UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Tine Ileršič

Naprava za dimenzijsko kontrolo lesenih veznih lamel

DIPLOMSKO DELO

UNIVERZITETNI ŠTUDIJ RAČUNALNIŠTVA IN INFORMATIKE

MENTOR: izr. prof. dr. Uroš Lotrič

Ljubljana, 2015

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Naprava za dimenzijsko kontrolo lesenih veznih lamel

Tematika naloge:

Razvijte napravo za avtomatsko dimenzijsko kontrolo lesenih veznih lamel. Izberite ustrezno strojno opremo za krmiljenje naprave in izdelajte potrebno programsko opremo, ki naj vključuje zajemanje slik in njihovo obdelavo, kontrolo veznih lamel ter razvrščanje in statistične analize.

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Spodaj podpisani Tine Ileršič sem avtor diplomskega dela z naslovom:

Naprava za dimenzijsko kontrolo lesenih veznih lamel

S svojim podpisom zagotavljam, da:

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

V Ljubljani, dne 5. aprila 2015 Podpis avtorja:

Zahvaljujem se mentorju prof. dr. Urošu Lotriču za strokovno pomoč, svetovanje in usmerjanje pri izdelavi diplomske naloge. Zahvala tudi moji družini za podporo v času študija. Prav tako bi se zahvalil direktorju podjetja MBvision d.o.o. Miroslavu Barutu za pomoč pri izvedbi projekta, sodelavcema Nataši Hribar in Janezu Udoviču za pomoč pri izdelavi programske opreme ter ostalim sodelavcem.

Kazalo

Slike

Tabele

Seznam uporabljenih kratic

kratica	angleško	slovensko
IDE	Integrated Development Environment	Integrirano razvojno okolje
CLI	Common Language Infrastructure	Skupna jezikovna infrastruktura
XML	Extensible Markup Language	Razširljiv označevalni jezik
XSLT	Extensible Stylesheet Language Transformations	Jezikovne pretvorbe razširljivih jezikovnih polj
HTML	Hyper Text Markup Language	Označevalni jezik za oblikovanje večpredstavnostnih dokumentov
XHTML	Extensible Hyper Text Markup Language	HTML, usklajen s sintakso XML
CSS	Cascading Style Sheets	Stilna predloga na spletni strani
VB	Visual Basic	Programski jezik Visual Basic
MSVC	Microsoft Visual C++	Razvojno okolje Microsoft Visual C++
GUI	Graphical User Interface	Grafični uporabniški vmesnik
WDM	Windows Driver Model	Ogrodje za gonilnike za operacijske sisteme Windows
I/O	Input / Output	Vhodno-izhodni
DDK	Microsoft Driver Development Kit	Paket za razvoj gonilnikov podjetja Microsoft
IRQ	Interrupt Request	Zahteva za prekinitev
DMA	Direct Memory Access	Neposredni dostop do pomnilnika
TTL	Transistor - Transistor Logic	Tranzistorsko-tranzistorska logika
LPT	Local Printer Terminal	Lokalni tiskalniški terminal

Povzetek

V diplomskem delu je predstavljena izdelava naprave za dimenzijsko kontrolo lesenih veznih lamel. Opisana je avtomatska mehanska linija naprave, ki zajema zalogovnik, dvigalo, rotacijski vibracijski dodajalnik, merilno mesto ter sistem za ločevanje dobrih in slabih lamel. Osrednji del diplomskega dela predstavlja razvoj programske opreme v okolju Microsoft Visual Studio 2013.

Programska oprema zajema krmiljenje vhodno-izhodnih signalov preko vzporednih vrat ter 32-kanalnega digitalnega vhodno-izhodnega modula, sistem programskih prekinitev za obdelavo slik v realnem času, funkcije za sledenje kosov in algoritem za iskanje povezanih komponent, funkcije meritev ter grafični uporabniški vmesnik.

Ključne besede: strojni vid, kontrola lamel, obdelava slik v realnem času, algoritem za sledenje, algoritem za iskanje povezanih ploskev

Abstract

The aim of this thesis is to present the development of a wood biscuit measuring and sorting system. A description of an automatic mechanical line consisting of a reservoir, an elevator, a vibrating rotary supplier, a measuring system and a sorting system that separates perfect wood biscuits from imperfect ones is included. The core part of the thesis is a presentation of software development in Microsoft Visual Studio 2013.

The developed software includes controlling input/output signals via a parallel port and a 32-channel digital input/output module, a system of software interruptions for real time picture processing, functions that allow tracing of individual items, an algorithm for connected-component labelling, measuring functions and a graphic user interface.

Key words: machine vision, wood biscuit control, real time picture processing, tracing algorithm, algorithm for connected-component labelling.

1. Uvod

Na trgu se je pojavila potreba po hitri in zanesljivi napravi za dimenzijsko kontrolo lesenih veznih lamel. V podjetju, v katerem sem zaposlen, so v preteklosti podobno napravo že izdelali, vendar so se s časom pokazale določene pomanjkljivosti in napake. Zaradi tega smo se odločili, da izdelamo napravo, ki bi te napake odpravila in povečala zanesljivost delovanja ter kapaciteto.

Izdelavo naprave smo razdelili na več segmentov. V prvi fazi smo preučili mehanske pomanjkljivosti prejšnje naprave in določili osnovni princip delovanja nove. Nato smo za potrebe testiranja izdelali prototip naprave. Vzporedno z izdelavo modela je potekala tudi priprava načrtov in izdelava strojne opreme ter prvih testnih verzij programske opreme. Sledila je priprava konstrukcijskega načrta ter ostale potrebne dokumentacije za izdelavo mehanskih delov naprave. Pripravljeni dokumentaciji je sledila realizacija mehanike, dokončna izdelava programske opreme, testiranje naprave in priprava navodil za uporabo ter ostale dokumentacije za končnega uporabnika.

V diplomskem delu je v grobem predstavljena izdelava celotne naprave, saj sem kot vodja projekta sodeloval pri skoraj vseh korakih izdelave. Moja primarna naloga pa je bila priprava programske opreme ter nekaterih delov strojne opreme, zato je ta del procesa v diplomski nalogi predstavljen podrobneje.

Osnovni cilj izdelave programske opreme je bil zagotoviti hitro, zanesljivo in robustno delovanje. Zanesljivost programske opreme je pri teh napravah ključnega pomena. Te namreč običajno obratujejo 24 ur na dan po 10 let in več, vsak izpad delovanja pa uporabniku povzroča dodatne stroške. Drugi cilj je bil, da v napravo vgradimo čim več dodatnih funkcij, ki bi le-tej zagotavljale dodano vrednost ter prednost pred morebitnimi konkurenčnimi napravami. Naprava namreč izmeri vsako vezno lamelo posebej, zato lahko s pomočjo grafičnih prikazov ter statistike uporabniku zagotovimo veliko koristnih informacij za optimizacijo proizvodnega procesa.

2. Dimenzijska kontrola lesenih veznih lamel

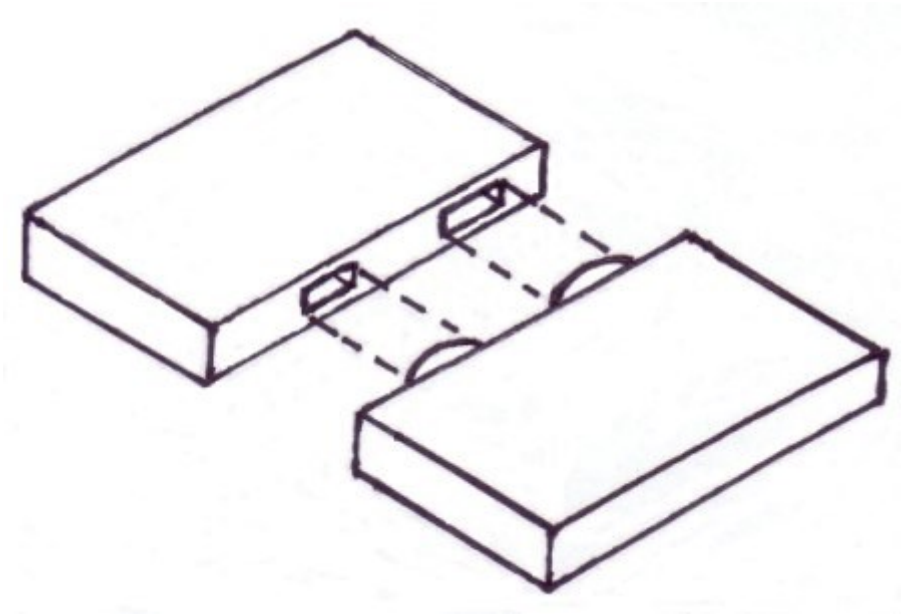
2.1. Opis problema in zahteve

Proizvajalci lamel želijo svojim kupcem ponuditi izdelke najvišje kakovosti, zato se je pojavila potreba po stoodstotni končni kontroli izdelkov. Ker v proizvodnem procesu takšne kakovosti ne morejo zagotoviti, želijo izdelke naknadno pregledati s kontrolnim sistemom. Ta mora vsaki lameli posebej izmeriti dolžino, širino, debelino, ploščino in odkriti morebitne poškodbe oziroma napake. Naprava mora znati pregledovati vse tri tipe lamel z možnostjo nadgradnje za kontrolo ostalih tipov, v kolikor bi kupec to želel. Naprava mora delovati avtomatsko, z velikim zalogovnikom na začetku in dvema ločenima zabojema na koncu, enim za lamele, ki ustrezajo nastavljenim kriterijem, in enim za neustrezne kose.

Kapaciteta stare kontrolne naprave znaša približno 6.000–7.000 lamel na uro. Naš cilj je bil kapaciteto podvojiti.

2.2. Vezne lamele

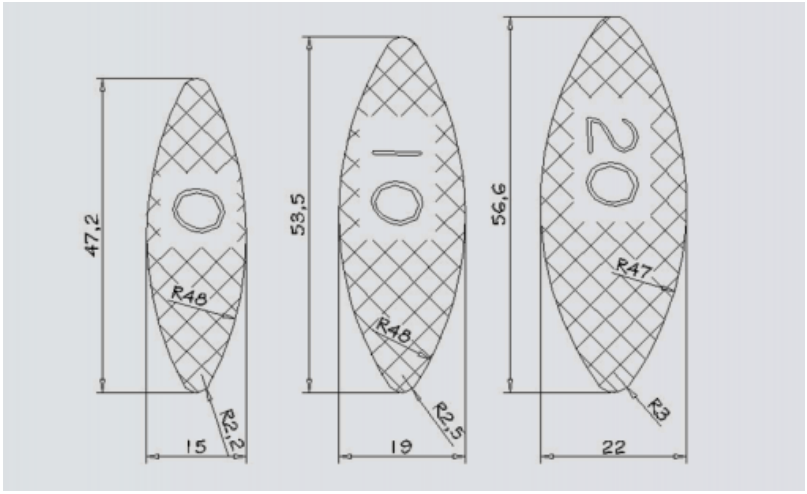
Vezne lamele so lesene ploščice ovalne oblike, namenjene spajanju lesenih elementov, kot je prikazano na sliki 1 [1]. Razvite so bile v petdesetih letih dvajsetega stoletja. Spoj naredimo tako, da v površino vsakega od dveh lesenih elementov, ki ju združujemo, s posebnim orodjem zarežemo ovalno zarezo. V zarezi nanesemo lepilo ter vezno lamelo. Lepilo vezno lamelo razširi in tako nastane močan, čist spoj. Vezne lamele so opazovalcu skrite, zato je ta tehnika spajanja priljubljena pri izdelovanju pohištvenih izdelkov [2].



Slika 1: Prikaz lesenega spoja z uporabo lamel

2.2.1. Tipi in dimenzije

Na trgu obstaja več tipov veznih lamel. Najbolj razširjene so lamele tipov P20, P10, P0, ki se razlikujejo po velikosti in po obliki. V diplomskem delu sem se osredotočil le na omenjene tri tipe, ki so prikazani na sliki 2 [3]; podrobni podatki so v tabeli 1.



Slika 2: Osnovni tipi lamel

Tabela 1: Podrobni podatki tipov lamel

Tip	Dolžina [mm]	Širina [mm]	Radij 1 [mm]	Radij 2 [mm]
P 20	56,6	22,0	47,0	3,0
P 10	53,5	19,0	48,0	2,5
P 0	47,2	15,0	48,0	2,2

2.2.1. Tipične napake

Na sliki 3 lahko vidimo sliko dobre lamele, na sliki 4 pa nekatere tipične primerke slabih lamel. Najpogostejše napake so manjkajoči deli lamele, razpoke na lameli ali pa odlomljeni delci lamele in pa napake na robovih, kadar lamela ni lepo odrezana, oziroma hrapavi robovi.

