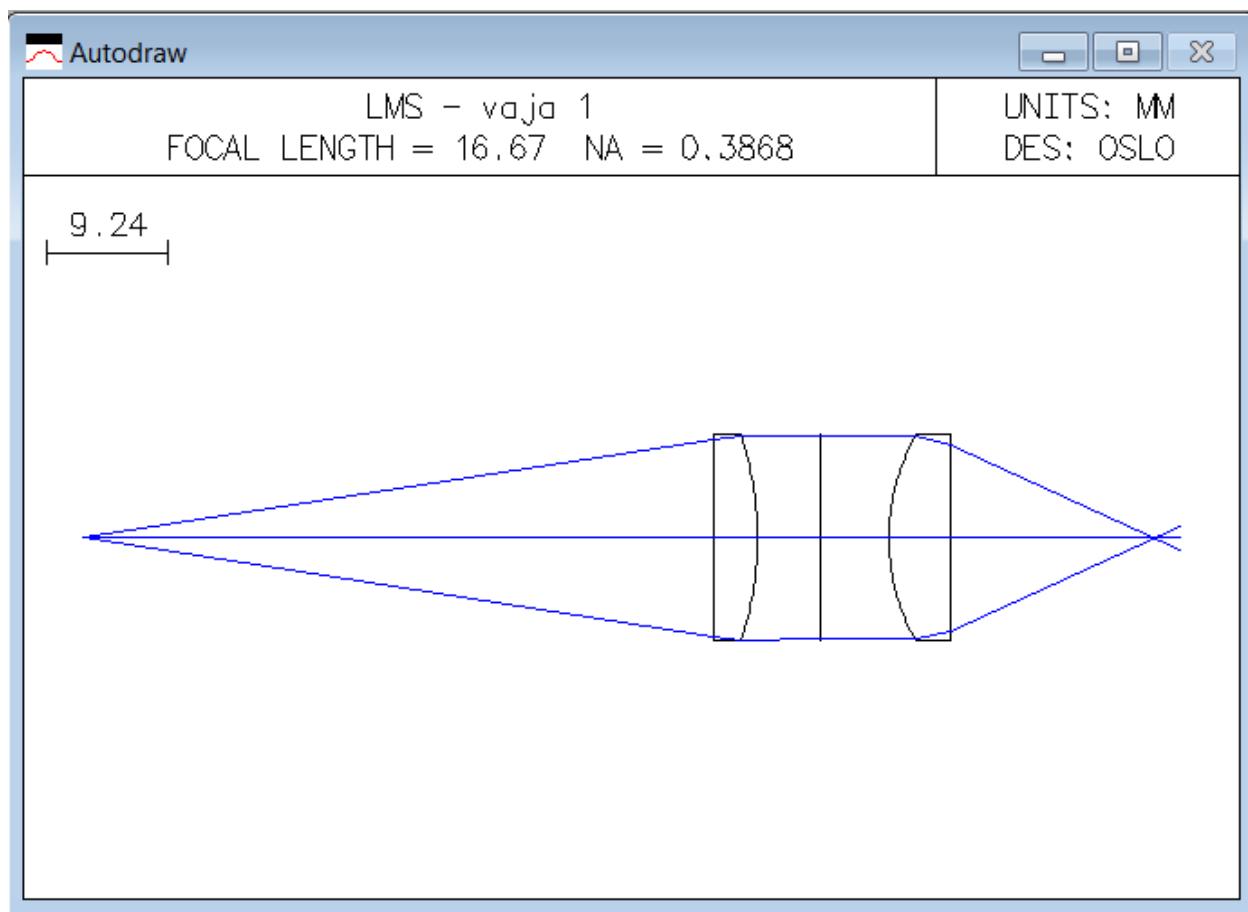
Okno *Autodraw* osveži geometrijo ob vsaki spremembi podatkov, ki so shranjeni v *Surface Data*. Kot vidimo, je privzet način izrisa takšen, da vidimo zgolj stranski pogled leč, brez zaslonke ter objektne in slikovne ravnine.

- Manjkajoče optične površine (zaslonko in objektno ter slikovno ravnino) izrišemo tako, da pritisnemo na gumb izbrane površine, v stolpcu **SPECIAL**, in izberemo *Surface Control*:

V oknu *Surface Data* poiščemo polje **Surface appearance in lens drawings** in izberemo možnost *Drawn*:

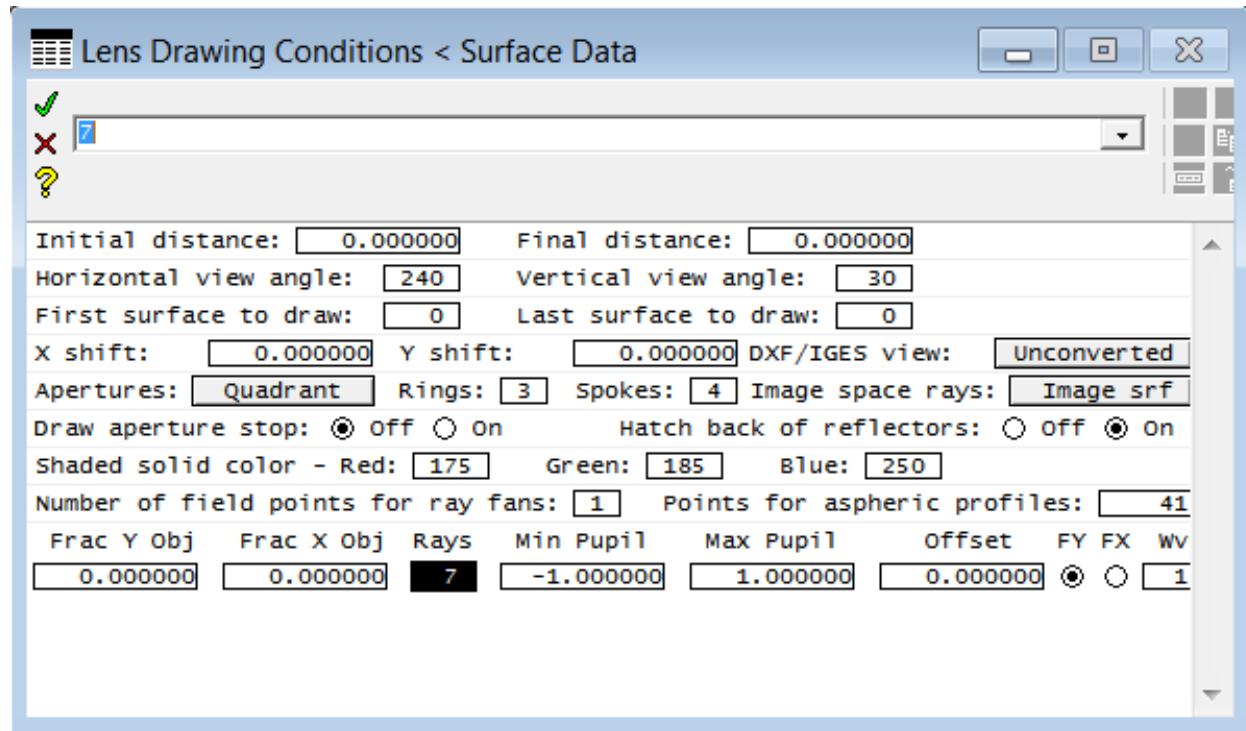
- Isti postopek ponovimo za ostali dve površini. Na koncu mora se mora v *Autodraw* oknu izrisati sledeča slika:





Sedaj vidimo, da se žarki širijo iz objektne točke, ki leži na optični osi, proti optičnemu sistemu, tako da ima snop na vstopu v 1. lečo premer 15 mm. Skozi sistem se žarki lomijo in imajo na izhodu skupno sečišče nekaj milimetrov pred slikovno ravnino. To sečišče ne sovпадa s slikovno ravnino, ker smo njeni lego izračunali na osnovi paraksialne aproksimacije, marginalna žarka pa oklepata že relativno velik kot z optično osjo.

- Nastavimo izris tako, da bomo poleg omenjenih treh žarkov videli še štiri vmesne. Izberemo meni *Lens->Lens drawing Conditions...*. Nato v oknu *Surface Data* nastavimo število objektnih točk (**Number of field points for ray fans**), iz katerih se širijo žarkovni snopi na 1. Naš optični sistem uporabljamo zgolj za preslikavo točkovnega svetlobnega izvora, ki leži na optični osi, zato je ena objektna dovolj. Nato pa v polju pod **Rays** vnesemo željeno število izrisanih žarkov, torej 7:



Spremenljivki **Min pupil** in **Max Pupil** določata relativno višino zgornjega in spodnjega žarka na vstopni odprtini in sicer glede na njen polmer. Po potrditvi sprememb se mora v Autodraw oknu izrisati sledeča slika:

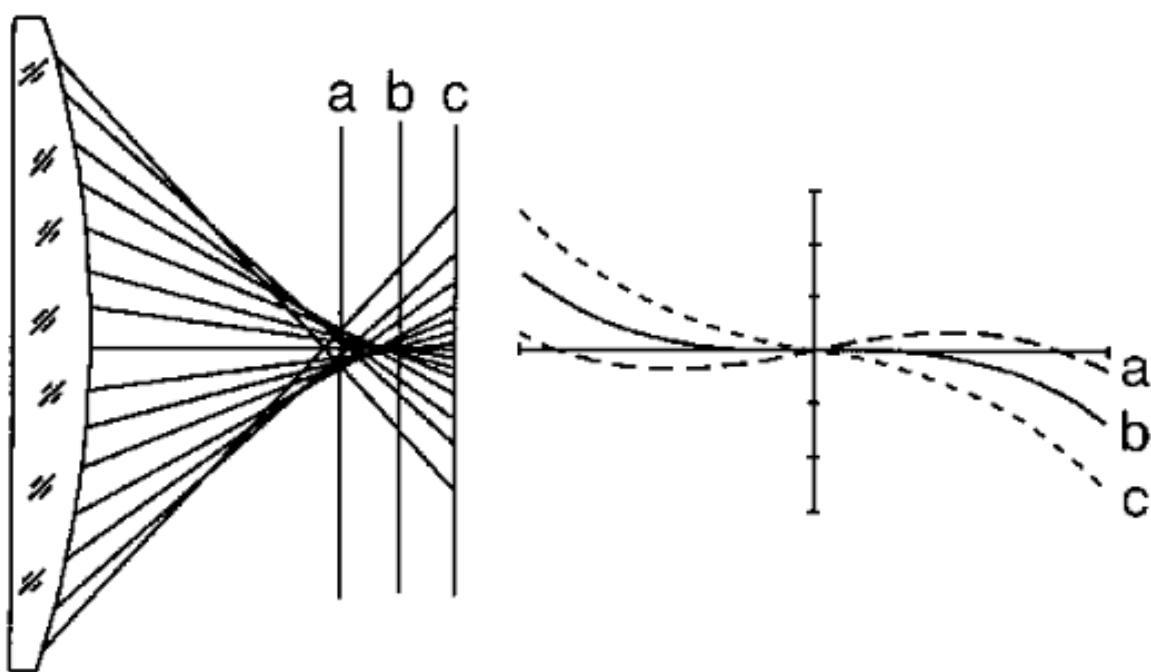
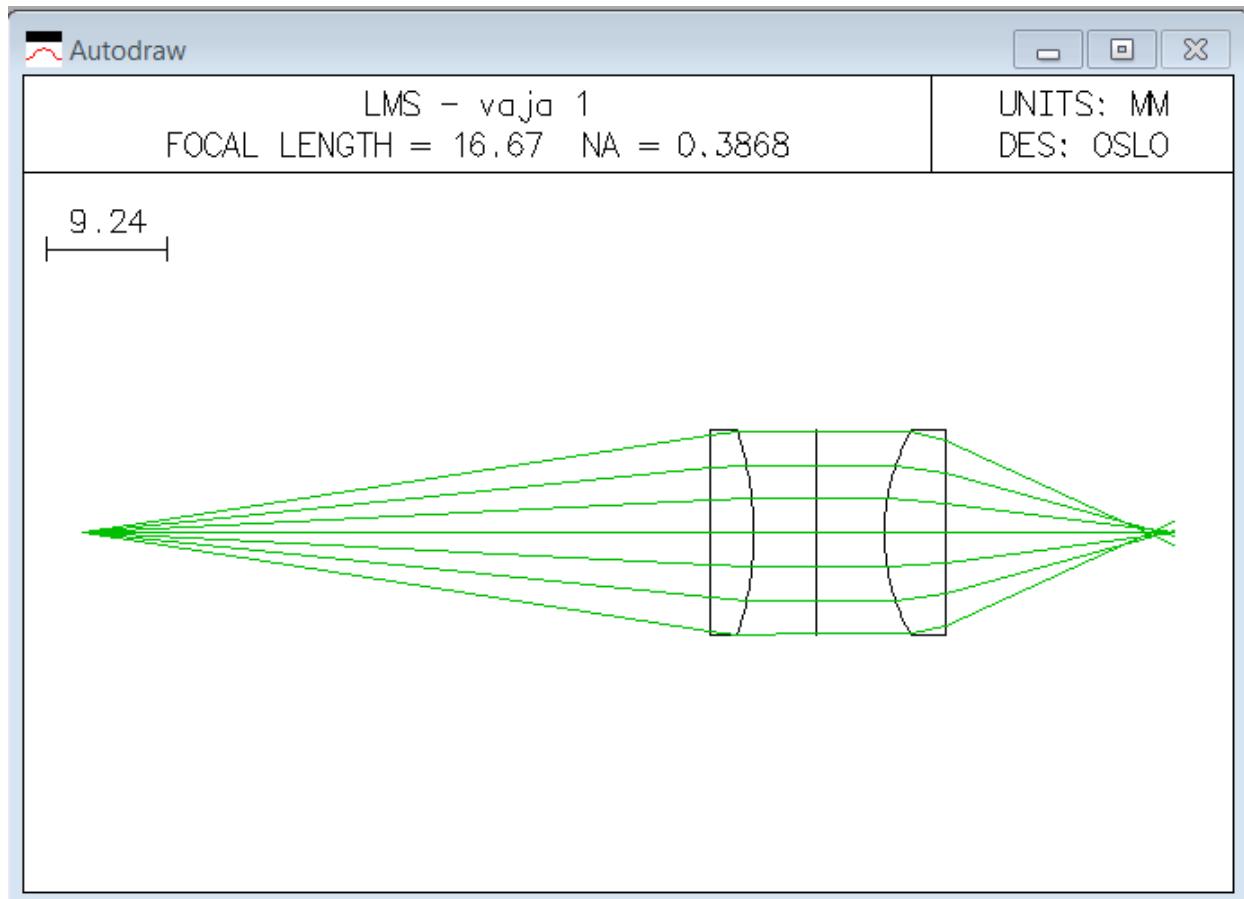
Vidimo, da se žarki, ki so blizu optične osi, dejansko sekajo na slikovni ravnini, tisti bolj oddaljeni pa se zaradi sferične aberacije sekajo bliže sistemu.