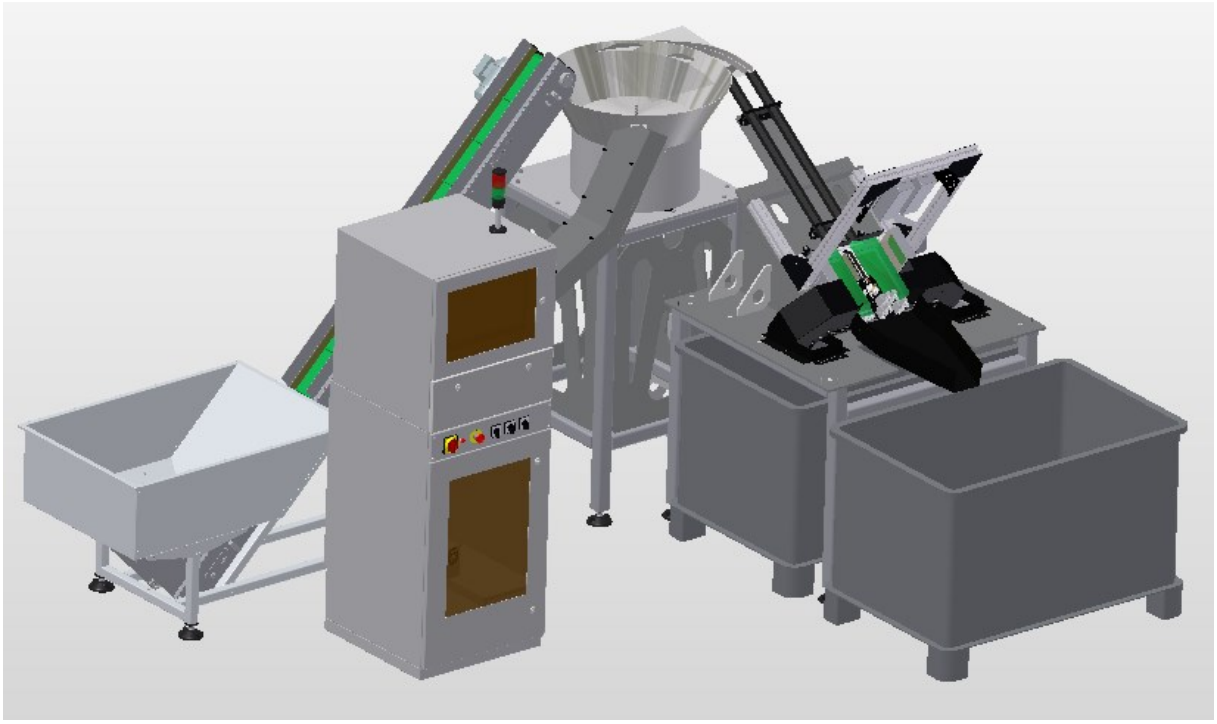
Slika 3: Primer dobre lamele

Slika 4: Primeri slabih lamel

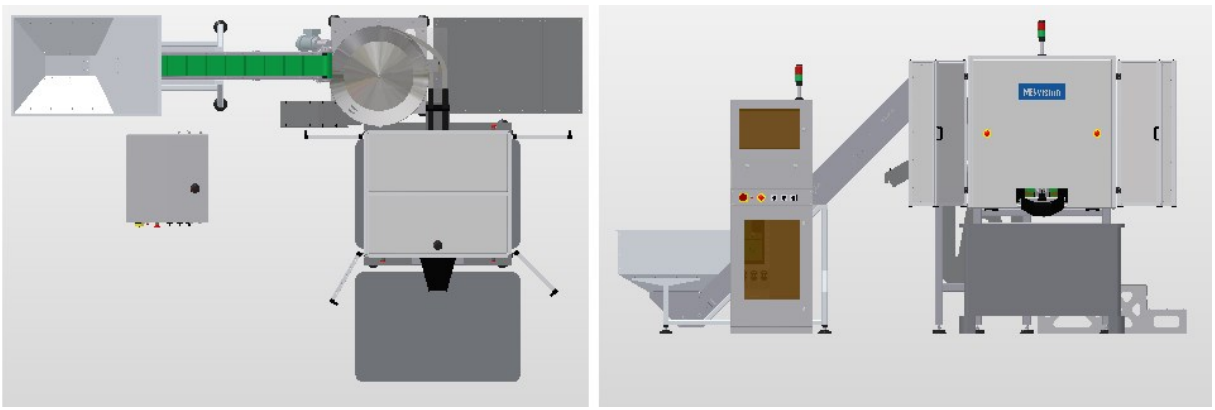
3. Opis naprave

3.1. Mehanika

Na sliki 5 in sliki 6 je prikazan model celotnega sistema. Sestavljajo ga zalogovnik z dvigalom, rotacijski vibracijski dodajalnik, kontrolna enota ter merilna enota.



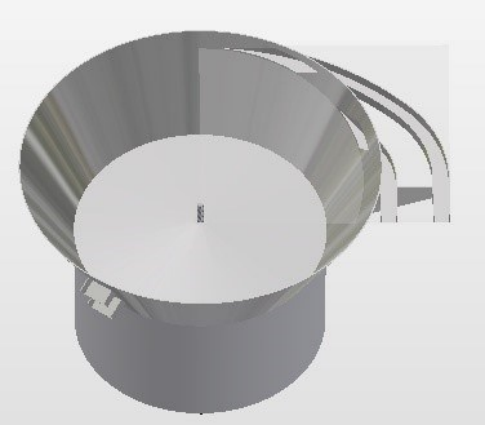
Slika 5: Model celotnega sistema brez zaščitnega pokrova



Slika 6: Model sistema, pogled zgoraj (levo) in spredaj (desno)

Velik zalogovnik s tekočim trakom skrbi za enakomerno dodajanje lamel. Dodajanje krmilimo s pomočjo optičnega senzorja, ki spremlja nivo lamel v rotacijskem vibracijskem dodajalniku.

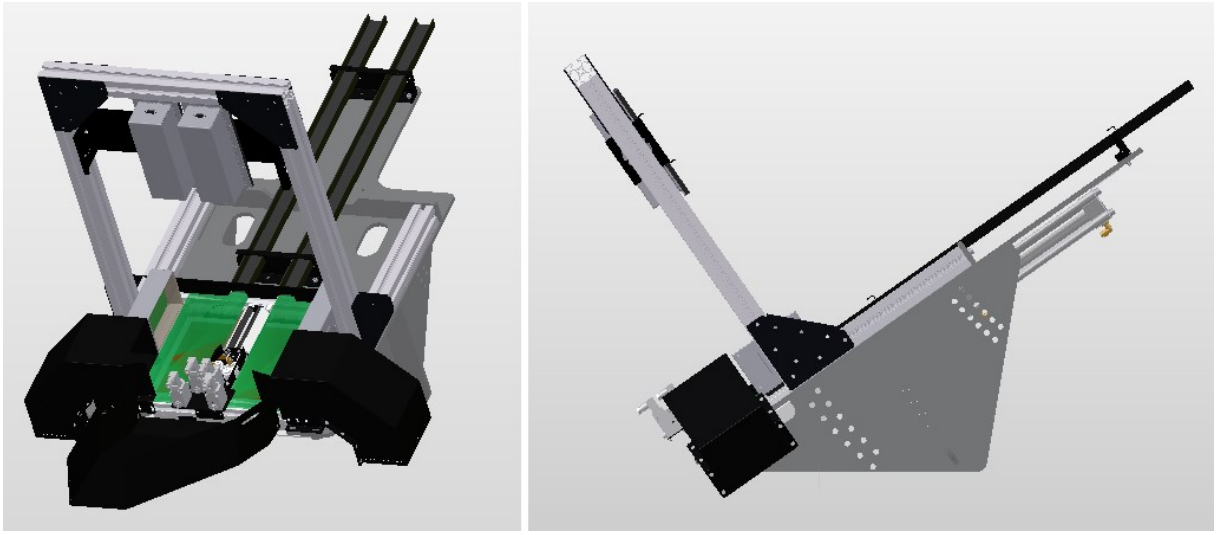
Rotacijski vibracijski dodajalnik, prikazan na sliki 7, ima dve ločeni liniji, deluje torej z dvojno hitrostjo. Pritrjen je na jekleno konstrukcijo in je popolnoma ločen od ostalih enot sistema. S tem se izognemo prenosu vibracij, ki bi lahko vplivale na kvaliteto meritev. Nivo lamel ohranjamo na zmerni stopnji, s čemer dosežemo optimalno delovanje vibratorja.



Slika 7: Rotacijski vibracijski dodajalnik

V kontrolni omari se nahajajo osebni računalnik z zaslonom, elektronika za krmiljenje sistema, stikala za kontrolo delovanja naprave in stikalo za izklop v sili. Omara je neprodušno zaprta in prilagojena za delovanje v prašnem okolju.

Merilna enota je sestavljena iz dveh merilnih mest. Kot prikazuje slika 8, je vsako merilno mesto sestavljeno iz drče, po kateri lamele prihajajo, steklene podlage na samem mestu merjenja, pod katero se nahaja luč, kamere, ki je nameščena pravokotno na stekleno podlago na višini 50 cm, ogledala, ki je na robu steklene površine namenjeno pogledu s strani, stranske luči nasproti ogledala, ventila, ki izpihuje dobre kose v posebni kanal, kanala za slabe kose ter nekaj šob za čiščenje merilnega mesta. Naklon je nastavljiv, z njim uravnavamo hitrost potovanja lamel. Kameri sta v kovinskem ohišju, ki preprečuje vdor vlage in umazanije.



Slika 8: Merilna enota

3.2. Kamera

Na napravi sta nameščeni dve kameri znamke The Imaging Source tipa DMK 23GV024 [4]. Posamezna kamera se napaja z 12V enosmerno napetostjo, prenos podatkov pa poteka preko gigabitne mrežne povezave. Na napravi je uporabljena najvišja ločljivost, ki jo kamera nudi, to je 752 x 480 slikovnih točk. Kamera lahko doseže do 100 slik na sekundo v prostem teku, pri zunanjem proženju pa nekoliko manj.

Senzor kamere je tipa CMOS, velikosti 1/3 inča z globalno zaslonko (ang. *global shutter*) in velikostjo slikovne točke 0,6 x 0,6 μm.

3.3. USB-DIO-32

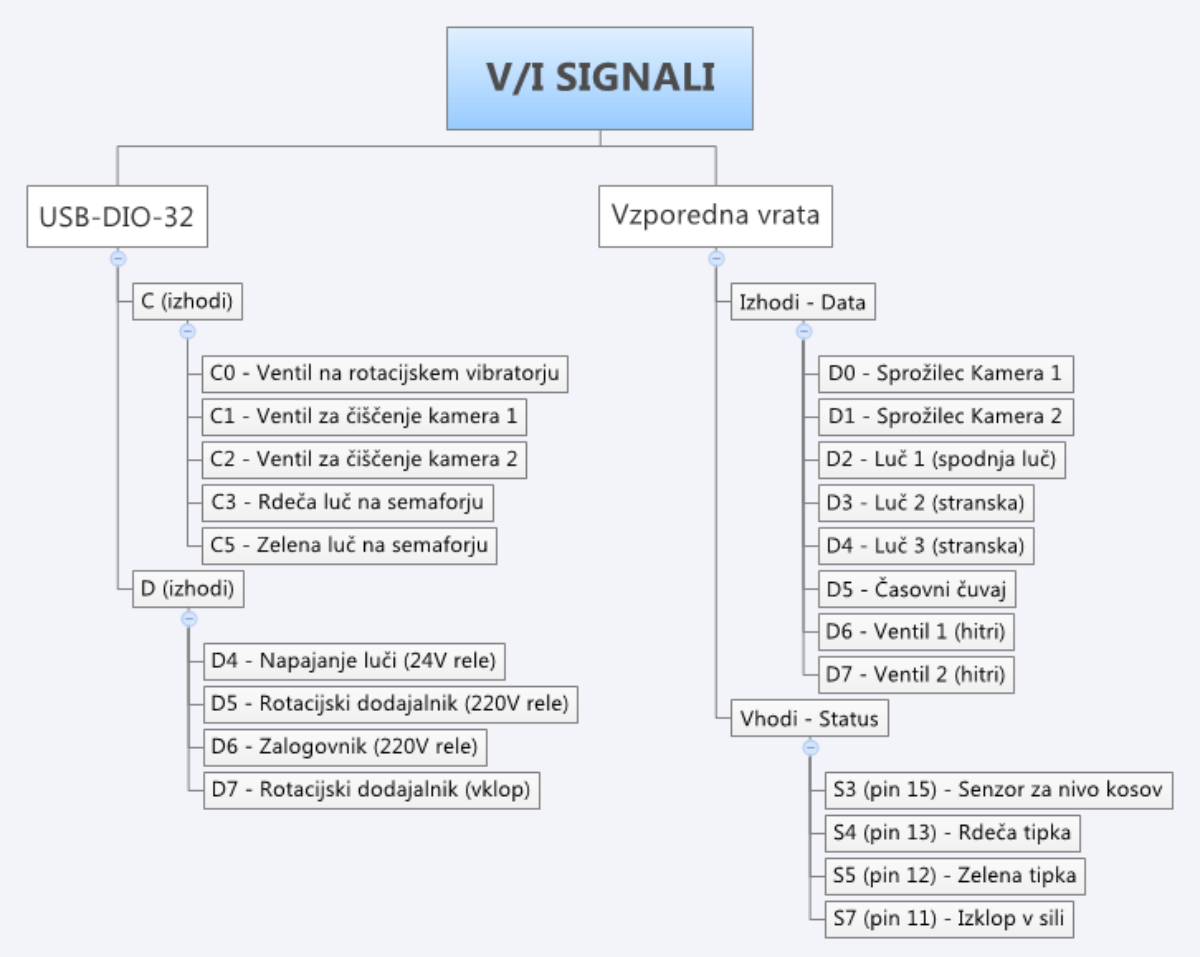
To je 32-kanalni digitalni vhodno-izhodni modul. Modul nudi najvišje možne hitrosti prenosa po standardu USB 2.0. Vsebuje štiri vmesnike s po osem digitalnimi vhodi oziroma izhodi tipa TTL ter tri števnike 82C54. Za vsak vmesnik lahko posebej nastavimo ali bomo na njem uporabljali digitalne vhode ali izhode. Modul lahko napajamo preko priključka USB ali zunanjega napajanja. Signale povežemo na modul preko standardnega 50-pinskega industrijskega priključka. Proizvajalec nudi podporo za okolje Visual C++ ter optimiziran gonilnik za najvišjo prepustnost podatkov [5].

4. Opis programa

4.1. Zasnova programa

4.1.1. Vhodno-izhodni sistem

Vhodne in izhodne signale krmilimo preko dveh vmesnikov, vzporednih vrat in vhodno-izhodnega digitalnega modula USB-DIO-32. Za signale, pri katerih je pomembna hitra odzivnost, uporabljamo vzporedna vrata, ki jih krmilimo s pomočjo gonilnika TVicHW32 6.0. Izmerjeni čas zakasnitve signala znaša približno 50 mikrosekund, kar povsem zadošča za nemoteno krmiljenje sistema. Zaradi omejenega števila signalov na vzporednih vratih, preostale signale krmilimo preko modula USB-DIO-32. Zakasnitve so tukaj večje, vendar zaradi narave signalov to nima vpliva na delovanje sistema. Kot vidimo na sliki 9, uporabljamo 9 izhodnih signalov preko modula USB-DIO-32, 5 signalov preko vmesnika C in 4 signale preko vmesnika D. Preko vzporednih vrat krmilimo 8 izhodnih signalov ter 4 vhodne.

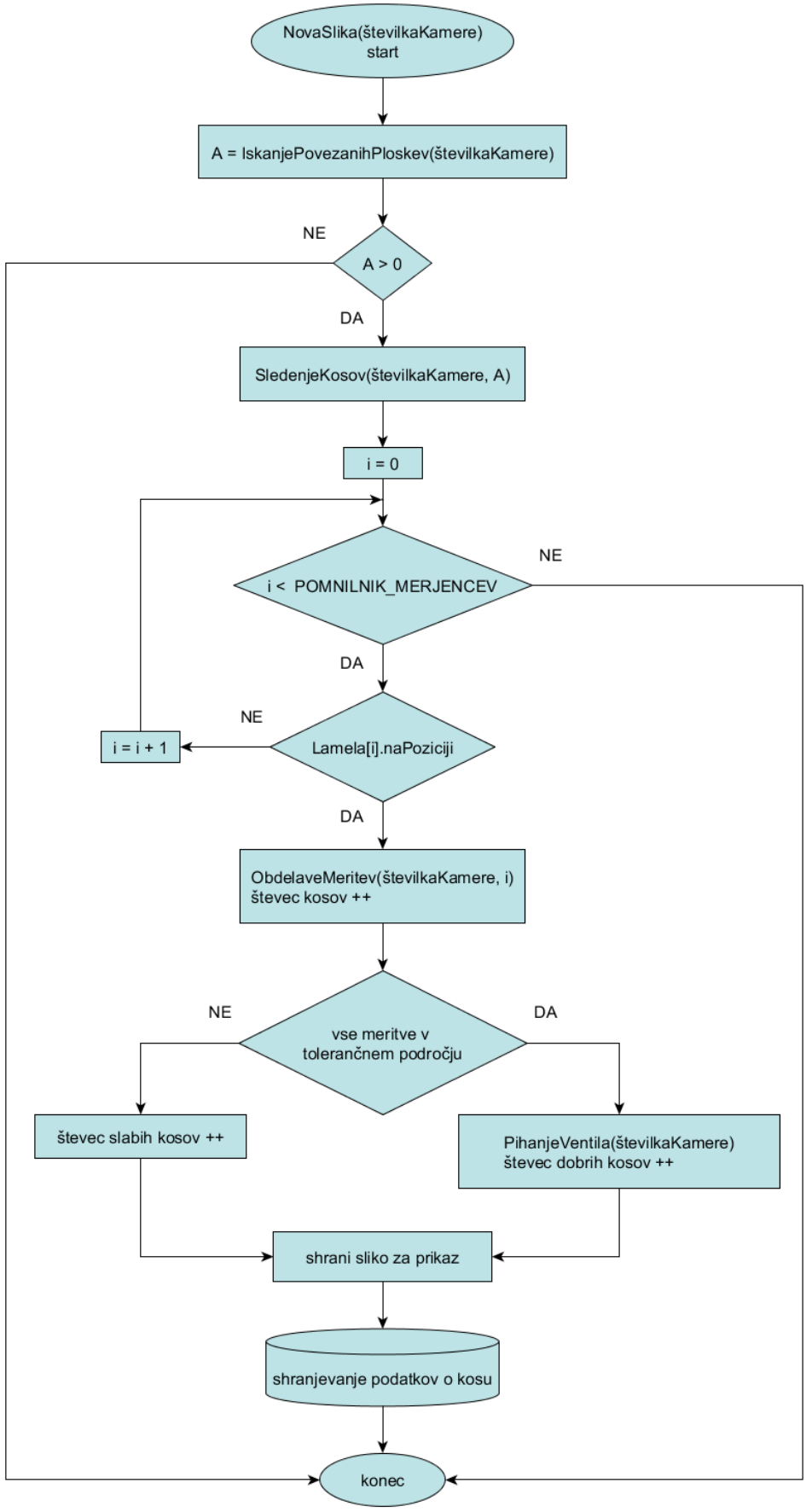


Slika 9: Vhodni in izhodni signali

4.1.2. Prihod nove slike v pomnilnik

Ob vsakem prihodu nove slike v pomnilnik, se izvede funkcija *NovaSlika*. Diagram poteka funkcije je prikazan na sliki 10. Funkciji za obe kameri se izvajata vzporedno. V obhodu funkcije se izvedejo vse s sliko povezane obdelave in vse meritve ter postavijo vsi signali vhodno-izhodnega sistema, zato govorimo o sistemu v realnem času. Časovni okvir izvajanja obdelav je omejen na čas med prihodoma dveh slik, ki pri frekvenci proženja 83,3 Hz znaša 12 milisekund.

Ob prihodu slike se najprej izvede funkcija za iskanje povezanih ploskev. V kolikor se na sliki nahaja vsaj ena ploskev, se izvede funkcija za sledenje, ki najdeno ploskev poveže s stanjem predhodnih slik. Ker je ploskev lahko več, izvede preverjanje ali je katera od teh na poziciji za izvedbo meritev. Ta kriterij lahko v danem trenutku izpolnjuje največ ena ploskev. V primeru izpolnjenega pogoja se izvedejo funkcije meritev, obenem pa se poveča števec izmerjenih lamel. Za vsako meritev se nato izvede preverjanje ustreznosti. V kolikor so izmerjene vrednosti v nastavljenem tolerančnem območju, se izvede funkcija za proženje ustreznega ventila in poveča števec dobrih kosov, v nasprotnem primeru pa se poveča števec slabih kosov. Slika zadnje izmerjene lamele se prepiše v pomnilnik za prikazovanje na zaslonu, izmerjeni podatki pa se shranijo v pomnilnik za prikaz izmerjenih kosov.



Slika 10: Graf poteka funkcije NovaSlika

4.1.3. Prekinitve

4.1.3.1. Hitre prekinitve

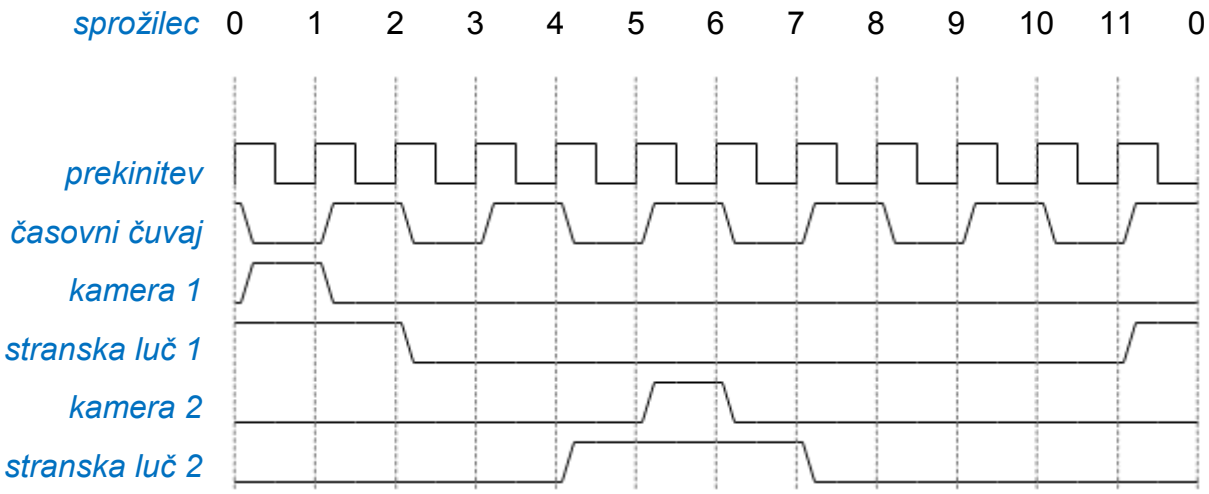
Hitre prekinitve so prekinitve z najvišjo prioriteto in frekvenco delovanja 1 kHz. Vsako milisekundo se v časovniku generira zahteva za prekinitev, pri kateri se izvede funkcija *HitraPrekinitev*. Zaradi ciklično generiranih zahtev je pomembno, da časovna zahtevnost funkcije ne presega ene milisekunde, zato v funkciji *HitraPrekinitev* program izvaja le operacije, ki zahtevajo hitro in stabilno servisiranje. Te operacije zajemajo krmiljenje nekaterih zunanjih naprav preko digitalnih izhodov vzporednih vrat. Števec v funkciji služi kot časovna enota. Zaradi omejitve maksimalne vrednosti, števec povečujemo po modulu nekega velikega števila.

Prvi signal je tako imenovani časovni čuvaj ali časovna straža (ang. *watchdog*). Stanje signala se spremeni ob vsakem obhodu funkcije. Za krmiljenje ostalih signalov uporabljamo spremenljivko *sprožilec*, to je števec po modulu 12. Vrednosti *sprožilca* od 0 do 11 predstavljajo en cikel proženja kamer in luči, kot je prikazano na sliki 11. Osnovno pravilo proženja kamer je, da ju prožimo izmenično in s tem izločimo medsebojne motnje zaradi osvetlitev. Pri proženju kamere je signal dvignjen eno periodo, pri proženju luči pa tri periode. Signal za proženje luči se dvigne eno periodo pred dvigom signala za proženje kamere in spusti eno periodo za njim. To omogoča optimalno in stabilno osvetlitev v času zajema slike. Poleg omenjenih signalov sta v funkciji krmiljena tudi signala za zračna ventila, ki ločujeta dobre in slabe kose.

Frekvenci delovanja časovnega čuvaja in kamere:

$$\text{frekvenca časovnega čuvaja } = \frac{\text{frekvenca hitrega časovnika}}{2} = 500 \text{ Hz}$$

$$\text{frekvenca kamere } = \frac{\text{frekvenca hitrega časovnika}}{12} = 83{,}3 \text{ Hz}$$



Slika 11: Časovni diagram enega cikla *sprožilca*

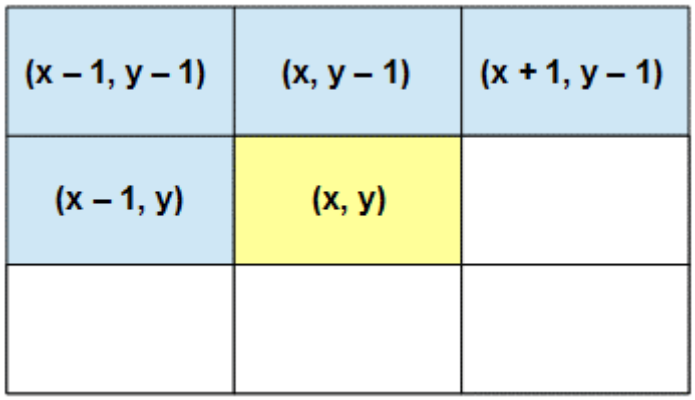
4.1.3.2. Osnovne prekinitve

Osnovne prekinitve so prekinitve z nizko prioriteto, frekvenca generiranja prekinitvenih zahtev je okoli 20 Hz. Ob prekinitvi se izvede funkcija *OsnovnaPrekinitev*, ki ni časovno omejena, zato frekvenca samih zahtev nekoliko niha. V funkciji *OsnovnaPrekinitev* se krmilijo digitalni vhodi in izhodi, ki niso obdelani v okviru hitrih prekinitev, izpisujejo se določeni podatki o stanju naprave in nekateri števci, aktivno se preverja stanje naprave, zapisujejo pa se tudi podatki v datoteke, kadar je to med delovanjem naprave potrebno.

4.1.4. Iskanje povezanih komponent

Za iskanje povezanih komponent (ang. connected-component labeling), smo nekoliko spremenili osnovni algoritem [6], ki predvideva povezanost piksla z osmih strani (slika 12) in dvema obhodoma slike. Osnovni algoritem za shranjevanje povezav med ploskvami, združevanje in iskanje uporablja drevesno strukturo. Pri implementaciji smo ugotovili, da je algoritem veliko hitrejši, če drevesno strukturo nadomestimo s tabelami. Iskanje ploskev izvajamo na binarni sliki, prag je kar polovica maksimalne intenzitete sivinske slike – 128. Spremenljivki *x* in *y* na sliki 12 predstavljata koordinati trenutnega piksla na binarizirani sliki.

Zaradi omejenega časa, ki ga imamo na voljo za izvedbo iskanja komponent, algoritem izvajamo na štirikrat pomanjšani sliki. Najdena središča ploskev in okvirje nato priredimo sliki osnovne velikosti. S tem postopkom izgubimo določeno mero natančnosti, vendar to ne vpliva veliko na sam proces sledenja kosov.



Slika 12: Povezanost slikovne točke z 8 strani

4.1.4.1. Funkcija za iskanje sosednjih ploskev slikovne točke

Funkcija razišče okolico slikovne točke, na kateri se nahajamo. Poišče slikovne točke, ki že pripadajo neki ploskvi, prebere kateri ploskvi pripadajo in prešteje, koliko je takšnih sosednjih slikovnih točk.