## 2.5 Karakterizacija preslikave točkovnega svetlobnega izvora

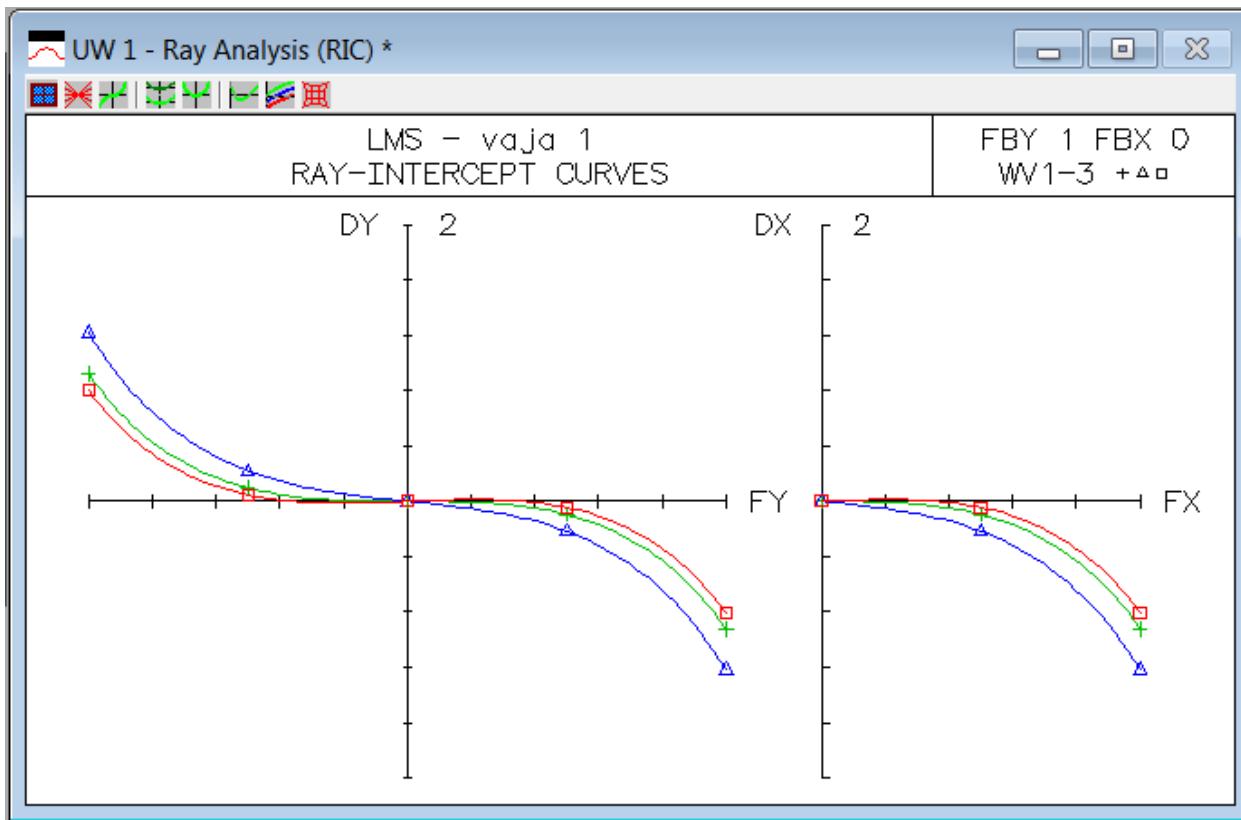
Oddaljenost med sečiščema opazovanega in glavnega žarka s slikovno ravnino se spreminja v odvisnosti od višine opazovanega žarka na vstopni odprtini. To odvisnost lahko prikažemo grafično s tako imenovano žarkovno presečno krivuljo (*angl.: ray intercept curve*), kjer abscisna os predstavlja relativno višino žarka na vstopni odprtini (imenovana *FX* oziroma *FY*), ordinatna os pa oddaljenost (*DX* oziroma *DY*).

Spodnja slika prikazuje primer treh žarkovnih presečnih krivulj, ki pripadajo trem legam slikovne ravnine. Vidimo, da se s spremenjanjem lege slikovne ravnine spreminja naklon presečne krivulje, ne pa tudi njena ukrivljenost. Krivulja b predstavlja potek oddaljenosti, kadar je slikovna ravnina na mestu paraksialnega nastanka slike. V tem primeru sekajo žarki, ki imajo na izhodu iz leče pozitivno višino, pod glavnim žarkom (zato ima krivulja negativno vrednost).

Žarki, ki izhajajo iz spodnje strani leče, pa sekajo slikovno ravnino nad glavnim žarkom.



- V programu *OSLO LT* izrišemo žarkovno presečno krivuljo v oknu *UW 1* (za grafični prikaz) tako, da izberemo **Ray Analysis** v meniju **Setup Window** (skrajno levi gumb, zgoraj). Nato pa izberemo **RIC Plot**. V oknu se nam izriše sledeč diagram:



Dejansko sta prikazana dva diagrama:  $DY(FX)$  in  $DX(FY)$ , ki pa se za primer osno postavljene objektne točke medsebojno ne razlikujeta. Poleg tega so na vsakem diagramu tri krivulje, ki predstavljajo žarkovna presečišča za različne valovne dolžine: modra barva za modro svetlobo, zelena za zeleno in rdeča za rdečo barvo svetlobe. Natančne vrednosti valovnih dolžin posamezne barve lahko spremojte v oknu *Surface Data* s klikom na gumb **Wavelengths**.

Iz diagrama vidimo, da se s povečevanjem vstopnega premera žarkovnega snopa vedno hitreje (po kubični odvisnosti) povečuje tudi razdalja DY oziroma DX. Maksimalna vrednost znaša približno 1 mm.

Takšen način prikazovanja optične napake je zelo učinkovit, saj lahko enostavno vidimo kakšen je še sprejemljiva velikost vstopne odprtine oziroma zaslonke.

## 2.6 Primerjava treh tipov leč pri preslikavi iz neskončnosti v končnost

Primerjajte **bikonveksno**, **plankonveksno** in »**Best-Form**« lečo za primer preslikave iz neskončnosti v končnost. Plankonveksna in »*Best-Form*« leča naj bosta obrnjeni *prav* in *narobe*.

- Podatke o lečah poiščite na spletu. Uporabite leče brez antirefleksnega nanosa, z goriščno razdaljo **50 mm** in premerom **1 inč**. Material leče naj bo **BK7**.
- V leče naj vstopata svetlobna snopa s polmeroma **2 mm** in **10 mm**.
- Primerjajte radije preslikane točke v:
  - gorišči točki, kot jo navaja dokumentacija leče (*BFL*)

- v paraksialnem gorišču
- na mestu najmanjše točke (**autofocus – minimal RMS spot size**).



---

CHAPTER  
THREE

---

## FIZEAUEV INTERFEROMETRER

Navodila za laboratorijsko vajo so zaenkrat dostopna le v *.pdf* verziji na povezavi Fizeaujev interferometer. Pri tej vaji booste uporabljali polariziran He-Ne laser z valovno dolžino  $633\text{ nm}$  in premerom snopa  $0,8\text{ mm}$ .

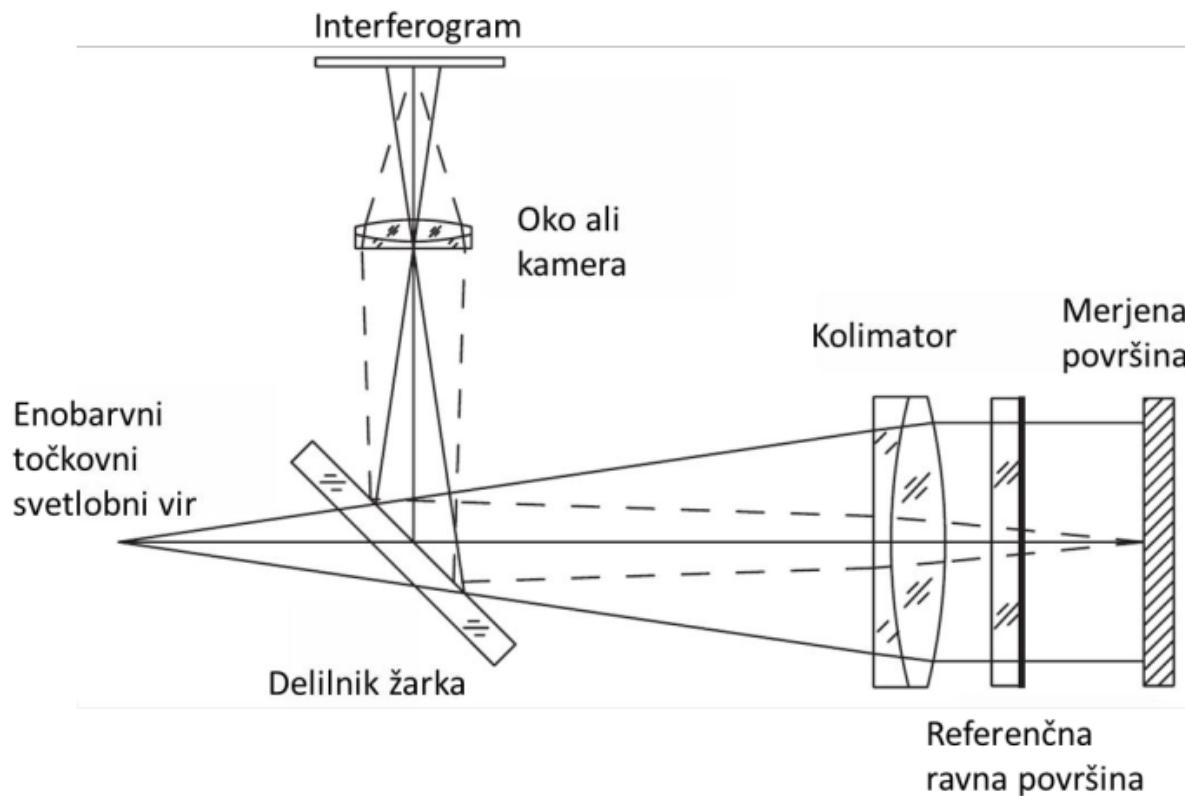


Fig. 1: Slika 1. Shema refraktorskega Fizeau-jevega interferometra, kjer je kolimator leča.

### 3.1 Praktične naloge:

1. Tik za laserski izvor postavite razširjevalnik snopa. Prva leča naj ima goriščno razdaljo  $7,5\text{ mm}$ , druga pa  $150\text{ mm}$ . Na ustrezno mesto vstavite zaslонko (premera  $15\text{ m}$ ) za prostorsko filtriranje žarka. Drugo lečo postavite na takšno razdaljo, da bo žarek na izhodu kolimiran.
2. Med zaslonko in drugo lečo postavite ploščati delilnik žarka tako, da se bo del žarka odbijal pravokotno na optično os laserja. Pazite da nosilec ne bo zastiral žarka.
3. Izmerite vzporednost prednje in zadnje površine planparalelnih stekel. Meritev izvedite na dveh vzorcih:
  - kvadratno steklo dimenzij  $25\times 25\text{ mm}$  in
  - okroglo steklo premera cca  $40\text{ mm}$ .

V ta namen postavite vzorec za kolimator in ga naravnajte pravokotno na optično os žarka. Pogoj za to je, da se odbiti žarki preslikajo nazaj v točkovni izvor - odprtino zaslonke. Interferogram na zaslonu fotografirajte in določite kot nagiba med prednjo in zadnjo površino. Iz fotografije tudi določite morebitna lokalna odstopanja od ravnosti površin! Za izračun boste potrebovali tudi podatek o povečavi interferograma (njegova velikost je namreč odvisna od pozicije zaslona). Zato del kolimiranega žarka tik pred vzorcem zastrite z okroglo palico znanega premera, s čimer boste tudi na interferogramu dobili senco znane širine. Kakšen bo interferogram, če vzorec zavrtite za  $90^\circ$  okrog optične osi žarka?

4. Izmerite ravnost prednje površine steklene plošče. V ta namen postavite za kolimator referenčno ploščo (okrogla klinasta prizma s kotom  $2^\circ$ ) in jo naravnajte tako, da se bodo žarki z zadnje površine odbili nazaj v odprtino zaslonke. Za referenčno površino dodajte še merjeno ploščo (kvadratno steklo dimenzij  $25\times 25\text{ mm}$ ), ki ji predhodno pobarvajte zadnjo površino. Za to uporabite črn flomaster za pisanje po tabli. Orientirajte jo tako, kot pred tem referenčno površino! Ko bosta obe površini dovolj vzporedni, boste na interferogramu opazili interferenčne proge. S finim pozicioniranjem merjene plošče skušajte na interferogramu dobiti osem do deset prog! Interferogram fotografirajte in določite neravnost površine!
5. Merjeno površino skušajte čim bolj poravnati, tako, da bo na interferogramu vidnih čim manj interferenčnih prog. Interferogram fotografirajte, nato pa kolimacijsko lečo izmanknite tako, da bo pas izhodnega žarka oddaljen približno tri metre. Kakšen je interferogram sedaj? Razložite!
6. Kolimacijsko lečo postavite nazaj na pravo pozicijo. Očistite barvo z zadnje površine vzorca in ga ponovno vstavite v interferometer. Kakšen je interferogram sedaj. Fotografirajte in razložite!

## TWYMAN-GRENOV INTERFEROMETER

Navodila za laboratorijsko vajo so zaenkrat dostopna le v *.pdf* verziji na povezavi Twyman-Greenov interferometer s faznim zamikanjem

Pri tej vaji boste uporabljali polariziran *He-Ne* laser z valovno dolžino  $633\text{ nm}$  in premerom žarka  $0.8\text{ mm}$ .

### 4.1 Praktične naloge:

1. Razširjevalnik žarka je sestavljen iz leč  $f_1 = 7,5\text{ mm}$  in  $f_2 = 150\text{ mm}$  ter zaslonke za prostorsko filtriranje žarka. Razdaljo med lečama nastavite z uporabo strižnega interferometra. V ta namen uporabite okroglo plan-paralelno steklo premera  $40\text{ mm}$ , s katero boste razdelili žarek za razširjevalnikom in žarka medsebojno prečno zamknili, kakor je prikazano na sliki 3.6. Fotografirajte in razložite interferograme, ko je kolimacijska leča preblizu, predaleč in v pravilni poziciji.
2. Sestavite *Twyman-Greenov* interferometer. Pri tem uporabite ploščati delilnik žarka dimenzij  $25 \times 35\text{ mm}$ , kvadratno zrcalo  $25 \times 25\text{ mm}$  (kot referenčno zrcalo –  $M1$ ) in zrcalo ( $M2$ ), katerega obliko površine morate izmeriti. Sestavljamte po naslednjem vrstnem redu:
  - Za kolimator postavite delilnik žarka pod kotom  $45^\circ$ . Delilnik vpnite v prijemo tako, da zagotovite nemoteno širjenje prepuščenega in odbitega žarka!
  - Kakšna mora biti smer polarizacije glede na delilnik žarka, da je sekundarni odboj z delilnika žarka minimalen? Skicirajte! Laserja ne obračajte, da se ne spremeni lokacija žarka glede na zaslonko v razširjevalniku žarka!
  - Zrcali postavite tako, da sta pravokotni glede na posamezen žarek. Postopek je enak, kot pri *Fizeaujevem* interferometru.
  - S finim nastavljanjem orientacije enega od zrcal skušajte doseči interferogram, ki bo imel približno deset prog orientiranih enkrat vertikalno in drugič horizontalno. Interferograma fotografirajte in razložite, kakšna je ukrivljenost valovnih front.
3. Pripravite mehanizem za fazno zamikanje zrcala  $M2$ . Na optično letev, za merjeno površino, namestite pozicionirno mizico z mikrometrskim vijakom. Pozicionirno mizico in nosilec merjene površine povežite z natezno vzmetijo. Razmik med obema elementoma nastavite tako, da bo vzmet raztegnjena za približno  $10\%$  glede na prvotno dolžino. Preverite, da so vse mehanske komponente dobro pritrjene! Slabo stisnjeni kontaktni spoji imajo namreč izrazito nelinearno karakteristiko deformacije v odvisnosti od obremenitve. Sprva je krivulja strma, nato pa vedno položnejša. Še večjo nevšečnost predstavlja kontaktno trenje, ki rezultira k veliki histereziji omenjene karakteristike.
4. Opravite umeritev aktuacijskega mehanizma tako, da povečate fazo merilnemu žarku petkrat po  $2\pi$  in pri tem izmerite potreben pomik pozicionirne mizice. Spremembo faze izmerite s štetjem interferenčnih maksimumov. Rezultat umeritve prikažite na diagramu  $\Delta\phi$  (sprememba faze) v odvisnosti od  $\Delta x_{poz.miz}$  (premik pozicionirne

mizice). V kolikor je nelinearnost večja od 20 %, preverite pritrditev vseh komponent. Poizkusite tudi s povečanjem prednapetja natezne vzmeti.

5. Fotoaparat brez objektiva namestite na mesto kjer sicer uporabljate zaslon za vizualno opazovanje interferograma. Preverite, da je na fotoaparatu izbran ročni način osvetlitve (program *M*), ter da je občutljivost *ISO* enaka 100 ter izbran najkrajši čas osvetlitve ( $1/4000\text{ s}$ ). Interferogram centrirajte na sredino senzorskega elementa s premikanjem nosilne plošče fotoaparata. Pri tem imejte na fotoaparatu vključen predogled slike (angl.: *Liveview*). Izvedite poskusno fotografijo in preverite osvetljenost slike. V ta namen uporabite histogram intenzitete, ki ga fotoaparat prikaže ob posneti sliki. Rdeča barva se nikakor ne sme prelivati v rumeno ozziroma belo! Če se to vseeno zgodi, dodajte pred fotoaparat atenuator – zatemnjeno steklo.
6. Zrcalo *M2* naravnajte tako, da bo približno 8 interferenčnih prog v vertikalni smeri in posnemite serijo 10-ih fotografij, med katerimi povečujete fazo po  $\pi/2!$  Med meritvijo skušajte zagotoviti čim boljše pogoje v smislu zmanjšanja vibracij. Na koncu serije fotografij vizualno preverite v smislu kontrole enakomernega premikanja interferenčnih prog v vertikalni smeri.

## UVOD V RAČUNALNIŠKO OBDELAVO SLIK

---

**Note:** Če še nimate nameščenega *Python*-a in *OpenCV*-ja sledite navodilom na povezavi: navodila\_python.

---

Na tej vaji se boste spoznali z osnovnimi koncepti računalniške obdelave slik, programskim jezikom *Python* (predpostavljamo, da ste osnovno znanje že osvojili pri predmetu *Programiranje in numerične metode v ekosistemu Pythona*) in predvsem s programsko knjižnico *OpenCV*.

### 5.1 Uvod

**OpenCV** (*Open Source Computer Vision Library*) je knjižnica programskih funkcij, ki je v osnovi namenjena procesiraju slik v realnem času. Gre za cross-platform knjižnjico in je [prosta za uporabo](#). Prav v tem leži ena njenih največjih prednosti, saj jo lahko uporabljamo v različnih programskih jezikih ([C++](#), [Python](#), [Java](#), [MATLAB](#)) in v različnih operacijskih sistemih ([Windows](#), [Linux](#), [macOS](#), [Android](#), [iOS](#)).

Na današnji vaji bomo spoznali osnovno fukcionalnost, trike in pasti.

---

**Note:** Kot velikokrat v življenju, tudi v programiraju velja, da do končnega cilja ne vodi le ena (pravilna) pot, ampak je le-teh (skorajda *neskončno*) mnogo. Različne poti se lahko bolj ali manj primerne, boli ali manj upoštevajo različne konvencije ipd... Pri predmetu LMS se bomo poskušali čim bolj držati pravil, ki ste jih spoznali pri predmetu *Programiranje in numerične metode v ekosistemu Pythona*); včasih najbrž neuspešno. Koda, ki jo bomo pisali bo daleč od optimalne Python kode, ampak bo zelo *skriptna* in čim bolj podobna *pseudo kodi*.

---

#### 5.1.1 Vaja 1: branje in pisanje slik z diska

Napišimo za začetek enostaven *Python* skripto, ki bo z diska prebrala sliko in jo prikazala.

```
1 import cv2
2
3 image = cv2.imread('/home/iovius/Downloads/Lenna_(test_image).png')
4 cv2.imshow('my image', image)
5 cv2.waitKey()
6
7 cv2.destroyAllWindows()
```

Če pogedamo sedaj vrstico po vrstico.

```
import cv2
```

V prvi vrstici vključimo *OpenCV*.

```
image = cv2.imread('/home/iovius/Downloads/Lenna_(test_image).png')
```

V vrstici 3 preberemo sliko z diska s klicom `cv2.imread()`. `cv.` pomeni, da bomo sedaj klicali funkcijo iz knjižnice, `imread` pa je ime funkcije. Vidite lahko, da funkcija sprejme samo en argument. Ta je tipa string in opisuje pot na disku do slike.

---

**Note:** Ta navodila so pisana v *OS Ubuntu/Linux*. Zato so vse poti napisane v Linux "stilu". Če uporabljate *OS Windows*, je zadeva nekoliko bolj zapletena. Za opisovanje poti imate nekako 3 možnosti. Recimo, da je vaša slika na lokaciji "`C:\SomeFolder\SomeOtherFolder\my_image.jpg`".

1. pot podate z dvojno levo poševnico (*backslash*-om), torej: "`C:\\SomeFolder\\\\SomeOtherFolder\\\\my_image.jpg`"
2. pot podate s poševnico (*slash*-om), torej: "`C:/SomeFolder/SomeOtherFolder/my_image.jpg`"
3. pot podate kot *dobeseden string (literal string)*, torej: `r"C:\\SomeFolder\\SomeOtherFolder\\my_image.jpg"`

Težava je v tem, da če pot kopirate, bodo v imenu leve poševnice, ti pa so rezervirani za *posebne znake (special characters)*, npr. "`\n`" pomeni novo vrstico, "`\t`" pomeni tab, "`\s`" presledek itd...). Python ne loči med ", torej lahko uporabljete kateregakoli, morata pa biti **v paru enaka znaka**.

---

**Warning:** Ne glede na operacijski sistem pa **morate podati celotno pot; torej od diska do končnice!** Izjema so t.i. *relativne poti*.

```
cv2.imshow('my image', image)
```

V četrti vrstici sliko prikažemo s klicom funkcije `cv2.imshow()`. Funkcija ima 2 parametra: prvi je ime okna (glej *sliko I*), z drugim pa podamo sliko/matriko, ki jo želimo izrisati.

---

**Note:** Slike so v računalništvu matrike. Če ima slika *640x480* (širina x višina) piksov, imamo torej matriko *480x640* (pazite, pri matrikah **najprej podamo število vrstic, potem število stolpcev!**) kjer vrednost vsakega elementa popošuje intenziteto posameznega piksla (v primeru sivinske - *grayscale* slike). Če imamo opravka z barvno sliko (npr. *RGB* barvni model) imamo 3 matrike, po eno za posamezen kanal (barvo).

---

```
cv2.waitKey()
```

V peti vrstici s funkcijo `cv2.waitKey()` čakamo na pritisk katerikoli tipke. Če pogledamo v [dokumentacijo](#) funkcije, je tam zapisano `retval = cv.waitKey([, delay])`. Branje in razumevanje dokumentacije je ena najpomembnejših vrlin, ki jo kot programerji moramo osvojiti. Poglejmo, kaj lahko iz tega razberemo. Znotaj () vidimo `[, delay]`. To pomeni, da funkcija sprejme 1 argument, ki pa je opcijski (to povesta []): to pomeni, da ga lahko podamo, ni pa nujno. Če nadaljnje preberemo dokumentacijo piše: "*The function waitKey waits for a key event infinitely (when delay0) or for delay milliseconds, when it is positive.*". Torej, če argumenta `delay` ne podamo, bo funkcija čakala na pritist tipke "v neskončnost", če pa jo, bo počakala samo število milisekund, kot smo ga podali s parametrom `delay`. Vidimo lahko tudi, da funkcija vrne `retval`. V dokumentacij piše: "*It returns the code of the pressed key or -1 if no key was pressed before the specified time had elapsed.*". Ugotovimo lahko, katero tipko smo pritisni (vrne **ASCII** kodo, najdete jo lahko v stolpcu **Dec** v [tabeli](#), ali s *Python* klicom `ord('<vaša črka>')`).

Po 5. vrstici bi se torej izvajanje programa morallo ustaviti in prikazati sliko. Ko pritisnemo tipko ze izvajanje skripte nadaljuje z zadnjo vrstico, kjer vsa odprta okna zapremo.

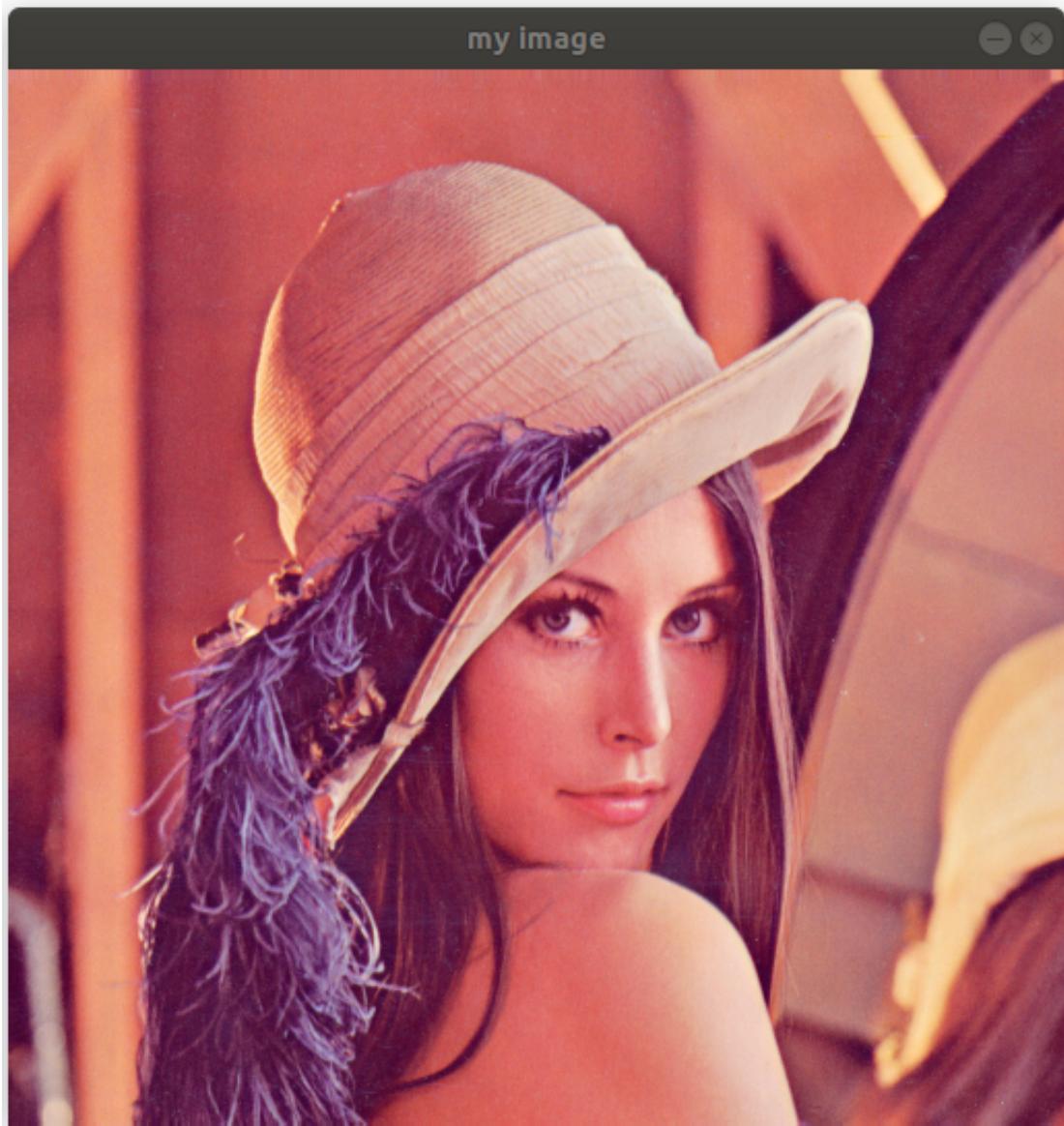


Fig. 1: Slika 1. Prikazana slika.

---

**Note:** Če ne kličemo funkcije cv2.destroyAllWindows() običajno okno ostane odprto in neodzivno (ne moremo ga zapreti), kar je lahko moteče.

---

Kot lahko vidimo, je prikazana slika barvna, torej jo sestavljajo 3 kanali, rdeč, zelen in modri. Dopolnimo skripto tako, da sliko razdelimo na posamezne kanale, jih prikažemo, premešamo in sliko spet shranimo.

```
1 import cv2
2
3 image = cv2.imread('/home/iovius/Downloads/Lenna_(test_image).png')
4
5 [blue, green, red] = cv2.split(image)
6 image_merged = cv2.merge([red, green, blue])
7
8 cv2.imshow('red', red)
9 cv2.imshow('green', green)
10 cv2.imshow('blue', blue)
11 cv2.imshow('my image', image)
12 cv2.imshow('image merged', image_merged)
13 cv2.waitKey()
14
15 cv2.imwrite('/home/iovius/Downloads/Lenna_(test_image).png', image_merged)
16
17 cv2.destroyAllWindows()
```

Dodali smo vrstice 5-10, 12 in 15.

---

**Note:** Čeprav običajo slišite "RGB", OpenCV uporablja zaporedje **blue, green, red**.

---

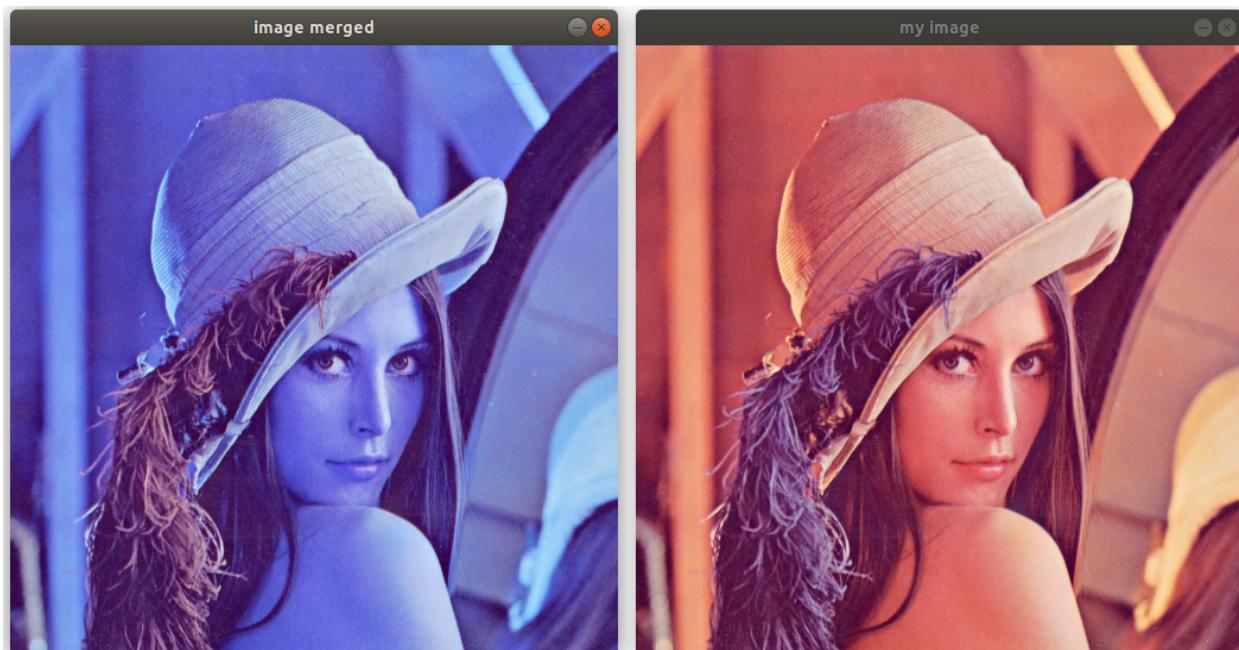


Fig. 2: Slika 2. Originalna Lenna in Lenna s premešanimi kanali.