Ob klicu funkcije, prikazane na sliki 13, podamo koordinati slikovne točke, v kateri se nahajamo, na koncu pa vrnemo kazalec na tabelo *sosedje*, ki ima 5 elementov. Elemente tabele *sosedje* in spremenljivko *števec* na začetku inicializiramo na vrednost 0, v spremenljivke *a*, *b*, *c*, *d* pa zapišemo številke ploskev sosednjih slikovnih točk, ki so na sliki 12 označene z modro barvo. Če slikovna točka ne pripada nobeni ploskvi, je njena vrednost v tabeli *aktivnePloskve* enaka 0. V prvi element tabele *sosedje* shranimo število različnih ploskev v okolici slikovne točke, v ostale elemente pa številke teh ploskev.

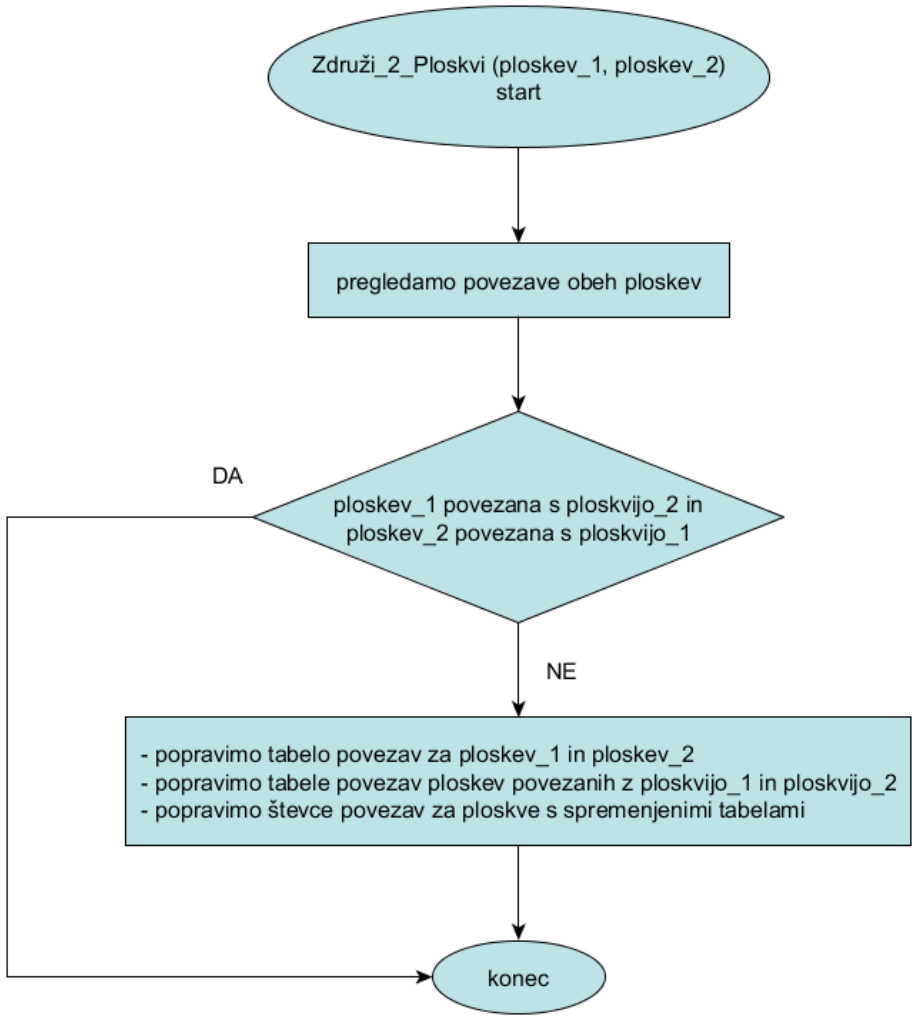
```
int *SosednjePloskve (x, y)
        sosedje [5] = {0, 0, 0, 0, 0}
        število = 0
        a = aktivnePloskve[x-1][y]
        b = aktivnePloskve[x-1][y-1]
        c = aktivnePloskve[x][y-1]
        d = aktivnePloskve[x+1][y+1]
        IF (a > 0)
                sosedje[število + 1] = a,  število ++
        ENDIF
        IF (b > 0) and (b != a)
                sosedje[število + 1] = b,  število ++
        ENDIF
        IF (c > 0) and (c != a) and (c != a)
                sosedje[število + 1] = c,  število ++
        ENDIF
        IF (d > 0) and (d != a) and (d != b) and (d != c)
                sosedje[število + 1] = d,  število ++
        ENDIF
        sosedje[0] = število
        RETURN sosedje
ENDFUNCTION
```

Slika 13: Funkcija SosednjePloskve

4.1.4.2. Funkcija za združevanje dveh ploskev

Funkcijo uporabimo pri združevanju ploskev, kadar dve sosednji slikovni točki pripadata različnima ploskvama. Ploskve, ki mejijo na sosednje ploskve zabeležimo in kasneje uporabimo pri združevanju.

Ob klicu funkcije *Združi_2_Ploskvi* na sliki 14 podamo številki dveh ploskev, ki jih želimo združiti. V tabeli *povezave* beležimo vse povezave neke ploskve z drugimi ploskvami. Če ploskvi še nista povezani, ju povežemo tako, da v tabelo povezav vpišemo vse povezave ene in druge ploskve. Popravimo tudi povezave vseh ostalih ploskev, ki so povezane z danima ploskvama.



Slika 14: Funkcija Združi_2_Ploskvi

4.1.4.3. Funkcija za iskanje povezanih komponent

S funkcijo *IskanjePovezanihKomponent* poiščemo lamele na sliki. Funkcija naredi dva obhoda po sliki, pri čemer v prvem obhodu poišče ploskve, v drugem pa ploskve pravilno združi v celoto. Rezultat funkcije je tabela v kateri so slikovne točke, ki pripadajo isti ploskvi, označene z isto številko. S pomočjo funkcije za vsako lamelo poiščemo slikovne točke, ki ji pripadajo, njeno središče in območje, kvadrat, ki jo omejuje.

Spremenljivke v funkciji:

- *vrstica*: vrstica v sliki, omejena s področjem iskanja *kvadrat*;
- *stolpec*: stolpec v sliki, omejen s področjem iskanja *kvadrat*;
- *novaPloskev*: števec ploskev;
- **sosedje*: kazalec na tabelo 5 elementov, ki hrani podatke o sosednjih ploskvah iskane slikovne točke;
- *center.Aktiven*: zastavica, ki se postavi ob najdeni ploskvi in jo upoštevamo pri sledenju kosov.

Tabele:

- *slika[stolpec][vrstica]*: sivinska slika, na kateri izvajamo iskanje ploskev z vrednostmi slikovnih točk med 0 in 255;
- *aktivnePloskve[stolpec][vrstica]*: tabela, v kateri vsaki slikovni točki določimo številko ploskve, kateri pripada. Slikovne točke, katerih vrednost je nad nastavljenim pragom in ne pripadajo nobeni ploskvi imajo vrednost 0;
- *povezave[maksimalno število ploskev][maksimalno število povezav ene ploskve]*: tabela, v kateri so shranjene vse povezave ploskve z drugimi ploskvami;
- *center*[*maksimalno število ploskev*]: tabela s središči vseh končnih, združenih ploskev,
- *številoTočk[maksimalno število ploskev]*: število točk vsake najdene ploskve.

Funkciji:

- *Poišči MinimalnoPovezavo(številka ploskve)*: v tabeli povezav za iskano ploskev poišče najmanjšo povezavo;
- *NastaviKvadratPloskve(številka ploskve)*: nastavi kvadrat, ki omejuje ploskev.

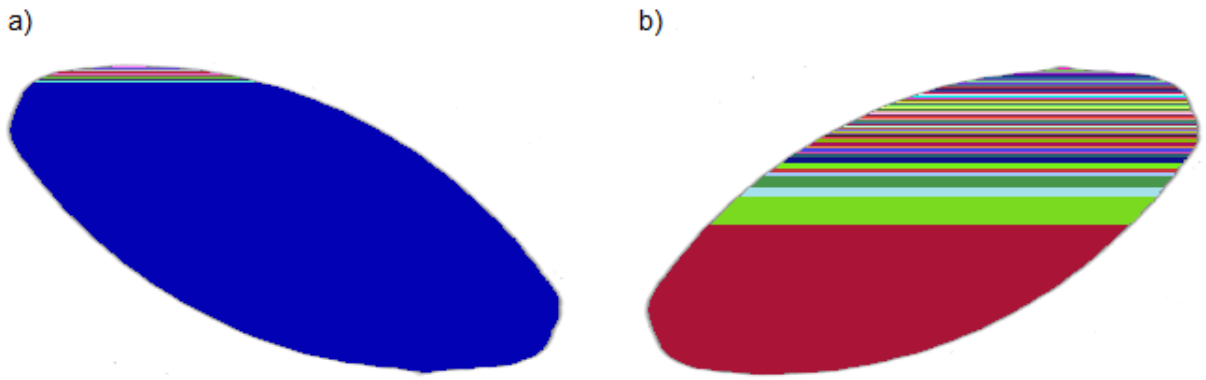
V prvem obhodu po sliki, ki ga prikazuje algoritem na sliki 17, za vsako slikovno točko, ki je nižja od nastavljenega praga, poiščemo ploskve sosednjih slikovnih točk s klicem funkcije *SosednjePloskve*. Pomikamo se z vrha navzdol in z leve proti desni. V spremenljivki *sosedje[0]* dobimo podatek o številu ploskev v okolici z različnimi številkami. V primeru, da imamo vsaj eno slikovno točko, ki že pripada neki ploskvi, dodelimo to ploskev trenutni slikovni točki. V primeru, da je sosednjih ploskev več, trenutni slikovni točki dodelimo poljubno sosednjo ploskev, hkrati pa izvedemo funkcijo za združevanje ploskev. Funkcije za združevanje se ločijo glede na število ploskev, ki jih združujemo, delujejo pa na enak način kot opisana funkcija *Združi_2_Ploskvi*. Če nobena od sosednjih slikovnih točk še ne pripada ploskvi, dodelimo trenutni slikovni točki novo ploskev in priredimo povezavo na to ploskev.

Po prvem obhodu slike pri iskanju ploskve ene lamele izgleda tabela povezav, kot je prikazano na Tabeli 2. Ker imamo na sliki eno ploskev, so vse najdene ploskve povezane med seboj.

številka ploskve	**Povezave**
1	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, …
2	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, …
3	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, …
⋮	

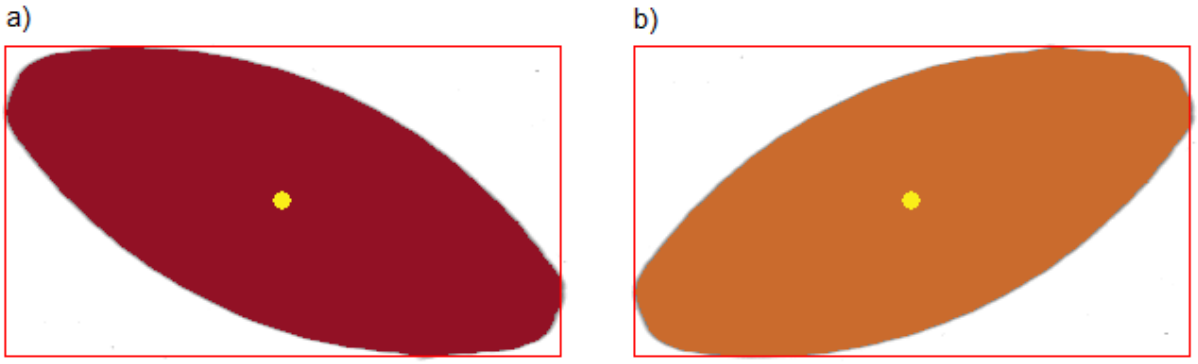
Tabela 2: Povezave po prvem obhodu slike

Zaradi narave samega algoritma je število novih ploskev odvisno od lege objekta na sliki. Sliki 15 a) in 15 b) prikazujeta najdene ploskve po prvem obhodu na istem objektu, če ga vodoravno prezrcalimo. Barve ploskev na sliki so generirane naključno.



Slika 15: Najdene ploskve po prvem obhodu

V drugem obhodu slike, ki ga prikazuje algoritem na sliki 18, vsaki slikovni točki, ki pripada neki ploskvi poiščemo povezavo z najnižjo številko ploskve in ji le-to priredimo. Preštejemo število točk, ki ploskvi pripadajo, izračunamo središča združenih ploskev ter kvadrate, ki jih omejujejo. Rezultat združevanja lahko vidimo na sliki 16 a) in b).



Slika 16: Rezultat združitev ploskev

```
IskanjePovezanihKomponent (prag, kvadrat)
*sosedje
novaPloskev = 1
FOR vrstica v sliki
        FOR stolpec v sliki
                IF slika[stolpec][vrstica] < prag
                        sosedje = SosednePloskve(stolpec, vrstica)
                        IF število sosednih ploskev > 0
                                aktivnePloskve[stolpec][ vrstica]  = sosedje[1]
                                IF število sosednih ploskev = 2
                                        Združi_2_Ploskvi (sosedje[1], sosedje[2])
                                ELSEIF število sosednih ploskev = 3
                                        Združi_3_Ploskvi (sosedje[1], sosedje[2]. sosedje[3])
                                ELSEIF število sosednih ploskev = 4
                                        Združi_4_Ploskvi (sosedje[1], sosedje[2], sosedje[3],
                                        sosedje[4])
                                ENDIF
                        ELSE
                                aktivnePloskve[stolpec][ vrstica] = novaPloskev
                                povezave[novaPloskev][0] = novaPloskev
                                novaPloskev ++
                        ENDIF
                ENDIF
        ENDFOR
ENDFOR
```

Slika 17: Funkcija *IskanjePovezanihKomponent* - iskanje ploskev

```
FOR vrstica v sliki
    FOR stolpec v sliki
        IF slika[stolpec][vrstica] < prag
            št_ploskve = aktivnePloskve[stolpec][ vrstica]
            aktivnePloskve[stolpec][ vrstica] = PoiščiMinimalnoPovezavo(št_ploskve)
            center[aktivnePloskve[stolpec][ vrstica]].x += vrstica
            center[aktivnePloskve[stolpec][ vrstica]].y += stolpec
            številoTočk[aktivnePloskve[stolpec][ vrstica]] ++
            NastaviKvadratPloskve(aktivnePloskve[stolpec][ vrstica])
        ENDIF
    ENDFOR
ENDFOR
FOR vsaka združena ploskev
    center[št_ploskve].x /= številoTočk[št_ploskve]
    center[št_ploskve].y /= številoTočk[št_ploskve]
    center.Aktiven = true
ENDFOR
ENDFUNCTION
```

Slika 18: Funkcija *IskanjePovezanihKomponent* - združevanje ploskev

4.1.5. Sledenje kosov

4.1.5.1. Razred CMerjenec

V programu smo definirali osem objektov razreda CMerjenec in jih poimenovali *Lamela*. Vedno, kadar se pojavi v vidnem polju nova lamela, ji dodelimo zaporedno ID številko po modulu osem. To je največje število objektov na sliki, ki jim lahko istočasno sledimo. V resnici se redko zgodi, da bi sledili več kot trem objektom hkrati. Na napravi imamo dve ločeni merilni mesti, zato moramo sledenje izvajati ločeno.