Fig. 3: Slika 3. Posamezni kanali.

### 5.1.2 Vaja 2: zajemanje in prikazovanje slike s kamere

V tem delu vaje bomo pogledali, kako s pomočjo *OpenCV*-ja zajamemo sliko s kamere, ki je priklopljena na računalnik.

```

1 import cv2
2
3 cam = cv2.VideoCapture(0)
4 flip = False
5
6 while True:
7     retval, frame = cam.read()
8     if not retval:
9         break
10    if flip:
11        frame = cv2.flip(frame, 1)
12    cv2.imshow("frame", frame)
13    key = cv2.waitKey(10)
14    if key == 27:
15        break
16    elif key == ord('f'):
17        flip = not flip
18
19
20 cam.release()
21 cv2.destroyAllWindows()
22

```

Skripta je sedaj nekoliko daljša.

```
cam = cv2.VideoCapture(0)
```

Najpomembnejša razlika v primerjavi s prejšnjo skripto je vrstica 3. `VideoCapture` je razred za zajem videa iz različnih video datotek, sekvenc slik in *hardware*-a - kamer. Če podrobneje pogledamo dokumentacij omenjenega razreda lahko vidimo, da imamo na voljo 5 različnih konstruktorjev :

1. <VideoCapture object> = cv.VideoCapture( ) ... default konstruktor, za enkrat za nas ni zanimiv
2. <VideoCapture object> = cv.VideoCapture( filename ) ... “odpremo” video datoteko

3. <VideoCapture object> = cv.VideoCapture( filename, apiPreference ) ...  
“odpremo” video datoteko in določimo s katero *video* knjižnico
4. <VideoCapture object> = cv.VideoCapture( index ) ... “odpremo” kamero
5. <VideoCapture object> = cv.VideoCapture( index, apiPreference ) ... “odpremo” kamero in določimo s katero *video* knjižnico

**Warning:** Kadarkoli “odpremo” nek kos *hardware*-a, ga moramo potem tudi zapreti (glejte vrstico 20). Če tega ne storimo, v ob naslednjem poskusu odpiranja to ne bo mogoče, saj je npr. kamera že odprta. Tudi v dokumentaciji piše: *In C API, when you finished working with video, release CvCapture structure with cvReleaseCapture()...*

---

**Note:** Dokumentacija *OpenCV*-ja je primarno napisana za *C++*; za *Python* je precej skopa in moramo velikokrat prebrati kaj piše za *C++* in to “preversti” v *Python*.

---

V vrstici 3 lahko vidite, da smo uporabiti opcijo 4. `index` je tipa `int` in določa, katero kamero želimo odpreti.

---

**Note:** *Python* tako kot vsak resen programski jezik začenja šteti z **0** in ne z **1** (*I'm looking at you, MATLAB!*); torej, *index prvega elementa* je **0**!

---

Klic nam vrne **objekt** tipa *VideoCapture*, ki ga “ujamemo” v spremenljivo `cam`. V vrstivi 6 začnemo neskončno **while** zanko.

```
retval, frame = cam.read()
```

V vrstici 7 kličemo metodo razreda *VideoCapture* `read()`. V dokumentaciji lahko preberemo, da metoda *Grabs, decodes and returns the next video frame..* Sintaksa je zapisana kot `retval, image = cv.VideoCapture.read( [ , image] )`. Spet lahko vidimo, da sprejme en opcjski parameter `image`, ki pa ga lahko “ujamemo” tudi na izhodnji strani. Glede `retval` pa piše *false if no frames has been grabbed*. Tako v vrsticah 8-9 preverim, ali je branje uspelo in če ni *neskončno* zanko zaključimo.

V vrstici 10 preverimo, kakšno je trenutno stanje spremenljivke `flip` in eventuelno izvedemo vrstico 11. **Vaša naloga je**, da na spletu poiščete dokumentacijo *OpenCV* funkcije `cv2.flip()` in **ugotovite**, kaj funkcija naredi in kaj sta oba parametra.

```
if key == 27:  
    break  
elif key == ord('f'):  
    flip = not flip
```

V vrsticah 14-17 imamo 2 klica povezana s tipko, ki smo jo (eventuelno) pritisnili v vrstici 13. 27 je koda tipke *ESC* in v tem primeru zanko zaključimo, če pa pritisnemo tipko *f* sprememimo *True/False* stanje spremenljivke `flip`.

### 5.1.3 Vaja 3: globina slike (*image depth*) in prikazovanje slike

Če uporabljate *Jupyter Notebook*, spremenljivke ostanejo v spominu tudi, ko se izvajanje skripte konča. V naslednjo celico napišite klic:

```
print(type(frame[0, 0, 0]))
output: <class 'numpy.uint8'>
```

Slika v spremenljivki `frame` je v resnici trodimenzionalna matrika, vrednosti znotraj oglatih oklepajev pa koordinata piksla, ki ga želimo izpisati: *prva* vrednost določa vrstico, *druga* stolpec in *tretja* kanal, če imamo RGB sliko. Če je slika sivinska moramo podati samo 2 vrednosti. Vidimo lahko, da je element tipa `uint8`. To pomeni, da je *unsigned integer* (celo število brez predznaka) popisan z osmimi biti. Najmanjša vrednost, ki jo torej lahko opišemo je torej '0' (binarno 0000 0000, hex 00), največja pa 255 (binarno 1111 1111, hex ff). **Vprašanje:** kaj se zgodi, če `uint8` spemelnjivki z vrednostjo 255 prištejemo 1?

Napišimo kratko skripto, ki bo izrisala 3 slike: črno, sivo in belo.

```
1 import numpy as np
2 import cv2
3
4 dimensions = [500, 500]
5
6 black = np.ones(dimensions, np.uint8)*0
7 gray = np.ones(dimensions, np.uint8)*126
8 white = np.ones(dimensions, np.uint8)*255
9
10 cv2.imshow('white', white)
11 cv2.imshow('gray', gray)
12 cv2.imshow('black', black)
13 cv2.waitKey()
14
15 cv2.destroyAllWindows()
```

Kot lahko vidite, smo sedaj uvozili še eno knjižnico, `numpy` (vrstica 1). Gre za “*The fundamental package for scientific computing with Python*”. Ker je neno ime dolgo, ga okrajšamo, tako da jo lahko kličemo kot `np..`. V vrstici 4 določimo dimenzijs slik. V vrsticah 6-8, naredimo najprej 3 matrike enic, ki jih potem pomnožimo z 0 za črno sliko, 126 za sivo in 255 za belo. Ko slike izrišemo, vidimo nekaj takega:



Fig. 4: Slika 4. Bela, siva in črna generirana slika.

V vrsticah 6-8 smo nastavili, da je posamezen element matrik tipa `uint8`. To imenujemo **image depth** (v prostem

prevodu ‐globina‐ slike). Spremenimo sedaj globino naših slik v float32, torej število s plavajočo vejico in 32 biti, in skripto ponovno poženimo.

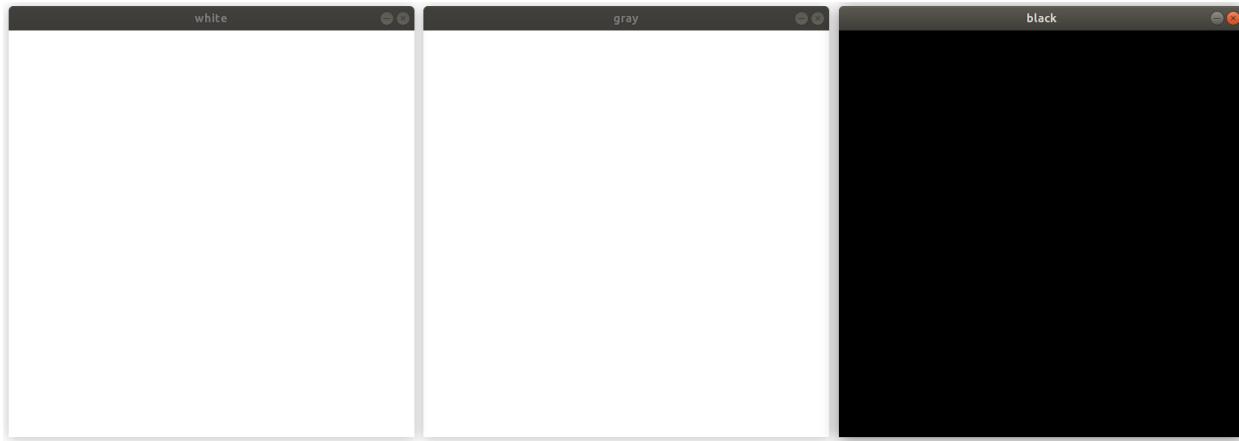


Fig. 5: Slika 5. Enake 3 slike kot na sliki 4, le da so sedaj v formatu float32.

Opazimo lahko, da sta sedaj tako slika gray kot slika white beli. Zakaj?

---

**Note:** **Zapomnite si:** če z OpenCV funkcijo imshow() izrisujemo slike globine integer, bo 0 ... črna, 255 ... bela, če pa izrisujemo slike globine float, bo 0.0 ... črna, 1.0 ... bela!

---

#### 5.1.4 Vaja 4: filtriranje slik

Pri procesiraju slik se pogosto poslužujemo različnih filtriranj. Filtriranje običajno temelji na principu konvolucije. V OpenCV-ju imamo na voljo različne predefinirane filtre, lahko pa ga ustvarimo sami. Karakteristike filtra določa njegovo jedro (angl. kernel). Poglejmo si primer enega najbolj uporablajnih filtrov - *Gaussov filter*.

```

1 import cv2
2 import numpy as np
3
4 image = cv2.imread('/home/iovius/Documents/docs/source/uvod_v_py/images/Lenna_(test_
5 ↵image).png')
6 image = image.astype(np.float32)
7
8 image_filter_1 = cv2.GaussianBlur(image, (5, 5), 3)
9
10 kernel = cv2.getGaussianKernel(5, 3, cv2.CV_32F)
11 image_filter_2 = cv2.sepFilter2D(image, cv2.CV_32F, kernel, kernel)
12
13 difference = image_filter_1 - image_filter_2
14
15 print("kernel:\n", kernel)
16 print("\ndifference:\n", (np.sum(difference)))

```

```

kernel:
[[0.17820325]
 [0.21052228]
 [0.22254895]
 [0.21052228]
 [0.17820325]]

```

(continues on next page)

(continued from previous page)

```
[0.17820325]]
```

```
difference:  
0.0
```

V vrstici 7 uporabimo predefiniran *Gaussov* filter v funkciji *GaussianBlur*. Medtem v vrstici 9 generiramo eno-dimenzionalen Gaussov filter z enakimi parametri, potem pa ga uporabimo v *sepFilter2D*. Tej funkciji podamo posebej filter v X in Y smeri. V *output* oknu lahko vidimo izpisa; vrednosti *kernela* in pa razliko med obema slikama, ki nam potrdi, da smo v obeh primerih sliko enako sfiltrirali.

Naredimo sedaj lasten *Moving Average* filter.

```
1 import cv2  
2 import numpy as np  
3  
4 image = np.zeros((500,500),np.float32)  
5 image[200:300, 200:300] = 1.0  
6  
7 kernel = np.ones((11,11),np.float32)  
8 kernel /= kernel.sum()  
9 print(kernel)  
10  
11 image_filt = cv2.filter2D(image,-1,kernel)  
12  
13 cv2.imshow("image", image)  
14 cv2.imshow("filtered", image_filt)  
15 cv2.waitKey()  
16  
17 cv2.destroyAllWindows()
```

```
[[0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.  
 ↪00826446 0.00826446 0.00826446 0.00826446]  
 [0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.  
 ↪00826446 0.00826446 0.00826446 0.00826446]  
 [0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.  
 ↪00826446 0.00826446 0.00826446 0.00826446]  
 [0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.  
 ↪00826446 0.00826446 0.00826446 0.00826446]  
 [0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.  
 ↪00826446 0.00826446 0.00826446 0.00826446]  
 [0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.  
 ↪00826446 0.00826446 0.00826446 0.00826446]  
 [0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.  
 ↪00826446 0.00826446 0.00826446 0.00826446]  
 [0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.  
 ↪00826446 0.00826446 0.00826446 0.00826446]  
 [0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.  
 ↪00826446 0.00826446 0.00826446 0.00826446]
```

V vrsticah 7 in 8 generiamo  $11 \times 11$  matriko enic in jo v normaliziramo (vsota vseh uteži mora biti 1.0). Na sliki 6 sta prikazani izvorna in filtrirana slika.

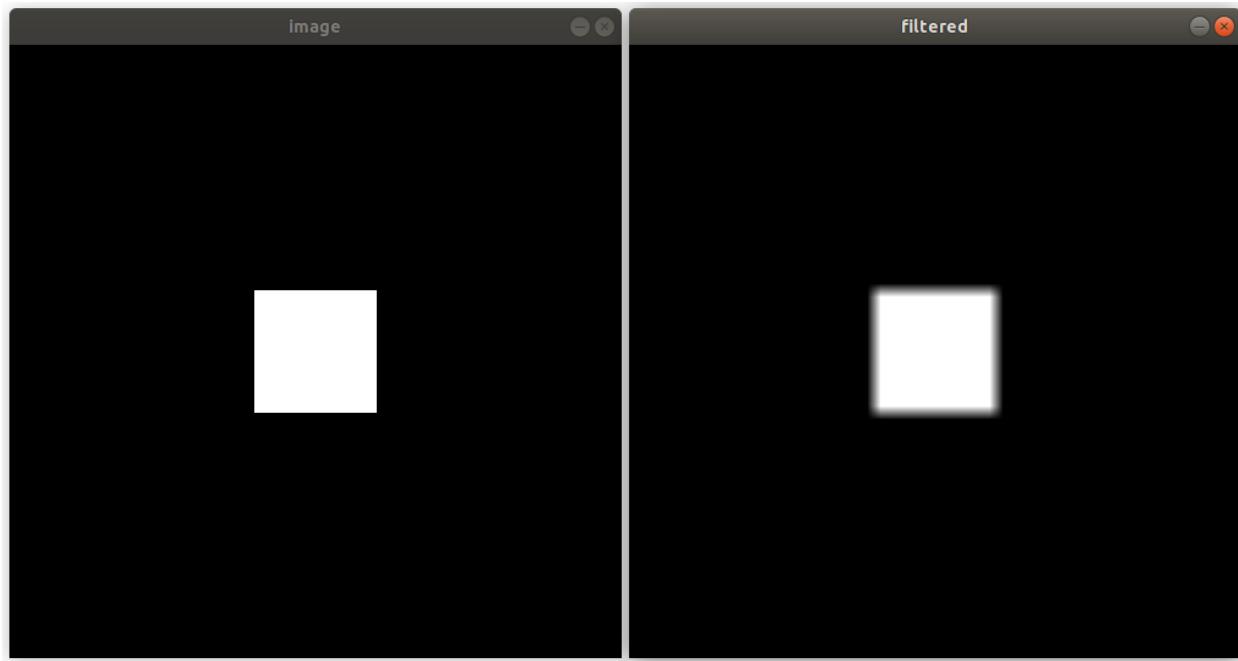


Fig. 6: Slika 6. Izvorna in filtrirana slika.

## 5.2 Median filter

*Median* filter je filter, kjer aktivnemu pikslu pripisemo vrednost mediane znotraj konvolucijskega okna. Gre za filter, s katerim lahko odstranimo npr. termičen šum.

```

1 import cv2
2 import numpy as np
3
4 dimensions = [500, 500]
5 N = 100
6
7 image = np.ones(dimensions,np.uint8)*126
8
9 ind_v = np.random.randint(0, dimensions[0], N)
10 ind_u = np.random.randint(0, dimensions[0], N)
11
12 for i in range(N):
13     if 0 == np.random.randint(0,2,1):
14         image[ind_v[i],ind_u[i]] = 0
15     else:
16         image[ind_v[i],ind_u[i]] = 255
17
18 image_med = cv2.medianBlur(image,7)
19
20 cv2.imshow("image", image)
21 cv2.imshow("image_med", image_med)
22 cv2.waitKey()
23
24 cv2.destroyAllWindows()
```

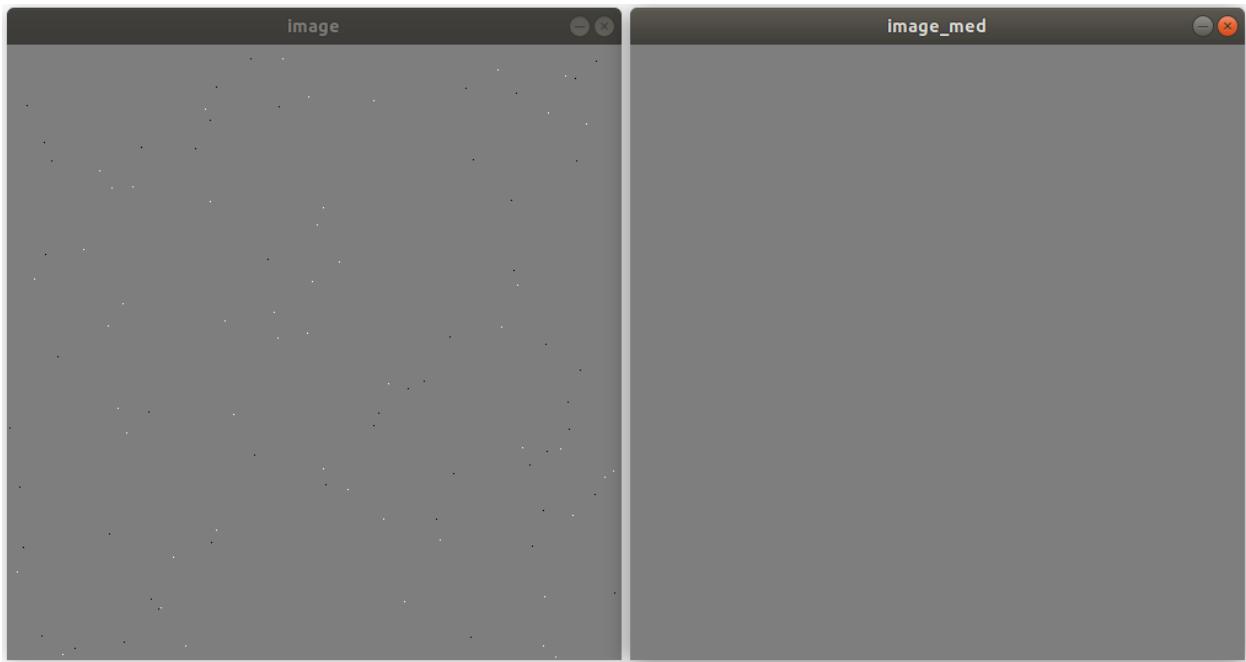


Fig. 7: Slika 7. Izvorna in filtrirana slika z *median* filtrom.

### 5.2.1 Vaja 5: *kopiranje slik*



## RAČUNALNIŠKA ANALIZA INTERFEROGRAMOV

---

**Note:** Če še nimate nameščenega *Python*-a in *OpenCV*-ja sledite navodilom na povezavi: navodila\_python.

---

Na prejšnji vaji smo zajeli slike interferogramov, ki jih želimo sedaj računalniško obdelati. Ker bo skripta razmeroma dolga, jo bomo razdelili na več celic. Sledite navodilom, ki so zapisana v komentarjih.

---

**Note:** Če ne uporabljate *Jupyter Notebooka*, boste morali vse celice zapisati v eno skripto.

---

---

**Note:** Besedilo se vedno nanaša na **sledečo celico**.

---

V prvi celici bomo uvozili knjižnice, ki jih potrebujemo (*cv2* in *numpy*). Prav tako lahko definiramo spremenljivki in 2 (ker se ime spremenljivke v Pythonu ne sme začeti s števil, jo bomo poimenovali 2).

```
1 import cv2
2 import numpy as np
3
4 #definiramo spremenjivko
5 =
6 #definiramo spremenjivko 2*
7 2 =
```

V naslednji celici bomo napisali funkcijo, ki bo prebrala in sprocesirala posamezno sliko. Definicijo funkcije moramo začeti z ukazom `def`. `read_and_process_image(filename, images, resize_factor)` pomeni, da je `read_and_process_image` ime funkcije, `filename`, `images` in `resize_factor` pa so argumenti, ki jih bomo morali podati.

```
1 #funkcija za branje in procesiranje slik
2 #filename ... lokacija slike na disku
3 #images ... seznam, kamor shranimo slike
4 #resize_factor ... faktor, za keterega sprememimo velikost slik
5 def read_and_process_image(filename, images, resize_factor):
6     #preberite sliko v barvah s funkcijo cv2.imread()
7     im =
8     #preberite dimenzije slike z metodo .shape
9     h, w =
10    #spremenite dimenzije slike s funkcijo cv2.resize()
11    im = cv2.resize
12    #"razbite" sliko po posameznih kanalih; cv2.split()
13
14    #filtrirajte sliko - razmislite, kateri kanal; cv2.GaussianBlur()
```

(continues on next page)

(continued from previous page)

```
15 # spremenite podatkovni tip vrednosti v float32 in ustrezno skalirajte
16
17 # v seznam images dodajte končno sliko
18 images.append
```

V tej celici podamo imena posameznih slik in jih zapišemo v *list*.

**Warning:** Spomnite se, kako navajamo poti na disku!

```
1 filenames = []
2 # dodajte ime prve slike
3
4 # dodajte ime druge slike
5
6 # dodajte ime tretje slike
7
8 # dodajte ime četrte slike
9
```

S sledčo celico bomo prebrali slike, jih dodali v *list* *images* in prikazali.

```
1 images = []
2 # za vsako ime v seznamu filenames
3
4     # uporabite funkcijo read_and_process_image
5
6
7 # preberite dimenzije slike; h ... višina, w ... širina
8 h, w =
9
10 # pojrite čez slike in vsako prikažite
11 for i, in enumerate( ):
12
13
14 # spomnite se, kaj morate storiti po tem, ko slike prikažete
15
16 # zaprite vsa okna
17
```

Sedaj bomo začeli z izračunom. V pomoč naj vam bodo *slajdi* s predavanj (carre). V tej celici želimo izračunati vrednost  $\alpha(x, y)$ . Če pogledate, kaj bomo morali storiti v naslednjem koraku, vidite, da je izračun *arctan* nekoliko nesmiselen, saj bomo morali potem v naslednjem koraku izračunati *tan*, tako da namesto  $\alpha(x, y)$  izračunajmo kar  $\tan(\alpha(x, y))$ .

```
1 # odštejte vrednosti slike 2 od vrednosti slike 1
2 I1m2 =
3 # odštejte vrednosti slike 3 od vrednosti slike 0
4 I0m3 =
5 # izračunajte števec
6 dividend =
7 # izračunajte imenovalec
8 divisor =
```

(continues on next page)

(continued from previous page)

```

10 #preverite, da imenovalec ni enak 0
11 divisor[np.abs(divisor) < 0.0000001] = 0.0000001
12
13 #izračunajte tan_alpha
14 tan_alpha =

```

Ko smo izračunali  $\tan(\alpha(x, y))$  nadaljujemo z izračunom faze ( $\Delta\Phi(x, y)$ ).

---

**Note:** Izračun arkustangensa kvocienta dveh števil je precej “zahtevna” operacija, saj imamo 6 možnih primerov. Zato ima praktično vsaka matematična knjižnica (ki da kaj nase) funkcijo `atan2`. V numpy jo najdemo kot `arctan2`.

---

```

1 # izračunajte sinusni del
2 sine =
3 # pomnožite sinusni del z tan_alpha
4
5 # izračunajte cosinusni del
6 cosine =
7
8 # izračunajte fazo; namig (atan2)
9 phase =
10
11 # prikažite sliko
12
13
14

```

Ker smo za izračun faze uporabili funkcijo `arctan` (v našem primeru v resnici `arctan2`) je faza omejena na interval  $[-\frac{\pi}{2}, \frac{\pi}{2}]$  (ker smo uporabili `arctan2` smo v resnici omejeni na interval  $[-\pi, \pi]$ ). Kakorkoli, ker želimo dobiti zvezen potek faze moramo izvesti t.i. *razvijanje faze* (angl.: `phase unwrapping`). Osnovna ideja (enostavnega algoritma za razvijanje faze) je, da se *pomikamo po fazi* in spremljamo, kje se pojavijo nezveznosti. Od tega mesta naprej prištejemo ali odštejemo  $2\pi$ , odvisno od tega, v katero smer se pojavi nezveznost.

```

1 #skopirajte vrednosti iz spremenljivke phase v spremenljivko phase_unwrapped
2 phase_unwrapped =
3
4 # "za vsako vrstico"
5
6     #spremenljivko phase_offset nastavite na vrednosti 0
7
8     # "za vsak element v vrstici"
9
10    #preberite prejšnjo vrednost v vrstici
11    p_previous =
12    #preberite trenutno vrednost v vrstici
13    p_current =
14
15    #če je "stopnica" od trenute do prejšnje vrednosti večja od
16
17        #vrednosti phase_offset odštejte 2
18
19    #če je "stopnica" od trenutne do prejšnje manjša večja od -
20
21        #vrednosti phase_offset prištejte 2
22

```

(continues on next page)

(continued from previous page)

```

23      # vrednosti phase prištejete phase_offset in jo zapišite v phase_unwrapped
24
25

```

Vizualizirajte rezultat.

```

1  # prikažite sliko razvite faze
2
3
4  # zaprite vsa okna
5

```

Precej verjetno vidite naj podobnega srednjemu stolpcu na spodnji sliki. Želimo pa rezultat, kot je v zadnjem stolpcu.

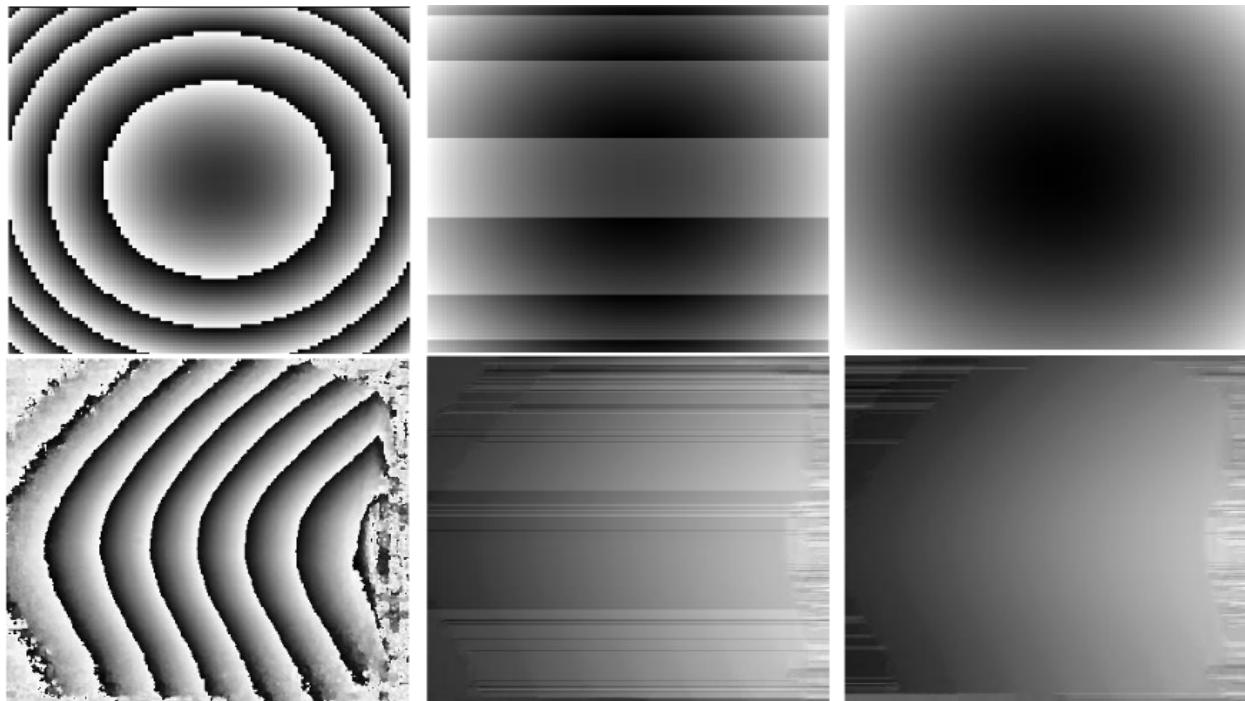


Fig. 1: Slika 1. Izvorne *faze* (prvi stolpec), *faza* kot bo razvita po našem algoritmu (sredinski stolpec) in rezultat, kot ga želimo (zadnji stolpec).

Težava je v tem, da našo *sliko faze* razvijamo samo z leve proti desni, v primeru kot je zgoraj levo na *sliki 1* pa se do sredinskega polja pa nabere že kar nekaj *napake*.