Slika 19 prikazuje del zaglavne datoteke (ang. *header file*) razreda CMerjenec. To je osnovni razred za sledenje in merjenje kosov. Razred vsebuje naslednje spremenljivke:

Tri zastavice, ki opisujejo stanje:

- *jeAktiven*: zastavica je dvignjena celoten čas sledenja merjenca;
- *naPoziciji*: zastavica je dvignjena, če je merjenec na položaju za merjenje in je sledenje njegovega gibanja potekalo pravilno;
- *jeIzmerjen*: zastavica je dvignjena, če je kos pravilno izmerjen.

Spremenljivki za sledenje:

- *sled[POMNILNIK_SLEDI]*: na vsaki sliki izračuna center ploskve merjenca. Gibanje merjenca poteka v smeri osi x. V krožni pomnilnik si ob vsaki sliki zabeleži x komponento centra;
- *stevecSledi*: števec merjenca se poveča ob vsaki sliki po modulu POMNILNIK_SLEDI.

Spremenljivke, ki opisujejo izmerjene vrednosti v slikovnih točkah:

- *ploscina*: ploščina ploskve;
- *dolzina*: dolžina merjenca;
- *sirina*: širina merjenca;
- *debelina*: debelina merjenca.

Spremenljivki, ki opisujeta lego:

- *kvadrat*: meja, ki obdaja merjeni objekt;
- *center*: točka, ki predstavlja težišče merjenega objekta.

Konstanti:

- *POMNILNIK_MERJENCEV*: določa število objektov razreda CMerjenec;
- *POMNILNIK_SLEDI*: velikost krožnega pomnilnika za beleženje sledi merjenca.

```
#define      POMNILNIK_MERJENCEV       8
#define      POMNILNIK_SLEDI           32

class CMerjenec: public CWnd
{
      DECLARE_DYNAMIC(CMerjenec)
public:
      CMerjenec();
      virtual ~CMerjenec();
public:
      bool         jeAktiven;
      bool         naPoziciji;
      bool         jeIzmerjen;
      int          sled[POMNILNIK_SLEDI];
      int          stevecSledi;
      int          ploscina;
      float        dolzina;
      float        sirina;
      float        debelina;
      CRect        kvadrat;
      CPointFloat  center;
```

Slika 19: Spremenljivke razreda CMerjenec

4.1.5.2. Algoritem za sledenje kosov

Algoritem za sledenje kosov nam omogoča sledenje lameli ves čas, ko je ta prisotna v vidnem polju kamere. V primeru več lamel na sliki je pomembno, da lamel ne zamenjamo. Z neprestanim sledenjem lahko določimo njeno hitrost, jo na ustreznem mestu izmerimo in določimo zakasnitev ventila, ki ločuje dobre in slabe lamele.

Ob klicu funkcije *SledenjeKosov*, prikazane na sliki 20 in sliki 21, podamo dva argumenta: številko kamere, s katere je bila slika posneta, in število predhodno najdenih ploskev na sliki. V zanki nato vsem aktivnim objektom *Lamela* povečamo interni števec sledi in preštejemo skupno število aktivnih objektov *Lamela* ob prihodu v funkcijo, se pravi končno stanje prejšnjega klica funkcije.

V drugem segmentu funkcije povežemo ploskve na novi sliki z obstoječimi aktivnimi objekti *Lamela*. Ker poznamo približno hitrost lamele in prejšnjo pozicijo, lahko v grobem ocenimo, na kateri poziciji se nahaja objekt na novi sliki.

V tretjem segmentu funkcije preverimo, ali je število najdenih ploskev enako številu aktivnih objektov *Lamela*. V kolikor je število ploskev večje, pogledamo kje se ploskve nahajajo, in, v kolikor so ploskve na začetku slike, jih dodelimo novim objektom. Če je število ploskev manjše od števila aktivnih objektov, aktivni objekt brez dodeljene ploskve pobrišemo. Po tem koraku funkcije imamo pravilno stanje aktivnih objektov *Lamela*, glede na najdeno število ploskev na zadnji sliki.

V četrtem segmentu pogledamo, ali je kateri od objektov *Lamela* na poziciji za merjenje. Če je, izračunamo razdaljo od prejšnje pozicije ter čas, ki je med tem potekel. Iz podatkov izračunamo hitrost potovanja, preostanek poti do pozicije ventila ter časovno zakasnitev do trenutka pihanja ventila. Prištejemo še izmerjeni čas od trenutka proženja kamere, do prihoda slike v pomnilnik, ki znaša približno 11,0–11,3 ms.

Spremenljivki s*prožilecTrenutneSlike* ter *zadnjiSprožilec* se nanašata na kamero in se ažurirata ob proženju.

```
FUNCTION SledenjeKosov(številkaKamere, številoNajdenihPloskev)
FOR vsaka Lamela v tabeli merjencev
    IF Lamela.jeAktiven = true THEN
        številoAktivnihLamel ++
        Lamela.številecSledi povečamo za 1 (po modulu POMNILNIK_SLEDI)
    ENDIF
ENDFOR
FOR vsaka najdena ploskev
    FOR vsaka Lamela v tabeli merjencev
        IF (Lamela.jeAktiven = true) and (predhodna sled Lamele obstaja) THEN
            IF ploskev izpolnjuje kriterije za dodelitev THEN
                Lamela.sled[Lamela.številecSledi] = centerPloskve.x
                Lamela.center = centerPloskve
                ploskevAktivna = false
            ENDIF
        ENDIF
    ENDFOR
ENDFOR
```

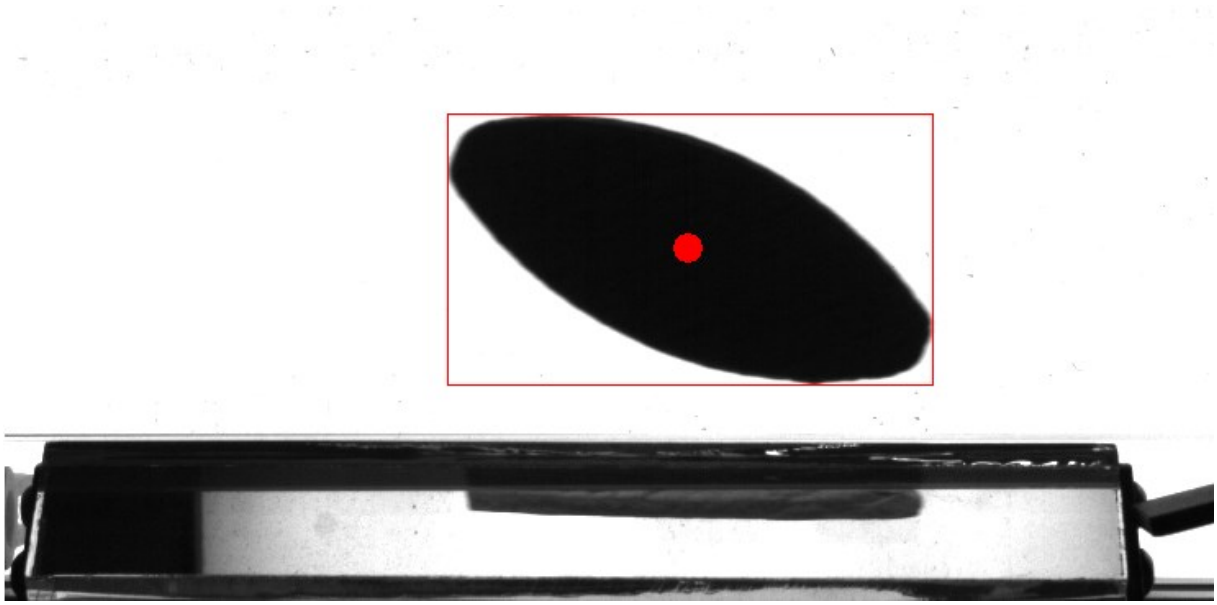
Slika 20: Funkcija *SledenjeKosov* 1/2

```
IF številoAktivnihLamel < številoNajdenihPloskev THEN
    WHILE številoAktivnihLamel < številoNajdenihPloskev
        IF (ploskevAktivna = true) and (centerPloskve je na zacetku slike) and
         (Lamela[IDštevec].jeAktiven = false) THEN
            Lamela[IDštevec].jeAktiven = true
            Lamela[IDštevec].sled[0]= centerPloskve.x;  Lamela.center = centerPloskve
            ploskevAktivna = false
            številoAktivnihLamel++
            ++ IDštevec po modulu (POMNILNIK_MERJENCEV)
        ENDIF
    ENDWHILE
ELSE IF številoAktivnihLamel > številoNajdenihPloskev THEN
    WHILE številoAktivnihLamel > številoNajdenihPloskev
        IF (Lamela.jeAktiven = true) and (sled ni bila dodeljena) THEN
            Lamela.Reset()
            številoAktivnihLamel --
        ENDIF
    ENDWHILE
ENDIF
FOR vsaka Lamela v tabeli merjencev
    IF (Lamela.jeAktiven = true) and (Lamela.sled na poziciji za merjenje)
        časMedProžilcama = prožilecTrnutneSlike - zadnjiProžilec
        razdalja = Lamela.sled[številecSledi] - Lamela.sled[številecSledi - 1]
        IF (razdalja > 0) and (časMedProžilcama > 0)
            hitrost = razdalja / časMedProžilcama
            pot = pozicijaVentila – Lamela.sled[številecSledi]
            zakasnitevVentila = (pot / hitrost) – (trenutniČas - sprožilecTrnutneSlike)
            Lamela.naPoziciji = true
        ELSE
            Lamela.Reset()
        ENDIF
    ENDIF
    zadnjiSprožilec = sprožilecTrnutneSlike
ENDFOR
```

Slika 21: Funkcija *SledenjeKosov* 2/2

4.1.6. Meritve

Meritve izvedemo, ko je center lamele na ustrezni lokaciji (dvignjena zastavica *Lamela.naPoziciji*), kot prikazuje slika 22. Potovanje lamel poteka od leve proti desni. Lokacija za merjenje se nahaja v zadnji tretjini vidnega polja kamere, kjer je nameščeno tudi stransko ogledalo za kontrolo debeline.



Slika 22: Lamela na poziciji za merjenje

4.1.6.1. Ploščina – algoritem polnjenja regij

Sledenje kosov izvajamo na štirikrat pomanjšani sliki, kar povzroči izgubo natančnosti meritev, zato za natančnejši izračun ploščine uporabimo poplavni algoritem (ang. *flood fill*) [7]. Ker ob začetku izvajanja meritev že imamo podan center lamele in kvadrat, ki lamelo omejuje, lahko meritev ploščine lamele izvedemo le znotraj danega kvadrata in s tem močno zmanjšamo čas izvajanja algoritma.

Kot vidimo na sliki 23, gre za preprost rekurzivni algoritem. Komponenti *x* in *y* sta koordinati slikovne točke na sliki, *kvadrat* je področje, ki omejuje lamelo, in *prag* vrednost med 0 in 255, ki ločuje ploskev od ozadja. V tabelo *pregledana* shranjujemo podatke o že pregledanih slikovnih točkah. Število slikovnih točk, ki sestavljajo ploskev, beležimo v spremenljivki *števec*.