Postopek razvijanja moramo zato ponoviti še enkrat, tokrat v navpični smeri.

```

1  # vrednost phase_unwrapped skopirajte v phase_unwrapped2
2  phase_unwrapped2 =
3  # izračunajte indeks sredinskega stolpca; pazite na tip!!
4  u_scan =
5  # postavite vrednost phase_offset na 0
6  phase_offset = 0
7
8  # "za vsako vrstico"
9

```

(continues on next page)

(continued from previous page)

```

10    # preberite vrednost prejšnjega elementa v sredinski vrstici
11    p_previous =
12    # preberite vrednost trenutnega elementa v sredinski vrstici
13    p_current =
14
15    # izračunajte razliko med trenutno in prejšnjo
16    diff_c_p =
17
18    #deljite z 2
19    diff_c_p
20    #zaikrožite
21    diff_c_p =
22    #pomnožite z 2
23    diff_c_p
24    #odštejte vrednost od phase_offset
25    phase_offset
26
27    #celotni treutni vrstici prištejte vrednost phase_offset
28    phase_unwrapped2[v,:]
```

Vizualizirajte rezultat *faze*.

```

1 # prikažite končni rezultat
2
3
4 # zaprite vsa okna
5
```

Če vas moti šum okoli faze (ljubkovalno imenovan tudi “šavje”), lahko preverite vidljivost signala na posameznem mestu in izrišete le točke, z visoko vidljivostjo.

```

1 # prikažite končni rezultat z upoštevanjem ROI
2
3 # izračunajte standardno deviacijo med slikami
4 STD =
5 # preverite, kje je STD večji od 0.01
6 ROI =
7 # skopirajte razvito sliko v phsUW
8
9 # v phsUW nastavite vrednosti, kjer je ROI enak Fasle na NaN
10
11
12 # izrišite sliko
13 imshow('STD', phsUW)
14 cv2.waitKey()
15 # zaprite vsa okna
16 cv2.destroyAllWindows()
```



## RAZVOJ PROGRAMSKE OPREME LASERSKEGA 3D MERILNIKA

---

**Note:** Če še nimate nameščenega Pythona in OpenCVja sledite navodilom na povezavi: navodila\_python.

---

### 7.1 Uvod

Cilj vaje je praktično spoznati osnove laserske triangulacije in izdelati programsko opremo za enostaven linearni laserski profilomer. Sistem in programska oprema za zajem sekvence merilnih slik sta predhodno pripravljena, vaša naloga pa je, da napišete program, ki bo iz te sekvence slik rekonstruiral obliko merjene površine.

### 7.2 Oprema

- Linearna koračna miza
- Krmilnik koračne mize
- Linijski laserski projektor
- Kamera

### 7.3 Potek vaje

- Ogled sistema (na koračno mizo sta pod kotom pritrjena kamera in laserski projektor, ogledamo si sliko na kamери pri različnih nastavivah kamere in različnih odbojnostih).
- Zajem sekvence merilnih slik.
- Izdelava programa za transformacijo ([Python & OpenCV](#) ).
- Prikaz izmerkov; preverili bomo ujemanje rekonstruirane površine z referenčno.

## 7.4 Osnovni pojmi

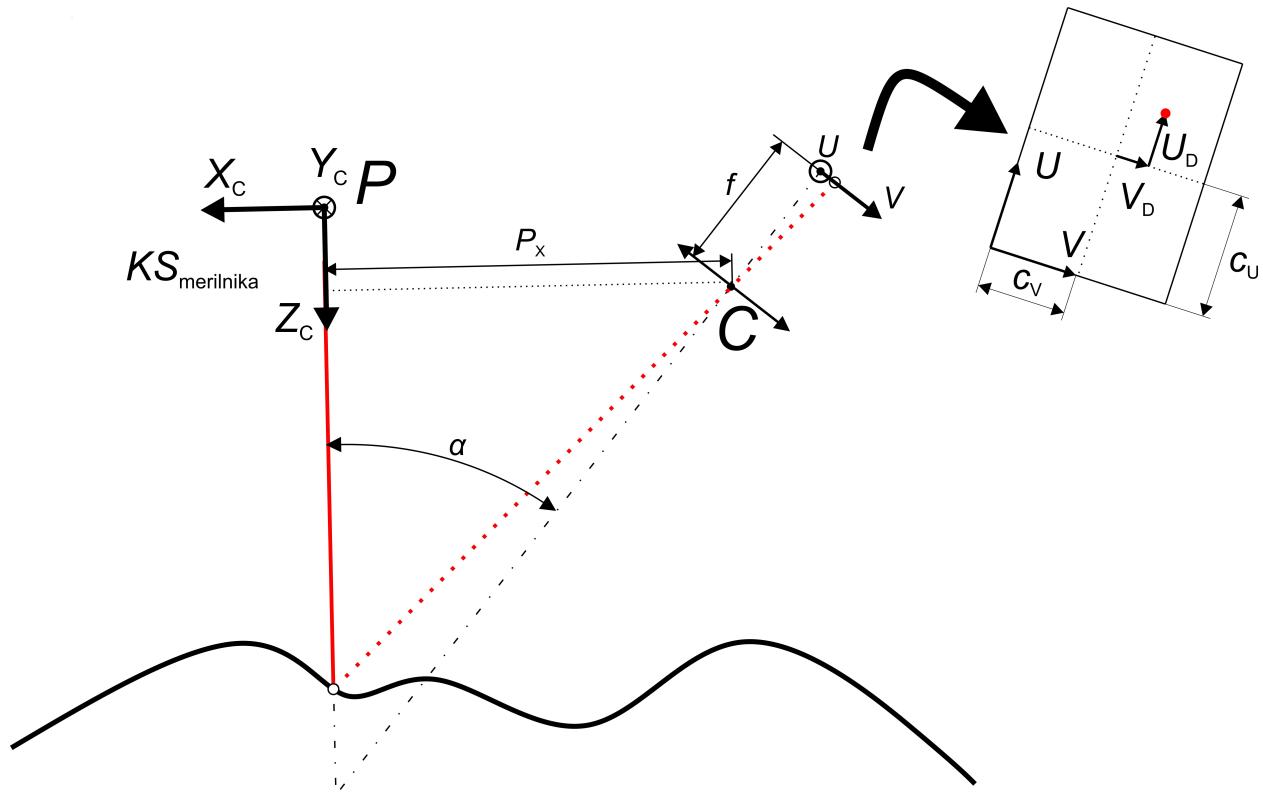


Fig. 1: Slika 1. Shema triangulacije z označenimi merami.

Skica merilnega sistema je prikazana na *sliki 1*. Projektor je postavljen na mestu  $P$  in projicira svetlobno ravnino proti merjeni površini. Kamera je postavljena na mestu, označenem s  $C$ . Koordinatni sistem (*k. s.*) merilnika ( $KS_{merilnika}$ ) je postavljen na presečišče laserske ravnine in pravokotne projekcije središčaleče na lasersko ravnino. Model preslikave temelji na uporabi perspektivne projekcije z enostavno geometrijo kamere z luknjico (*camera obscura*, angl. *pinhole camera*). Senzor je postavljen na razdalji  $f$  od koordinatnega izhodišča. Optična os objektiva preboda ravnino kamere na koordinati ( $c_U, c_V$ , angl. *principal point*) merjeno glede na zgornji lev rob senzorja (standard pri navajanju položaja točke pri senzorjih, zaslonih, slikah etc.). Projektor je glede na koordinatno izhodišče premaknjen le v  $X$  smeri za razdaljo  $P_x$  in projicira svetlobno ravnino pod kotom glede na optično os objektiva kamere.

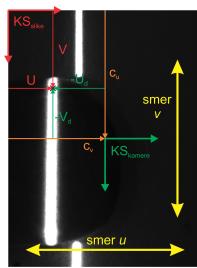


Fig. 2: Slika 2. Ena iz množice slik. Poiskati moramo lokacijo sentra presečne krivulje v k.s. kamere.

Na *sliki 2* je prikazana slika, kot jo vidi kamera. Vidimo lahko, da je merjena površina v osrednjem delu nekoliko dvignjena glede na spodnji in zgornji del. Označena sta *k. s.* slike ( $KS_{slike}$ ) in *k. s.* kamere ( $KS_{kamere}$ ), ki leži na mestu preboda optične osi. Označeni sta tudi smeri  $u$  in  $v$ .

---

**Note:** Slika 2 je transponirana, kar pomeni, da sta smeri  $v$  in  $u$  zamenjani. Sliko transponiramo zato, ker se je laže premikati po vrstici, kot po stolpcu.

---

Merilnik se med merjenjem translatorno premika v  $X$  smeri in ob tem zajel  $N$  slik. Glede na to, da sta hitrosti premikanja in zajemanja slik konstantni, lahko določimo premik merilnika med dvema zaporednima slikama kot:

$$x_{\text{STEP}} = \frac{x_{\text{MOVE}}}{N}$$

kjer sta  $x_{\text{MOVE}}$  celoten premik med merjenjem in  $x_{\text{STEP}}$  premik med dvema zaporednima slikama. Z vsake slike pa lahko razberemo obliko osvetljenega profila površine in tako dobimo površino kot množico zaporednih profilov. Za izračun koordinate posamezne točke profila v  $k$ . s. merilnika, moramo najprej poiskati koordinate posamezne točke profila v  $k$ . s. kamere (na senzorju) na posamezni sliki. V praksi to pomeni, da moramo v vsaki vrstici (za vsak  $v$ ) poiskati center presečne krivulje v smeri  $u$  (koordinato  $u$ ) (glejte sliko 2).

## 7.5 Glavni program

V tem delu programa bomo spisali ogrodje programa. Njegova glavna naloga je, da prebere vsako sliko in s pomočjo funkcije za iskanje presečne krivulje (ki jo bomo napisali ločeno) na vsaki sliki najde lokacijo maksimumov intenzitete. Uporabljali bomo programski jezik *Python* v kombinaciji s knjižnico *OpenCV*. V pomoč pri izdelavi programa naj vam bo diagram poteka (glejte priloga1). Na predvideno mesto (osnova je že pripravljena) napišete jedro programa. Rekonstrukcijo površine boste naredili posebej, kot funkcijo na vnaprej pripravljenem mestu. Bodite pozorni, kako se naslavljajo elemente v matriki (primer:  $B(i, j)$  ... je  $i$  koordinata vrstice ali stolpca ?).



Fig. 3: Slika 3. Skica povezava funkcije `DetectLine()` in matrike profilov.

Zaznane profile na sliki boste zapisali v matriko profilov. Logiko matrike profilov lahko vidite na sliki 3. Dolžina vrstic mora biti enaka dimenziji slike v v smeri (npr. če so slike visoke 640 slikevih elementov (s. e.), mora vsaka vrstica sprejeti po 640 elementov), število vrstic pa mora biti enako številu zajetih slik  $N$  (npr. če smo zajeli 300 slik, torej mora imeti matrika profilov 300 vrstic). Na vsaki sliki bomo tako v vsaki vrstici poiskali položaj presečne krivulje v smeri  $u$ . Npr. če je na 17. sliki v 37. vrstici položaj presečne krivulje na koordinati 103, bomo zapisali `profiles[17, 37] = 103`. Pri pisanju programa naj vam bosta v pomoč Priloga 1 in template programa. Sledite naslednjemu postopku:

1. Na začetku programa najavite in nastavite globalne spremenljivke, ki jih bomo potrebovali tekom celotnega programa (npr. mapa, kjer se nahajajo datoteke, število merilnih slik, alociramo prostor za profile).
2. Začnite zanko.
3. Generirajte ime slike in jo naložite v spomin. Namig: Uporabite `cv2.imread()`
4. Sliko transponirajte (to storite izključno zaradi laže obdelave, saj se je laže premikati po vrstici kot po stolpcu) in jo pretvorite v primeren format (32 bitna števila s plavajočo vejico). Namig: `cv2.transpose(...)`.
5. Na tem mestu kličite funkcijo `detect_profile(...)`. Kot drugi parameter podajte kazalec na vrstico matrike profilov, ki ima isti indeks kot trenutna slika, ki jo obdelujemo (glejte sliko 3).
6. Zaključite zanko.
7. Rekonstruirajte površino.

## 7.6 Funkcija `detect_profile(...)`

Uporabite predlogo `detect_profile(image, profile, filter_size)`.

1. Filtrirajte sliko z *Gaussovim filtrom*. Širina filtrirnega okna mora biti širša od presečne krivulje na sliki! Namig: `cv2.GaussianBlur(...)`
2. Izračunajte odvod slike. Namig: `cv2.Sobel(...)`
3. Poiščite lokacijo prevoja na presečni krivulji.
  - a. Prevoj na sliki je lahko več, a uporabite predpostavko, da je intenziteta na mestu prevoja presečne krivulje večja od prevoj na drugih mestih. Namig: preverite, ali je intenziteta ne tem mestu večja, kot na predhodno zaznanem mestu prevoja.
  - b. Poiščite lokacijo prevoja. Uporabite v prejšnjem koraku izračunani odvod intenzitete. Namig: Iščemo samo maksimume; razmislite, katere prehode torej iščemo.
  - c. Napredno: Poiščite lokacijo središča presečne krivulje (prevoja) s podtočkovno ločljivostjo. Namig: Glejte spodnjo sliko (slika 4).



Fig. 4: Slika 4. Skica za izračun podtočkovne ločljivosti.

4. Rezultat morala biti vektor, ki je enako dolg kot slika v smeri  $v$ .

## 7.7 Rekonstrukcija površine

Normalizirane lokacije presečne krivulje izračunamo po spodnjih enačbah:

$$U_N = \frac{U_d \cdot du}{f} = \frac{(U - c_u) \cdot du}{f}$$

in

$$V_N = \frac{V_d \cdot dv}{f} = \frac{(V - c_v) \cdot dv}{f}$$

kjer sta  $U_N$  in  $V_N$  normalizirani koordinati (smerna vektorja žarka, ki izhaja iz senzorja proti merjeni površini),  $c_u$  in  $c_v$  lokaciji centra slike v smeri  $u$  in  $v$ ,  $du$  in  $dv$  velikosti senzorskega elementa v  $u$  in  $v$  smeri, ter  $f$  goriščna razdalja objektiva kamere. Oddaljenost merjene točke v koordinatnem sistemu merilnika v  $z$  smeri  $z_M$  izračunamo kot:

$$z_M = \frac{P_X}{\tan(\alpha - \arctan(\frac{V_N}{f}))}$$

kjer je triangulacijski kot in  $P_X$  razdalja med kamero in projektorjem. Koordinato točke  $x_M$  izračunamo kot:

$$y_M = z_M \cdot U_N$$

Razmislite, kako izračunamo koordinato  $x_M$ . (Namig: izkoristite for zanko, kakšen je korak med mestoma zajema dveh zaporednih slik?)

## 7.8 Vizualizacija rezultatov

Za vizualizacijo rezultatov uporabite osnovno funkcionalnost Pythona.

## 7.9 Priloge

- priloga1
- priloga2
- priloga3



## UMERITEV LASERSKEGA 3D MERILNIKA

### 8.1 Uvod

Cilj vaje je spoznati osnovne pristope in praktično izvesti umeritev laserskega rotacijskega merilnika.

### 8.2 Oprema

- Laserski rotacijski 3D merilnik – Merkur.
- Koračna miza.
- Referenčna površina.
- Programska oprema za izračun optimalnih transformacijskih parametrov.

### 8.3 Potek vaje

- Ogled sistema.
- Zajem sekvence umeritvenih meritev.
- Izvedba umeritve.
- Primerjava površin pred in po umeritvi.

### 8.4 Predstavitev parametrov rekonstrukcije

- Notranji parametri:
  - **f** goriščna razdalja leče [mm]
  - **cu** lokacija presečišča optične osi leče in senzorske ravnine v U smeri [št. senzorskih elementov]
  - **cv** lokacija presečišča optične osi leče in senzorske ravnine v V smeri [št. senzorskih elementov]
  - **alfa** triangulacijski kot [°]
  - **beta** kot nagiba senzorskega elementa glede na pravokotnico optične osi kamere; pozitivna smer je proti Scheimpflugovi korekciji [°]
  - **psi** rotacija kamere okrog lastne optične osi [°]
  - **Py** lega projektorja glede na kamerin k. s. v Y smeri [mm]

- **Pz** lega projektorja glede na kamerin k. s. v Z smeri [mm]
  - **du** dimenzija senzorskega elementa v U smeri [mm]
  - **dv** dimenzija senzorskega elementa v V smeri [mm]
  - **k0** parameter distorzije kamere
  - **k1** parameter distorzije kamere
  - **k2** parameter distorzije kamere
  - **k3** parameter distorzije kamere
  - **Ry** lega k. s. projektorja glede na vrtičje merilnika (k. s. modula) v Y smeri [mm]
  - **Rz** lega k. s. projektorja glede na vrtičje merilnika (k. s. modula) v Z smeri [mm]
  - **Ko\_rot** korekcija zasuka projektorja okrog njegove osi (odstopanje od vzporednosti s kamero) [°]
  - **domega** kotni zasuk mizice pri nem koraku motorja [°]
- **Zunanji parametri**
    - **TX** translacija v X smeri [mm]
    - **TY** translacija v Y smeri [mm]
    - **TZ** translacija v Z smeri [mm]
    - **RX** rotacija okrog X osi [°]
    - **RY** rotacija okrog Y osi [°]
    - **RZ** rotacija okrog Z osi [°]

## 8.5 Nastavitev programa

1. Z uporabo merilnega traku, kotomera in priložene dokumentacije ocenite začetne približke transformacijskih parametrov. Parametri, ki jih je potrebno oceniti so:
  - **cu**; kot začetni približek vzemite polovično število slikovnih elementov v U smeri
  - **cv**; kot začetni približek vzemite polovično število slikovnih elementov v V smeri
  - **alfa**
  - **Py**
  - **Pz**
  - **du**; poglejte v dokumentacijo kamere
  - **dv**; poglejte v dokumentacijo kamere
  - **domega**
2. Ostale parametre bomo določili s pomočjo umeritvenega programa.
3. Ko ste ocenili začetne približke, odprite LabView program calibration for rotational profilomer (ver 2\_0 for prf). Spreminjate lahko naslednje nastavitev:
  - a. **Detection parameters:**
    - i. **NofSurf** ... število odprtih umeritvenih meritev
    - ii. **first srf** ... indeks prve meritve

- iii. **SurfIncr** ... kolikšen je inkrement med dvema meritvama (1 ... zaporedne meritve, 2 ... vsaka druga itd ...); če povečujete, je potrebno ustrezno povečati DZ (pomik med dvema zaporednima odprtima meritvam)
- iv. **first N prf., last N prf., prf. start, prf. end** ... parametri, s katerimi »obrežemo meritev«, uporabite v primeru šuma o robovih meritve

#### b. TRP & opt out

4. Ostale parametre spreminja le v nujnih primerih in po posvetovanju z asistentom.

## 8.6 Zagon optimizacije

Predlagamo, da za umerjanje uporabite naslednji »algoritem«:

1. Ob nastavljenih začetnih približkih odprite eno samo meritve in jo vizualno primerjajte z idealno meritvijo (**gumb draw** ideal). Če so odstopanja res velika (predvsem **merilo**), preverite začetne približke.
2. Če meritve *ni vsaj približno* v merilu (na merilo v Y smeri vpliva predvsem parameter **domega**), poskusite ročno spreminjati kot triangulacije (**alfa**), kot med dvema profiloma (**domega**) in razdaljo med kamero in projektorjem (**Py**) ter goriščno razdaljo leče (**f**).
3. Zaženite optimizacijo zunanjih parametrov (**TX, TY, TZ, RX, RY, RZ**), da poravnate površino z referenčno (kanali morajo sоппадати).
4. Ko standardna deviacija odstopkov med izmerjeno in referenčno površino pada pod 3-4 mm, lahko (za kratek čas) poskusite vklopiti tudi optimizacijo kota triangulacije (**alfa**), goriščne razdalje objektiva (**f**), kota med dvema profiloma (**domega**), sredine slike (**cu, cv**) in razdalje med kamero in projektorjem (**Py**). Rezultat vseskozi vizualno nadzorujte. Če se meritve povsem »podre«, optimizacijo ustavite.
5. Če se oblika površine ni povsem »podrla«, odprite dodatne meritve in preverite, kako vplivajo na standardno deviacijo odstopkov (SDO). Če je oblika vseh rekonstruiranih površin približno pravilna, SDO pa je drastično narasla, preverite, ali sta **SurfInc in DZ** pravilna.
6. Če imate občutek, da umeritev konvergira v pravo smer (smo v globalnem minimum), vklopite umeritev vseh parametrov, se udobno namestite in čakajte ... Ko pada standardna deviacija pod približno 1,3 mm ste (skoraj) zagotovo v globalnem minimumu in **lahko optimirate vse parametre**.

Če se vam meritve povsem podre, preverite, kateri parameter je ušel in se vrnite korak ali dva nazaj.

## 8.7 Zabeležite rezultate

Končni rezultat, ki ga morate predstaviti v poročilu, je SDO med referenčno in izmerjeno površino in dobljeni transformacijski parametri. Naredite tudi »screen shot« polja odstopkov (**gumb draw diff**) in ga priložite poročilu. Primerjajte začetno in končno referenčno površino, ter preverite, kako parametri, ki so težko izmerljivi/ocenljivi vplivajo na obliko rekonstruirane površine in SDO.



## OBDELAVA 3D MERITEV

### 9.1 Uvod

Pri tej nalogi boste z rotacijskim laserskim profilomerom, ki smo ga umerili na prejšnji vaji, izmerili serijo meritov istega objekta z več različnih pogledov. Meritve boste nato obdelali, sestavili in poravnali v programskem okolju Geomagic Studio (GM).

### 9.2 Izmerite merjeni objekt iz več pogledov

Merjeni objekt postavite pred merilnik na primerno oddaljenost in ga izmerite. Pri tem poskrbite, da se med zajemanjem posamezne meritve merilnik in merjeni objekt ne bosta premikala. V nasprotnem primeru lahko pride do nenatančnih meritov, ki jih ne bo mogoče poravnati. Zato bodite med merjenjem **pazljivi in natančni**. Objekt izmerite iz toliko pogledov, da bo izmerjena celotna površina. Pazite, da bo med dvema sosednjima meritvama dovolj prekrivanja (priporočamo približno 30 % prekrivanje) in da je na območju prekrivanja kakšna značilka (angl.: *feature*).

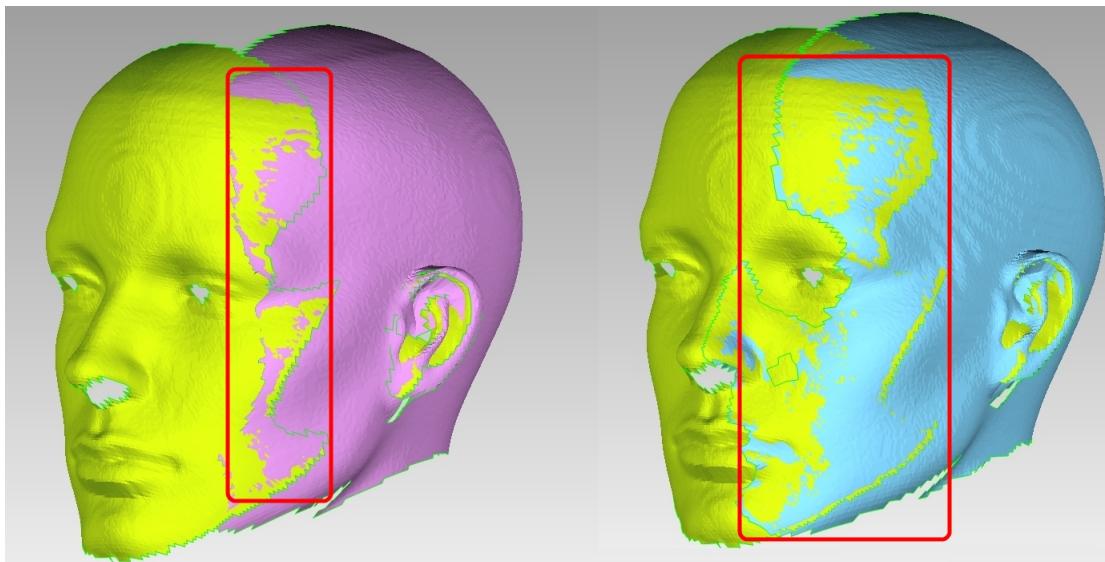


Fig. 1: Slika 1. Prenizka stopnja prekrivanja (levo), primerna stopnja prekrivanja (desno).

## 9.3 Uvoz množice meritev v GM

V GM bomo meritve uvozili v obliki standardnega trikotniškega formata *.VRML* (včasih imenovan tudi *.wrl*) s funkcijo **Import** (**File → Import**).

---

**Note:** Naslednjo meritev izberete s tipko *F3*, prejšnjo s tipko *F4*, vse meritve izberete s tipko *F5*, vse, razen izbranih meritve pa skrijete s tipko *F2*.

---

**Warning:** Zaradi velike količine podatkov, ki jih GM obdela ob vsakem klicu funkcije, funkcija *UNDO* deluje le izjemoma, zato **vmesne rezultate shranjujte sproti pod različnimi imeni.**

## 9.4 Priprava meritvev

Precej verjetno je, da je merilnik poleg samega merjenega objekta izmeril tudi del okolice. Ta ni predmet naloge in bo le oteževala poravnava. Posamezno meritev zato najprej »očistite«; izbrišite vse površine, ki ne pripadajo merjenem objektu.

---

**Note:** Izberite vsaj eno točko na površinah, ki jih želite obdržati in izberite *Bounded Components* (**Select→Select Components→Bounded Components**), nato pa desni klik in *Reverse Selection* (**RMC→Reverse Selection**). Tako boste izbrali vse površine, ki niso sklenjene z na začetku izbranimi točkami.

---

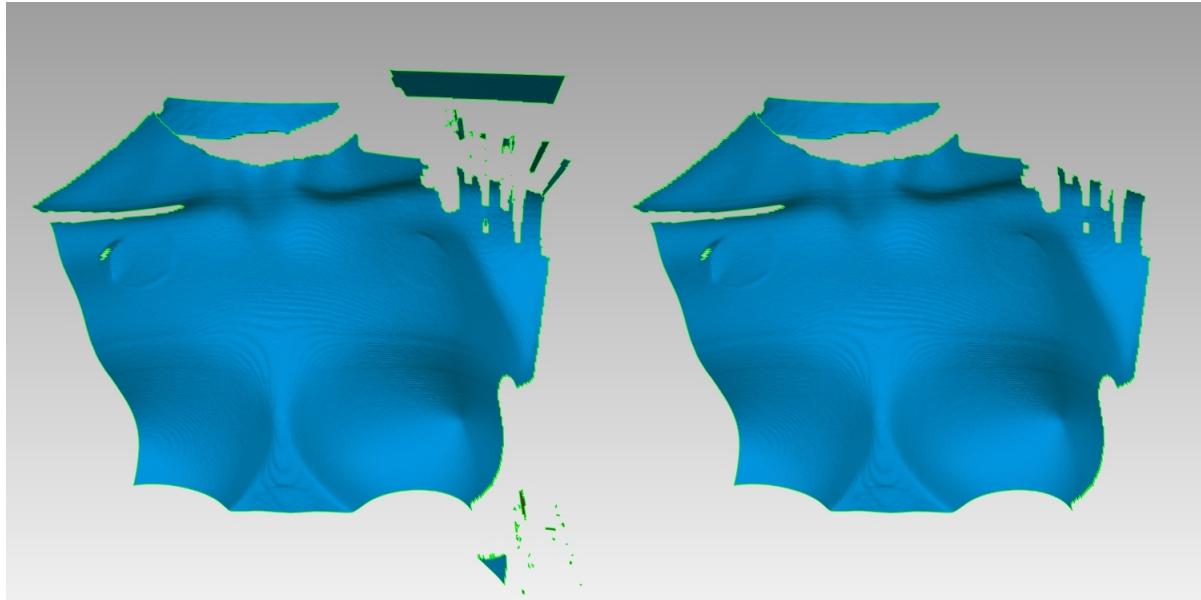


Fig. 2: Slika 2. Izvorna meritev (levo) in »očiščena« meritev (desno).

## 9.5 Ročna poravnava (sestavljanje)

Meritve najprej poravnajte ročno. Pri tem koraku ne izgubljajte preveč časa s pretirano natančno poravnavo, saj boste to storili v naslednjem koraku z uporabo temu namenjenih funkcij. Dovolj je, da so meritve vsaj približno pravilno medsebojno orientirane. Najprej izberite vse meritve, ki jih želite poravnati in izberite *Manual Registration (Alignment→Manual Registration)*. Na levi se odpre pogovorno okno, kjer izberete način poravnave (**A**), površino, ki bo fiksna (**B**) in površino, ki jo bomo poravnali na fiksno (**C**). Izbirate lahko med poravnavo z določitvijo ene soležne točke (*IRP*) ali več kot treh soležnih točk (*NPR*). Način *IRP* je uporaben predvsem v primerih, ko ne najdete veliko soležnih točk, hkrati pa je prekrivanje razmeroma veliko. Pred uporabo tega načina je priporočljivo, da obe površini orientirate v približno enak pogled. Način *NPR* uporabljate kadar lahko najdete množico soležnih točk, želite hitreje zlagati površine ali poravnava z *IPR* ne uspe. Ni potrebe, da bi določili  $N$  povsem istih točk; manjše razlike v mestu klika nimajo vpliva na algoritem, saj se nato običajno naknadno sproži še funkcija *Register* (**D**), ki odpravi manjše odstopke med poravnanimi površinama. Če ste z rezultatom zadovoljni pritisnite gumb *Next* (**E**) in na poravnani površini poravnajte naslednjo površino. Tako (na grobo) zložite skupaj vse meritve. Pomembno je, da ob tem vizualno preverjate, kako so površine medsebojno orientirane, saj lahko kakšna površina »uide«.

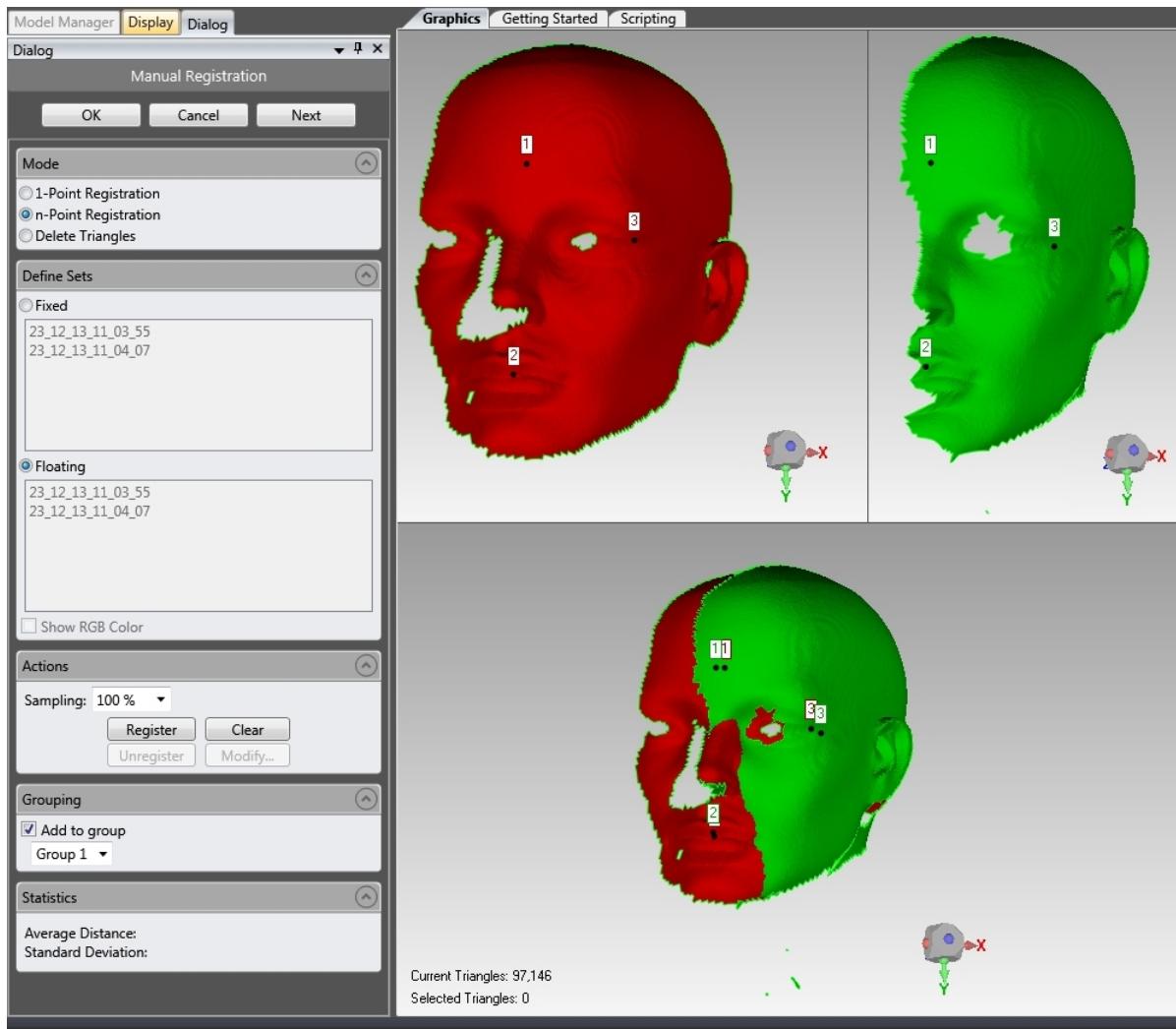


Fig. 3: Slika 3. Slika uporabniškega vmesnika z označenimi osnovnimi gumbi, ki jih boste potrebovali.