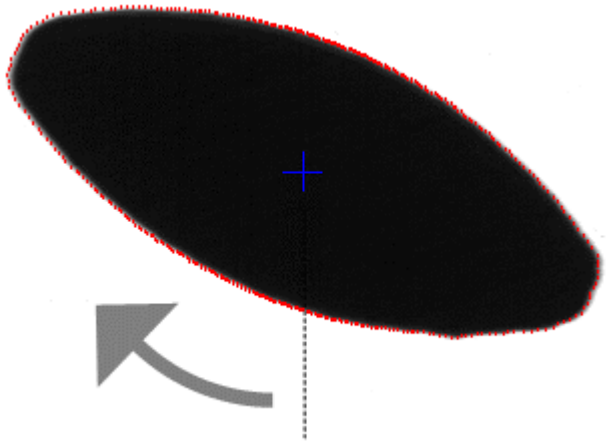
```
NapolniRegijo (komponenta x, komponenta y, kvadrat, prag)
    IF (x > kvadrat.levo) and (x < kvadrat.desno) and (y > kvadrat.gor) and (y < kvadrat.dol)
        IF (slika[x][y] < prag) and (pregledana[x][y] = false)
            števec = števec + 1
            NapolniRegijo( x – 1, y, kvadrat, prag)
            NapolniRegijo( x + 1, y, kvadrat, prag)
            NapolniRegijo( x, y - 1, kvadrat, prag)
            NapolniRegijo( x, y + 1, kvadrat, prag)
            NapolniRegijo( x – 1, y + 1, kvadrat, prag)
            NapolniRegijo( x + 1, y - 1, kvadrat, prag)
            NapolniRegijo( x + 1, y + 1, kvadrat, prag)
            NapolniRegijo( x – 1, y - 1, kvadrat, prag)
        ENDIF
    ENDIF
ENDFUNCTION
```

Slika 23: Funkcija *NapolniRegijo*

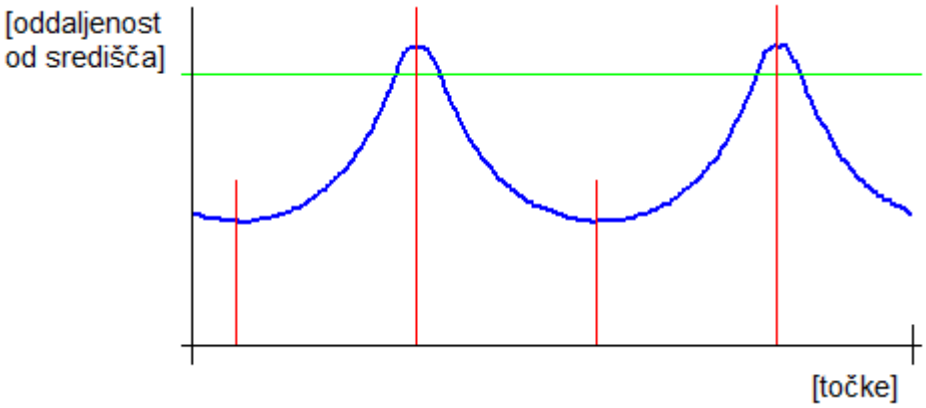
4.1.6.2. Dolžina in širina

Pri iskanju dolžine in širine lamele moramo najprej določiti orientacijo. Zaradi širine kanala, po katerem potuje, je lahko lamela po vzdolžni osi zamaknjena do trideset stopinj, odvisno seveda od tipa.

Pri določanju lege najprej poiščemo grob obris. Začnemo v središču, kjer je intenziteta nizka in potujemo proti spodnjemu robu slike. Ko zaznamo prvo slikovno točko z višjo intenziteto od našega nastavljenega praga, shranimo točko v tabelo robnih točk. Postopek ponavljamo z zamikom ene stopinje v smeri urinega kazalca. Dobimo 360 robnih točk, ki tvorijo grob obris lamele. Najdene robne točke pri opisanem postopku prikazuje slika 24.



Slika 24: Grob obris lamele



Slika 25: Graf oddaljenosti robnih točk od središča

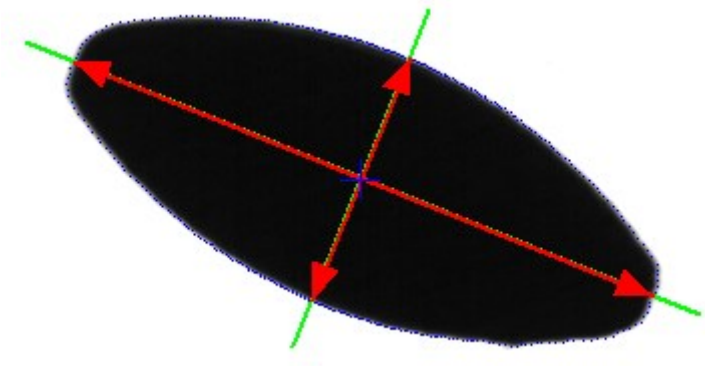
Za točke na robovih izračunamo tudi njihovo oddaljenost od središča. Kot vidimo na sliki 25, imamo na grafu oddaljenosti točk dva maksimuma. Za prag vzamemo 90 % maksimalne vrednosti, kar na sliki 25 ponazarja zelena črta, in izračunamo težišči obeh likov nad pragom. X komponenti težišč nam določita robni točki, ki tvorita vzdolžno os. Prečno os določimo z robnima točkama, zamaknjenima za 90 stopinj.

Za merjenje dolžine in širine poiščemo natančnejši prehod intenzitete v okolici vsakega od osnih robov. V obeh smereh določimo prehod na stotinko natančno z razmerjem med razliko intenzitete slikovne točke in praga ter razliko intenzitete slikovne točke in intenzitete predhodne slikovne točke ob predpostavki, da je intenziteta predhodne slikovne točke pod pragom in intenziteta trenutne slikovne točke nad pragom. Vrednosti *(x, y)* sta celoštevilski koordinati trenutne slikovne točke na sliki, *(x', y')* pa koordinati prehoda:

$$x' = \mathrm{x} - \frac{intenziteta - prag}{\text{intenziteta} - \text{intenziteta predhodne slikovne točke}}$$

$$y' = \mathrm{y} - \frac{intenziteta - prag}{\text{intenziteta} - \text{intenziteta predhodne slikovne točke}}$$

Zaradi robustnosti meritev posamezno končno točko določimo kot povprečje več robnih točk v okolici. Razdalja med vzdolžnima točkama predstavlja dolžino lamele, razdalja med prečnima pa širino, kot prikazuje tudi slika 26.



Slika 26: Prikaz meritev dolžine in širine

4.1.6.3. Napake na robovih

Za iskanje napak na robovih naredimo primerjavo oddaljenosti robnih točk od centra z referenčnim kosom in seštejemo razlike. S tem izločimo kose z manjšimi napakami, ki jih ne moremo izločiti po velikosti ploščine.

4.1.6.4. Debelina

Merjenje debeline s pomočjo ogledala ni mogoče, saj se lege kosov preveč spreminjajo. Nekateri kosi se pri hitrosti 2 m/s celo nekoliko dvignejo od podlage, zato meritve ne bi bile natančne. Debelino merimo le grobo, da s tem izločimo primere, ko ena lamela leži na drugi lameli. Poleg debeline merimo tudi ukrivljenost lamele.

4.2. Uporaba orodij in tehnologij

4.2.1.1. Microsoft Visual Studio 2013

Visual Studio je integrirano razvojno okolje (IDE) podjetja Microsoft. Namenjeno je razvoju aplikacij in programov v okolju Microsoft Windows in pa izdelavi spletnih strani, spletnih aplikacij ter spletnih storitev. Orodje ima vgrajena posamezna okolja, ki podpirajo številne programske jezike: C, C++ in C++/CLI (okolje Visual C++), VB.NET (okolje Visual Basic .NET), C# (Visual C#) in F#. S pomočjo dodatnih paketov nudi podporo tudi za jezike M, Python in Ruby ter okolja XML/XSPLT, HTML/XHTML, JavaScript in CSS. [8]

Pri diplomskem delu sem uporabljal okolje Visual C++. Okolje nudi razna orodja za razvijanje in razhroščevanje C++ kode, ki v veliki meri olajšajo delo programerja, saj omogočajo preglednost kode in hitro odkrivanje napak. Razhroščevalnik (ang. *debugger*) nudi možnost izvajanja kode po korakih, spremljanje vsebine na pomnilniških lokacijah, orodja za razvoj sodobnih uporabniških grafičnih vmesnikov (GUI) ter številne druge funkcije. S pomočjo čarovnikov (ang. *wizard*) in predlog (ang. *template*) omogoča hitro in enostavno razvijanje kode tudi manj izkušenim uporabnikom. [9][10]

4.2.1.2. IC imaging control 3.3

IC Imaging Control je družina komponent za zajemanje slik z industrijskih videokamer podjetja The Imaging Source. Razvijalcem aplikacij omogoča enostaven programski dostop do The Imaging Source WDM virov videovsebine. Paket vsebuje IC Imaging Control .NET komponente za C#, VB.NET in C++ knjižnico. Uporaba C++ knjižnice v okolju Visual C++ je dokaj preprosta, saj paket vsebuje veliko Visual C++ projektov in primerov uporabe knjižnice. [11]

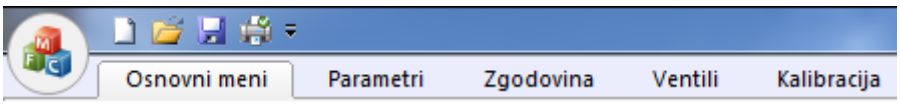
4.2.1.3. TVicHW32 6.0

TVicHW32 je splošnonamenski gonilnik, ki ga lahko uporabljamo v katerem koli programskem jeziku. Gonilnik dovoljuje dostopanje do strojnih naprav in upravljanje z le-temi neposredno iz okolja Microsoft Windows brez uporabe Windows DDK. Vsebuje funkcije za uporabo vhodno-izhodnih vrat (ang. *I/O ports*), dostopanje do fizičnega pomnilnika, obdelavo strojnih zahtev po prekinitvah (ang. *IRQ handling*) in operacij DMA. [12]

4.3. Prikaz uporabe programa

Grafični uporabniški vmesnik uporabniku naprave omogoča spremljanje procesov, ki se na napravi izvajajo, pregled nad stanjem naprave, spreminjanje nekaterih nastavitev in spremljanje statističnih grafov ter slikovnih prikazov.

Glavni meni programa je razdeljen na pet področij, kot je prikazano na sliki 27: Osnovni meni, Parametri, Zgodovina, Ventili in Kalibracija.



Slika 27: Glavni meni programa

4.3.1. Osnovni meni

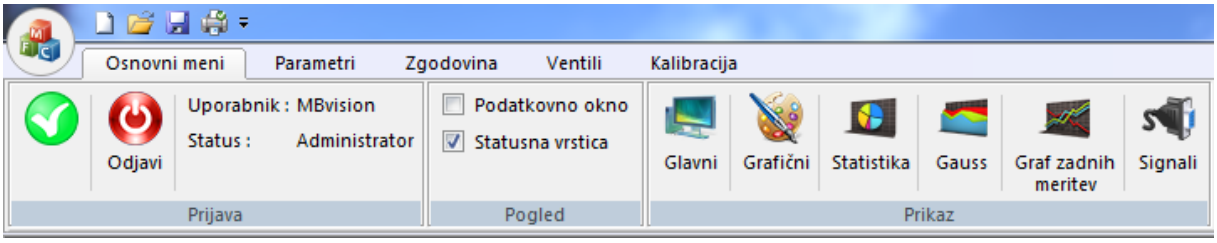
Osnovni meni, ki ga prikazujeta sliki 28 in 29, vsebuje šest podpodročij: Prijava, Pogled, Prikaz, Kamera, Lastnosti kamere in Stikala.

Prijava v sistem je namenjena omejevanju spreminjanja nastavitev sistema. Prilagodi se nabor nastavitev, ki so zaklenjene, uporabniška imena in prioritete po želji uporabnika oziroma kupca sistema.

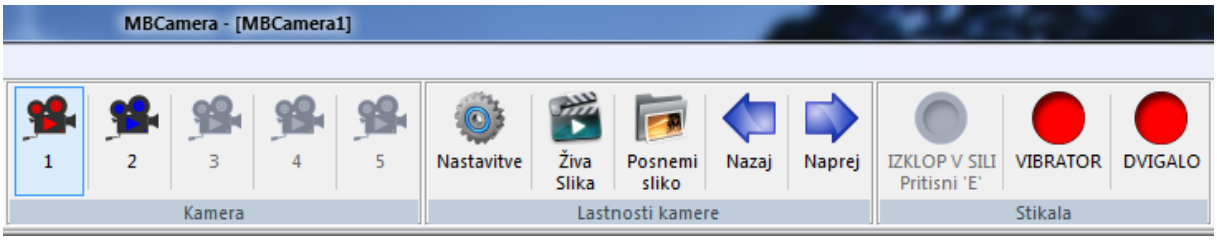
V podpodročju Prikaz uporabnik lahko izbira med šestimi različnimi grafičnimi prikazi.

Podpodročje Kamera je namenjeno izbiri kamere, v podpodročju Lastnosti kamere pa lahko letej spreminjamo določene nastavitve, shranjujemo slike, pogledamo sliko kamere v živo in pregledujemo shranjene slike.

Podpodročje Stikala prikazuje stanje rotacijskega vibracijskega dodajalnika, stanje dvigala zalogovnika in stikala za izklop v sili.



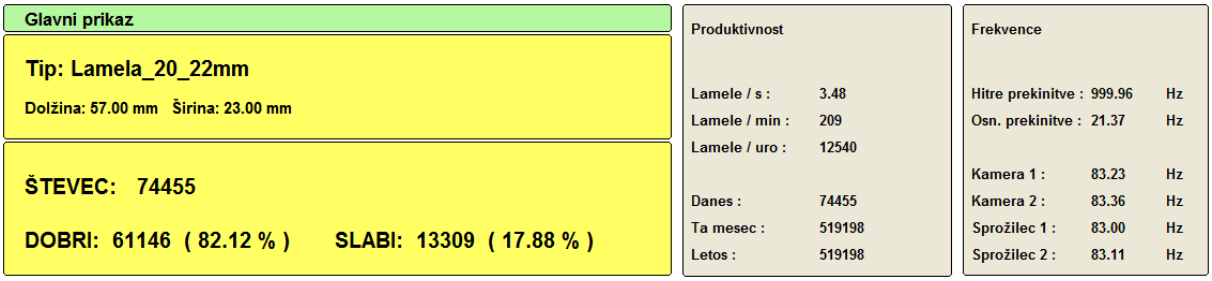
Slika 28: Osnovni meni 1/2



Slika 29: Osnovni meni 2/2

4.3.1.1. Glavni prikaz

Glavni prikaz (slika 30) vsebuje prikaz števcev dobrih in slabih kosov, izpis trenutno nastavljenega tipa lamele, nekaj podatkov za spremljanje produktivnosti in nekaterih frekvenc.

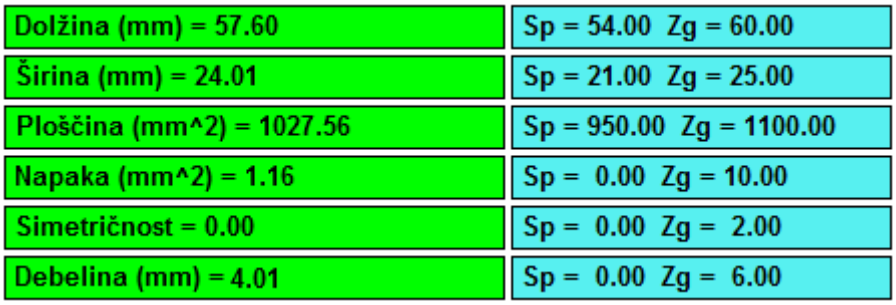


Slika 30: Glavni prikaz 1/3

Za vsako kamero so prikazani nekateri števci procesa za spremljanje pravilnega izvajanja, časovne meritve nekaterih funkcij in informacije o sledenju zadnjega izmerjenega kosa, kar lahko vidimo na sliki 31.

Slika 31: Glavni prikaz 2/3

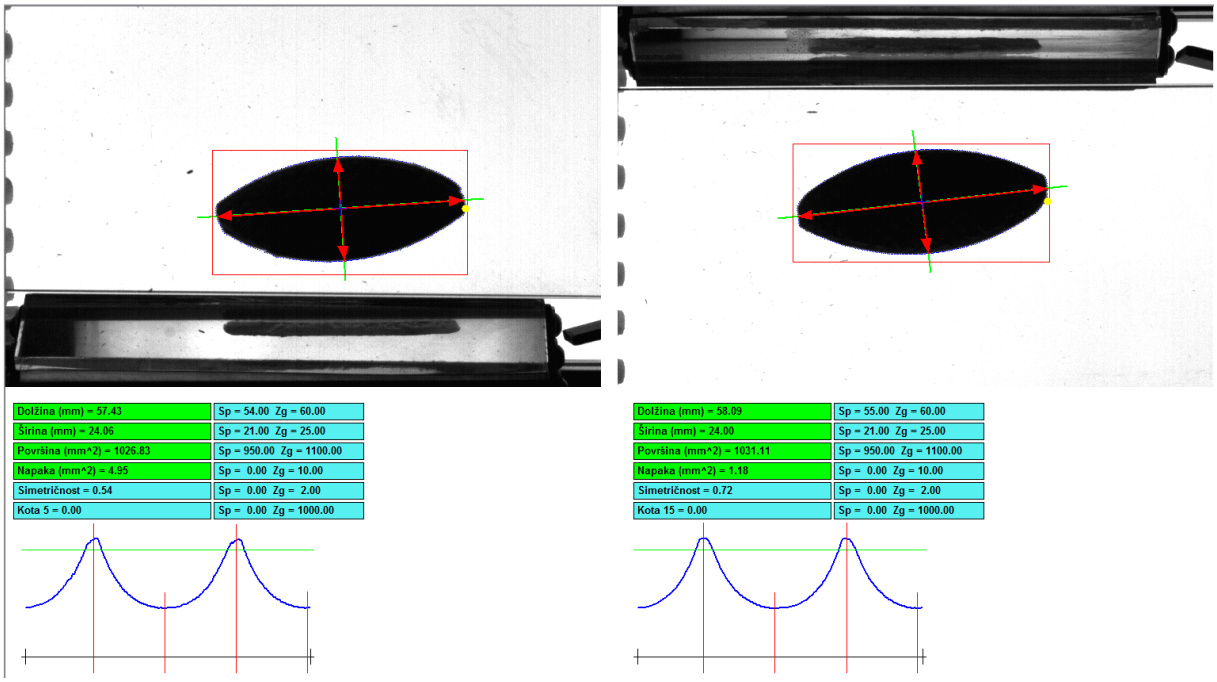
Prikaz rezultatov zadnje meritve za kamero 1 prikazuje slika 32. Za vsako merjeno vrednost je izpisano ime, izmerjena vrednost in nastavljeno tolerančno področje, tj. spodnja in zgornja meja. V primeru, da je izmerjena vrednost znotraj tolerančnega področja, je pravokotnik obarvan zeleno, v nasprotnem primeru pa rdeče.



Slika 32: Glavni prikaz 3/3

4.3.1.2. Grafični prikaz

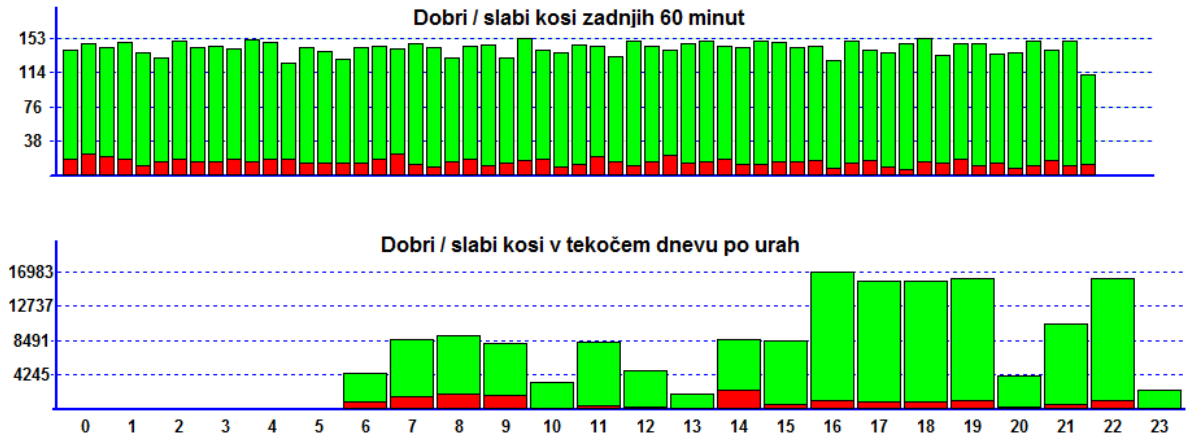
Grafični prikaz na sliki 33 prikazuje sliko zadnje izmerjene lamele na kameri 1 (levo) in kameri 2 (desno) in nekatere oznake ter grafe, ki smo jih spoznali v prejšnjih poglavjih.



Slika 33: Grafični prikaz

4.3.1.3. Prikaz statistike

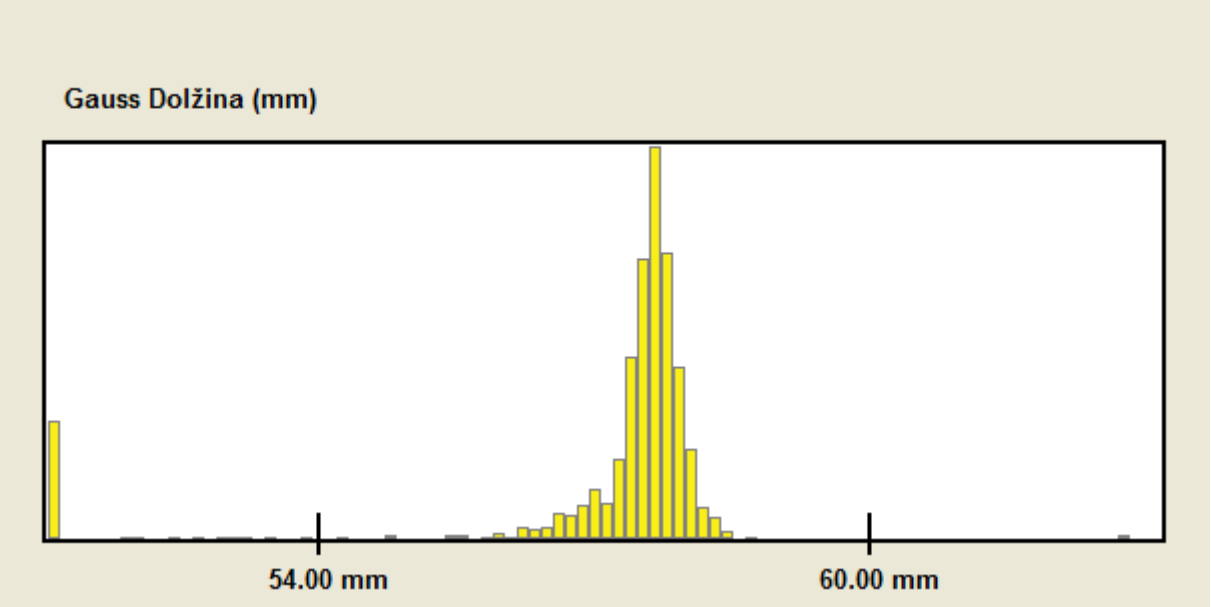
V prikazu statistike sta na voljo dva grafa meritev. Prvi graf na sliki 34 prikazuje količino dobrih in slabih kosov za vsako minuto zadnjih 60 minut. Drugi graf na sliki 34 prikazuje količino dobrih in slabih kosov za vsako uro v tekočem dnevu.



Slika 34: Prikaz grafa dobrih in slabih kosov

4.3.1.4. Prikaz Gaussovih grafov

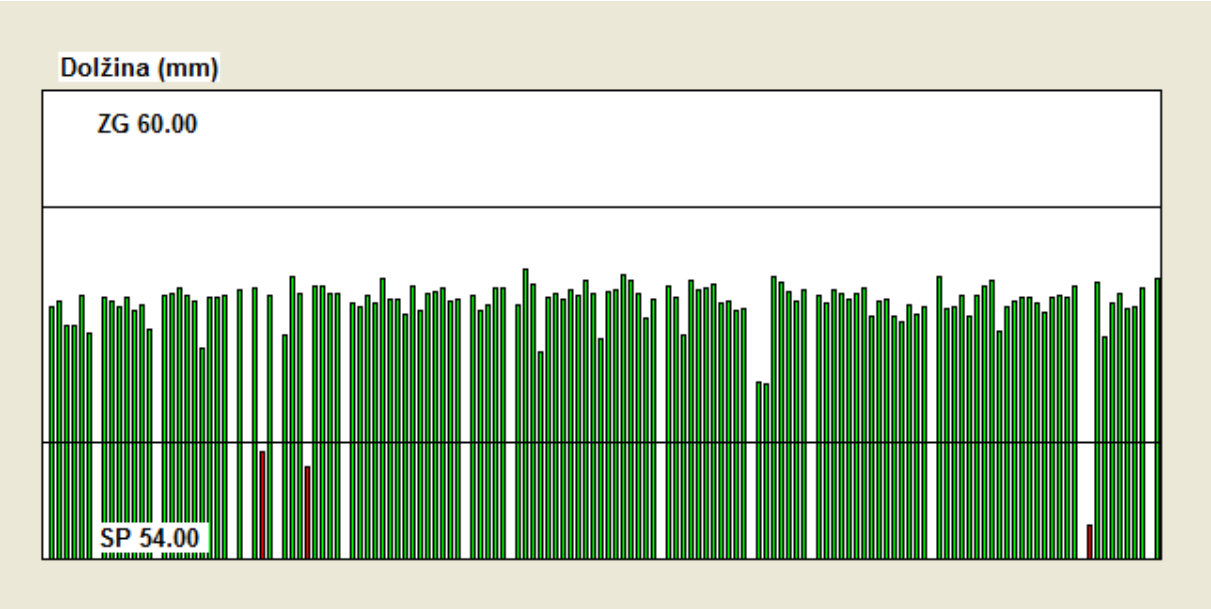
Gre za prikaz Gaussovih grafov oziroma krivulj zadnjih meritev. Število meritev, ki jih graf zajema je nastavljivo. Na grafu na sliki 35 sta označeni zgornja in spodnja tolerančna meja ter ime meritve. V prikazu so grafi za dolžino, širino in ploščino lamele.



Slika 35: Prikaz Gaussovega grafa

4.3.1.5. Prikaz zadnjih 150 meritev

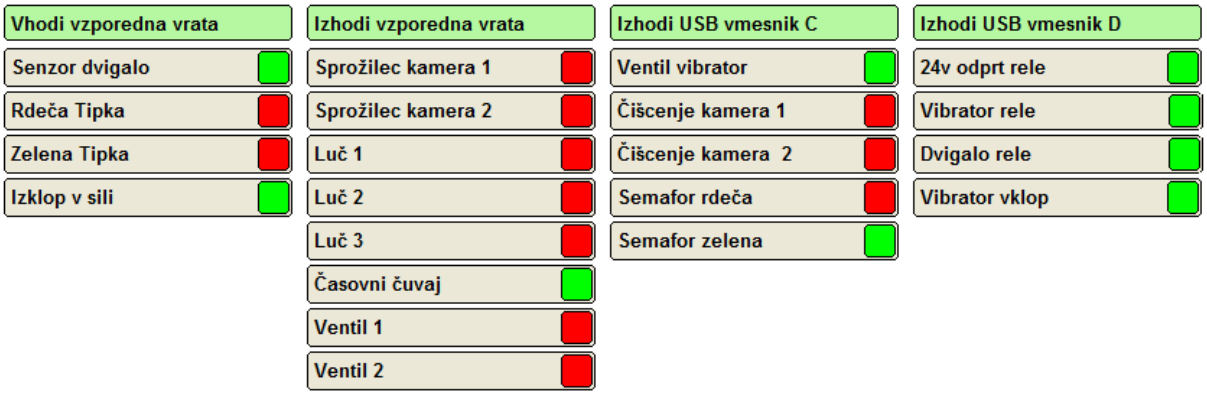
V prikazu zadnjih 150 meritev na sliki 36 lahko vidimo raztros meritev zadnjih 150 izmerjenih lamel. Na grafu sta označeni spodnja in zgornja tolerančna meja in ime merjene vrednosti. V kolikor se izmerjena vrednost nahaja v tolerančnem področju, je stolpec obarvan zeleno, v nasprotnem primeru pa rdeče. Ob vsaki novi meritvi se stolpci pomaknejo za eno mesto v levo. V kolikor je izmerjena vrednost veliko nižja od spodnje tolerance, stolpec manjka, kar vidimo v nekaterih primerih na sliki 36.