## 9.6 Globalna poravnava

Ko ste »na grobo« poravnali oz. sestavili vse površine zaženite funkcijo *Global Registration (Alignment→Global Registration)*. Tako bo algoritem »na fino« poravnal vse površine med seboj. Rezultat vizualno preverite, predvsem na mestih, kjer bi bila morebitna neuspešna poravnava še posebej očitna.

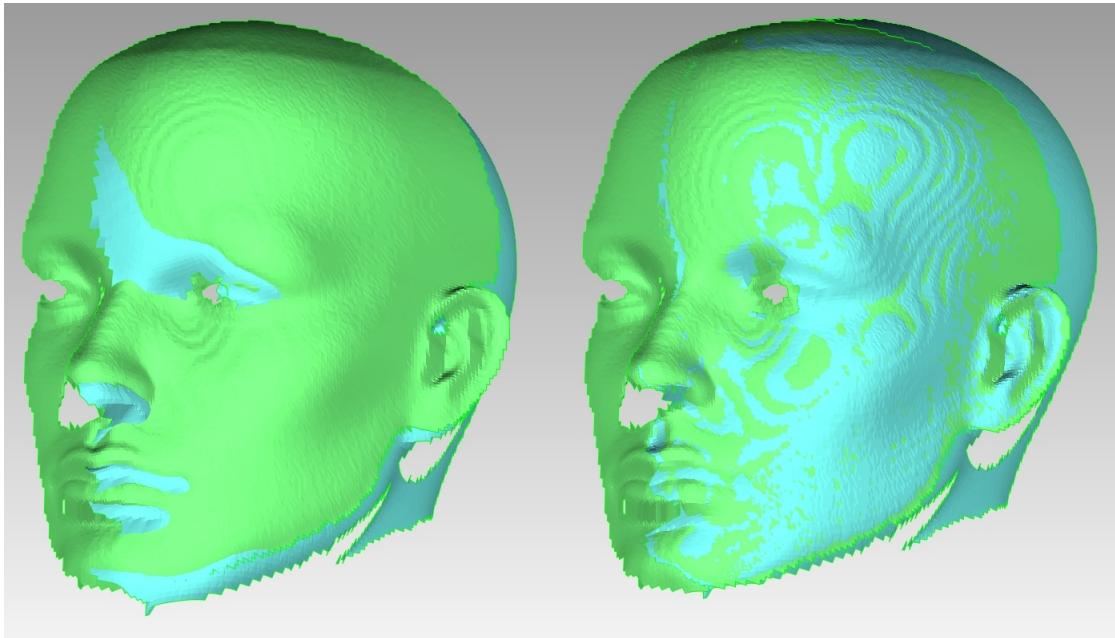


Fig. 4: Slika 4. Meritvi po ročni on pred globalno poravnavo (leva), meritvi po globalni poravnavi (desno).

## 9.7 Končna obdelava

Ko so vse površine poravnane tudi »na fino« jih združite v eno samo površino. To storite z ukazom *Merge (Polygons→Merge)*. Po želji nastavite lokalno in globalno filtriranje, želeno število točk, dodatno globalno poravnavo ipd... Na mestih, kjer se posamezne meritve prekrivajo ali je prisotnega nekoliko več šuma pogosto pride do nepravilnega kombiniranja meritev, kar ima za posledico nepravilno obliko površin. GM nam ponuja vrsto orodij, s pomočjo katerih lahko te nepravilnosti odpravimo. V tej fazi se lotite tudi morebitnega odstranjevanja reliefnih markerjev (če ste jih uporabljali). Prva metoda je, da reliefne markerje iz površin enostavno izrežemo, druga pa, da uporabite funkcijo *Defeature (Polygons→Defeature)*. Ker je uporaba markerjev potrebna le na površinah, ki nimajo izrazitih geometrijskih značilnosti, se interpolacijske metode zapolnjevanja luknenj izkažejo za precej uspešne in na površini ni vidnih ostankov markerja. Če so ti prisotni, je bilo izbrano mesto izbrano neposrečeno.

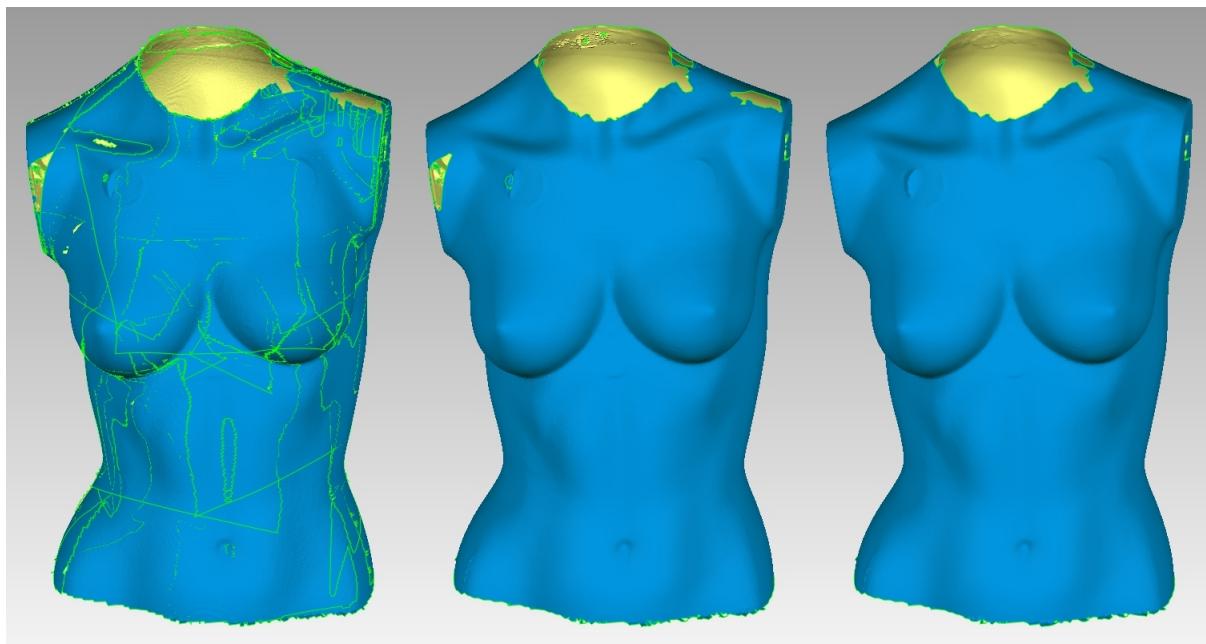


Fig. 5: Slika 5. Poravnane meritve (levo), združene meritve (sredina), končni rezultat merjenja z zapolnjenimi luknjami in zglajenimi vidnimi šivi (desno).



## VLAKENSKI SENZORJI

### 10.1 Uvod

Na vaji boste spoznali dva vlakenska senzorja: *vlakenski senzor pomika* in *vlakenski fluorescentni termometer*. Spoznali boste osnovne principe delovanja, področja uporabe tovrstnih senzorjev in izvedli umeritev obeh senzorjev.

### 10.2 Vlakenski senzor pomika

Principle delovanja optičnih vlaken ste že spoznali. Osnovni pojav je totalni odboj, ki se vrši na meji med jedrom in plaščem vlakna, kadar je vpadni kot žarka glede na normalo meje dovolj velik; večji od mejnega kota  $\Theta_C$ .

Če je vlakno ravno, moramo tako le zagotoviti, da žarek v vlakno uvedemo pod kotom, ki je manjši od  $\Theta_{NA}$  in vsa uvedena svetloba bo izšla tudi iz vlakna. Mejni vstopni kot  $\Theta_{NA}$  izračunamo kot:

$$NA = \sin\theta_{NA} = \frac{1}{n_0} \sqrt{n_J - n_P}$$

kjer je  $NA$  numerična odprtina vlakna,  $n_0$ ,  $n_J$  in  $n_P$  pa so lomni količniki *oklice, jedra in plašča*.

Če optično vlakno ukrivimo lahko ugotovimo, da intenziteta izstopne svetlobe pade. Zakaj? Na ukrivljenem delu se v tem primeru meja med jedrom in plaščem nagnjena glede na žarek in tako je za del žarkov vpadni kot zmanjšan. Žarki, za katere se vpadni kot zmanjša pod dopustno mejo ( $\Theta_C$ ), iz vlakna izidejo in na pridejo do konca vlakna (*slika 1*).

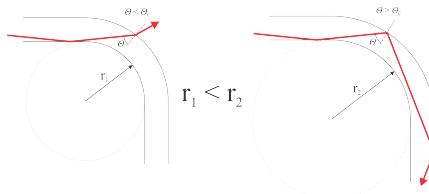


Fig. 1: Slika 1. Na levi strani je skica pogojev, ki nastopijo pri krivljaju vlakna za premajhen radij, na desni strani pa ukrivljost še omogoča vodenje žarka.

Omenjeni pojav je običajno neželen, saj nas omejuje v možnostih, kako lahko vlakno položimo. A pojav lahko tudi izkoristimo v naš prid. Shema sistema za merjenje pomika na osnovi ocenjevanja zmanjševanja izhodne intenzitete je vidna na *sliki 2*. Svetlobo od laserskega vira vodimo po optičnem vlaknu do fotodiode, ki meri količino vpadle svetlobe. S pomočjo sistema »grabljic« v osrednjem delu vlakno krivimo. Pri tem so spodnje grabljice nepremične, zgornje pa lahko primikamo ali odmikamo in tako spremenjamo ukrivljjenost vlakna. Bolj ko bodo »grabljice« primanjene, bolj bo vlakno ukrivljeno in manjša bo intenziteta na diodi vpadle svetlobe.

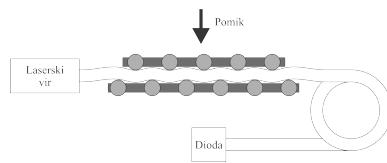


Fig. 2: Slika 2. Sistem "grablje".

### 10.3 Vlakenski fluorescentni termometer

**Fluorescencija** imenujemo pojav, ko snov odda svetlobo po tem, ko je predhodno svetlobo ali drugo elektromagnetno valovanje absorbirala. Pojav se široko izkorišča; predstavljajte si le fluorescentne premaze na primer urah in fluorescentna svetila. Eden izmed materialov, ki fluorescira, je kristal rubina (aluminijev oksid dupiran s kromovimi ioni,  $Al_2O_3Cr$ ). Prav ta kristal je bil uporabljen tudi v prvem delujočem laserskem viru (*T. H. Maiman, 1960*). Gre za tronivojski sistem, ki je prikazan na *sliki 3*.

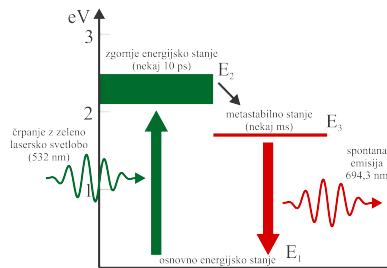


Fig. 3: Slika 3. Energijski nivoji rubinovega laserja.

Atomi v kristalu se običajno nahajajo v osnovnem energijskem stanju  $E1$ . Če ga obsevamo z zeleno svetlobo valovne dolžine približno  $532\text{ nm}$ , bodo atomi začeli prehajati v energijsko stanje  $E2$ , kjer pa lahko obstanejo le kratek čas (nekaj  $10\text{ ps}$ ). Hitro bodo prešli v metastabilno stanje  $E3$ , kjer pa lahko obstanejo precej dlje, nekaj milisekund. Ob prehodu v osnovno energijsko stanje  $E1$  bodo energijo oddali v obliki spontane emisije – fotonov z valovno dolžino  $694,3\text{ nm}$ .

Kako hitro se bo zgodil prehod iz metastabilnega v osnovno stanje, opisuje razpolovni čas . Ta pa je odvisen predvsem od temperature snovi; bolj kot je snov segreta, večja je amplituda nihanja kristalne rešetke, laže odda molekula energijo drugi molekuli. Tako se bo snov še dodatno segrevala (manjši bo izkoristek), pa se bo skrajšal.

Časovni potek prehajanja molekul v osnovno stanje in oddajanja svetlobe lahko opišemo z eksponentno enačbo:

$$I(t) = I_0 \cdot e^{-\frac{t}{\tau}}$$

kjer je  $I_0$  začetna intenziteta.

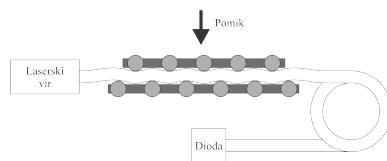


Fig. 4: Slika 4. Intenziteta oddane svetlobe v odvisnosti od časa. S povečevanjem temperature snovi se krajša.

### 10.3.1 Potek vaje

Za merjenje temperature boste uporabljali vlakno, ki ima na konici v epoksidno smolo vmešan fluorescentni rubinov prah (*slika 4*). Vzbujali ga bomo z zeleno lasersko svetlobo, ki jo bomo vodili po vlaknu. Ta bo vzbudila rubinov prah, ki bo fluoresciral še nekaj milisekund po tem, ko bomo laserski vir ugasnili. Del oddane svetlobe rubina bo potoval po vlaknu proti fotodiidi.

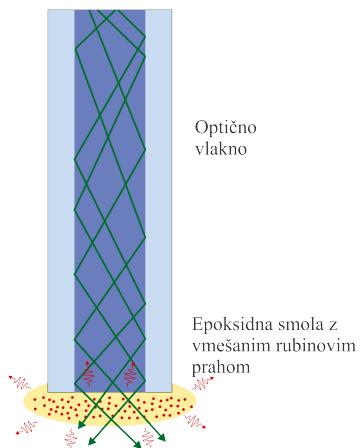


Fig. 5: Slika 5. Skica vlakenskega senzorja za merjenje temperature.