Slika 36: Prikaz zadnjih 150 meritev

4.3.1.6. Signali

Kot je vidno na sliki spodaj, prikaz signalov omogoča spremljanje stanja vhodnih in izhodnih signalov. Razdeljen je v štiri skupine: izhodi vzporednih vrat, vhodi vzporednih vrat, izhodi vmesnika C in izhodi vmesnika D modula USB-DIO-32. Dvignjeni signali so označeni z zelenim kvadratkom, spuščeni z rdečim.



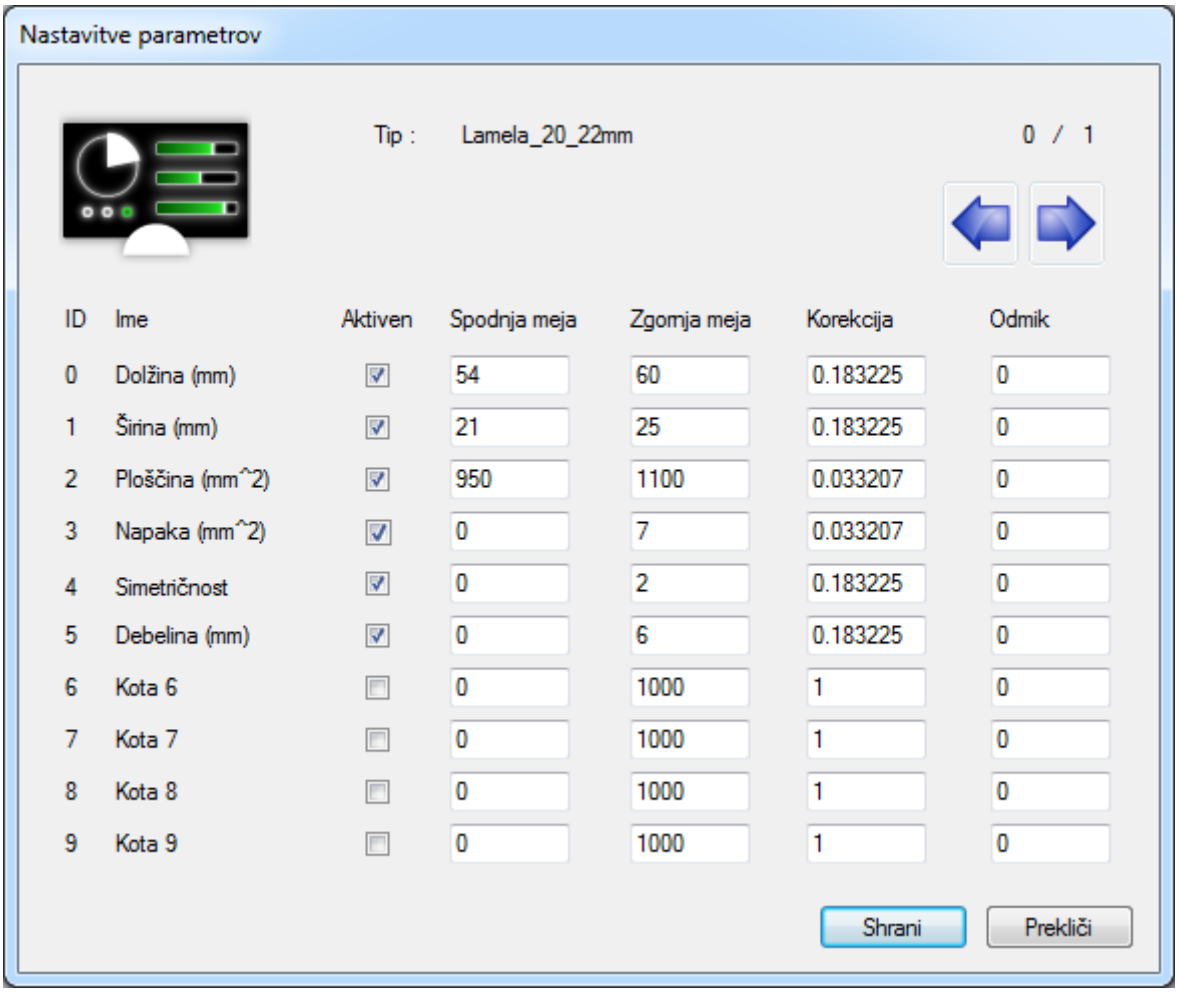
Slika 37: Prikaz signalov

4.3.2. Parametri

V področju Parametri lahko uporabnik spreminja tip lamele. Izbira lahko med tremi tipi lamel, po potrebi pa je mogoče dodati še nove.

Poleg tipa lahko v meniju, ki ga prikazuje slika 38, uporabnik nastavi tudi spremenljivke posameznih parametrov:

- Aktiven: ali se meritev upošteva pri določanju, ali je kos dober ali slab;
- Spodnja meja: spodnja tolerančna meja;
- Zgornja meja: zgornja tolerančna meja;
- Korekcija: faktor za pretvarjanje slikovnih točk v milimetre;
- Odmik: število, ki ga prištejemo izmerjeni vrednosti za potrebe korekcije.



Slika 38: Meni za nastavitev parametrov

4.3.3. Zgodovina, Ventili in Kalibracija

Področje Zgodovina služi pregledu preteklih meritev. Število shranjenih meritev je nastavljivo.

Področje Zgodovina služi pregledu preteklih meritev. Število shranjenih meritev je nastavljivo.

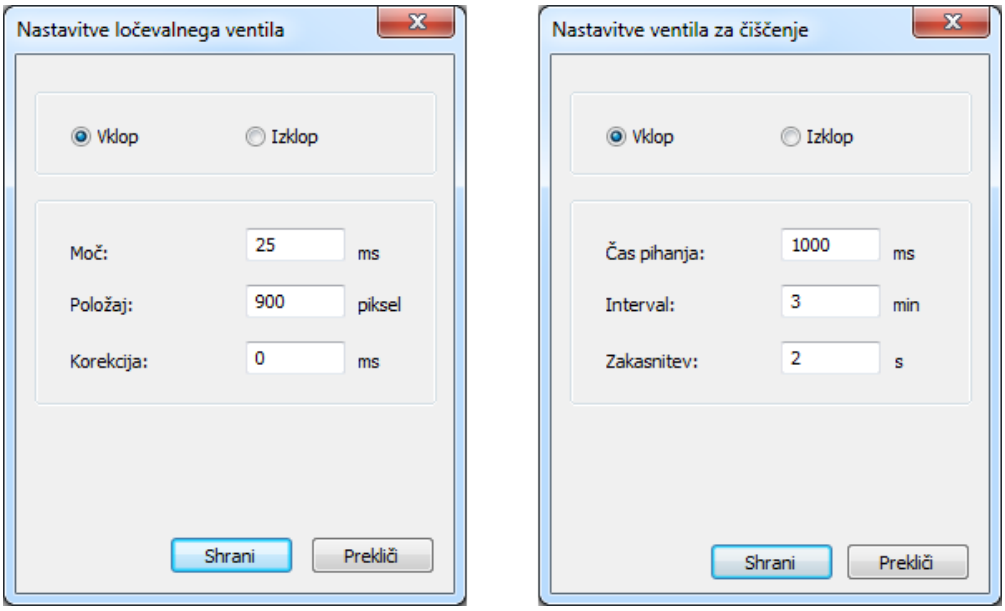
Področje Ventili je namenjeno nastavljanju ventilov za ločevanje kosov in ventilov za čiščenje. V meniju nastavitev ventilov za ločevanje (slika 39, levo) uporabnik lahko nastavlja vrednosti:

- Vklop/izklop;
- moč: čas pihanja ventila v milisekundah;
- položaj: pozicija ventila na sliki v smeri osi x izražena v slikovnih točkah;
- korekcija: možnost nastavljanja časovnega zamika pihanja ventila.

V meniju nastavitev ventilov za čiščenje (slika 39, desno) pa uporabnik lahko nastavlja vrednosti:

- Vklop/izklop;
- čas pihanja: čas pihanja čistilnih ventilov v milisekundah;
- interval: časovni interval vklopa ventilov za čiščenje;
- zakasnitev: čas med zaustavitvijo naprave in pričetkom čiščenja, potreben zato, da se drča sprazni.

Področje kalibracija omogoča uporabniku shranitev referenčne lamele za primerjanje pri meritvah.



Slika 39: Nastavitve ločevalnih in čistilnih ventilov

5. Sklepne ugotovitve

Razvili smo novo napravo za dimenzijsko kontrolo lesenih veznih lamel. Glavni cilj je bil izboljšati zanesljivost in zmogljivost obstoječe naprave, kar nam je tudi uspelo. Prva pomanjkljivost stare naprave je način ločevanja dobrih in slabih lamel. Te naključno razporejene potujejo po širokem vodoravnem prozornem tekočem traku, medtem pa zračni ventili slabe lamele izpihujejo s traku. Slaba stran takšnega načina ločevanja je, da zračni ventili ob slabi lameli velikokrat izpihnejo tudi sosednjo lamelo, ki pa je dobra. Hkrati pa v primeru, da zračni ventili slabe lamele ne odstranijo, ta po tekočem traku potuje naprej v zabojnik z dobrimi. To težavo smo odpravili s spremenjenim načinom ločevanja. Pri novi napravi lamele zaporedno potujejo po poševni drči do ventila za ločevanje. Mehanska omejitev onemogoča, da bi po drči vzporedno potovalo več lamel hkrati. Na mestu za ločevanje ventil v poseben kanal izpihuje dobre lamele, slabe pa nadaljujejo pot v zabojnik za slabe lamele. Ta način ločevanja se je izkazal kot zanesljivejši, saj v primeru kakršnekoli napake, dobra lamela potuje v zabojnik s slabimi in nikoli obratno. Pri 14-dnevnem testiranju naprave se je pokazalo, da je takih napak 1–2 ‰, kar je relativno malo. Poleg načina ločevanja smo spremenili tudi postopke merjenja. Stara naprava je lamele ločevala izključno po velikosti ploščine. Z novo napravo lamelam merimo tudi dolžino, širino, napake na robovih, simetričnost dolžine in širine ter, s pomočjo stranskih ogledal, ukrivljenost lamele. Z dodanimi meritvami smo zagotovili zanesljivejšo kontrolo in izločanje nekaterih slabih lamel, ki sicer po ploščini ustrezajo kriterijem.

Poleg zanesljivosti je bil cilj povečati tudi zmogljivost. Zmogljivost stare naprave znaša 6.000–7.000 lamel na uro. Lamele so skoznjo potovale naključno razporejene na tekočem traku s hitrostjo 0,2–0,3 m/s. Zaradi zamenjave tekočega traku z dvema drčama, se je v veliki meri spremenil način sledenja in merjenja, saj lamele po drči potujejo zaporedno s hitrostjo med 2 in 2,5 m/s. Sprememba hitrosti pa je zahtevala veliko višjo frekvenco zajema slik in hitrejše obdelave, kar je na začetku predstavljalo kar velik problem. Končna zmogljivost naprave se glede na tip lamel nekoliko razlikuje. Za lamele tipa P 20, za katere je zmogljivost najmanjša, ta znaša približno 16.000–16.500 lamel na uro, vendar je za optimalno delovanje naprave priporočljiva nekoliko manjša hitrost in zmogljivost, in sicer okrog 14.000–15.000 lamel na uro.

Glavna cilja pri izdelavi programske opreme sta bila zagotoviti zanesljivost delovanja in preprost, uporabniku prijazen grafični uporabniški vmesnik s prikazi za nadziranje procesa in nekaterimi grafičnimi prikazi rezultatov meritev in statistike.

Največjo negotovost je predstavljalo vprašanje, kako bo naprava delovala v prašnem okolju, ki se mu je v lesni industriji nemogoče izogniti; kako bosta prah in umazanija kljub sistemu čiščenja vplivala na kakovost meritev in delovanje celotnega procesa. Naprava je uspešno prestala 14 dnevni test delovanja v industrijskem okolju in izpolnila vsa naša pričakovanja in pričakovanja uporabnika.

6. Literatura

[1] Slika spoja z uporabo lesenih lamel (marec 2015). Dostopno na: http://www.woodworkbasics.com/biscuit-joint.html , 2015

[2] Opis produkta lesena vezna lamela (marec 2015). Dostopno na: http://www.wisegeek.com/what-is-a-biscuit-joint.htm

[3] Opis produkta lesena vezna lamela (marec 2015). Dostopno na: http://www.dcsys.com.pl/ed_oferta.php?text=4

[4].Tehnični podatki kamere The Imaging Source DMK 23GV024 (marec 2015). Dostopno na: http://www.theimagingsource.com/en_US/products/cameras/gige-cmos-ccd-mono/dmk23gv024/

[6] L. G. Shapiro, G. Stockman: Computer Vision, Upper Saddle River, New Jersey: Prentice Hall, 2001.

[5] Opis produkta USB-DIO-32 (marec 2015).: Dostopno na: http://accesio.com/go.cgi?p=../usb/usb-dio-32.html

[7] Algoritem polnjenja regij Wikipedija (marec 2015).: Dostopno na: http://en.wikipedia.org/wiki/Flood_fill

[8] Microsoft Visual Studio Wikipedija (marec 2015).. Dostopno na: http://en.wikipedia.org/wiki/Microsoft_Visual_Studio

[9] Microsoft Visual C++ Wikipedija (marec 2015).. Dostopno na: http://en.wikipedia.org/wiki/Visual_C%2B%2B

[10] I. Horton: Ivor Horton's Beginning Visual C++, Indianapolis: John Wiley & Sons, Inc., 2014.

[11] IC Imaging Control .NET Component for C#, VB.NET, C++ Class Library for C++ projects (marec 2015).. Dostopno na: http://www.theimagingsource.com/en_US/support/downloads/details/icimagingcontrol/

[12] Opis produkta TVicHW32 (marec 2015).: Dostopno na: http://www.entechtaiwan.com/dev/hw32/index.shtm