UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Jernej Bodlaj

PRENOSNI, OČEM VARNI LIDAR Z RAČUNALNIŠKIM KRMILJENJEM IN PODPORO

Diplomska naloga
na univerzitetnem študiju

Mentor: prof. dr. Franc Solina
Somentor: prof. dr. Tomaž Pisanski

Ljubljana, 2007

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Namesto te strani **vstavite** original izdane teme diplomskega dela s podpisom mentorja in dekana ter žigom fakultete, ki ga diplomant dvigne v študentskem referatu, preden odda izdelek v vezavo!

Zahvala

Pri izvedbi diplomskega dela se v veliki meri zahvaljujem Matjažu Gregoriču, ki mi je pomagal pri izdelavi vezja in pokrova gibalnega modula naprave ter pri testiranju. Zahvala gre tudi somentorju in direktorju inštituta UP PINT v okviru katarega je bilo delo opravljeno, prof. dr. Tomažu Pisanskemu in mladima raziskovalcema Borisu Horvatu in Primožu Lukšiču za koristne nasvete. Nasvete sta mi delila tudi mentor prof. dr. Franc Solina in prof. dr. Igor Poberaj, ki je izdelal optični del naprave. Zahvaljujem se še Nataši Grlj, ki je z napravo opravila nekaj testnih meritev in mi posredovala informacije o napakah. Navsezadnje se zahvaljujem še znancem, ki so me poslušali, ko sem naglas razmišljal o težavah v zvezi s projektom in mi na svojevrtsen način ter z dobro voljo pomagali na poti do rešitev.

Moji materi, ki mi je tekom šolanja nudila trdno oporo.

Kazalo

Seznam uporabljenih kratic in simbolov

Kratica	Pomen
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
CCD	Charge-Coupled Device
DTR	Data Transfer Ready
ISR	Interrupt Service Routine
LASER	Light Amplification by Stimulated Emission of Radiation
LCD	Liquid Crystal Display
LIDAR	Light-Imaging Detection And Ranging
MCC	Math/Chem/Comp
Nd:YAG	Neodymium-doped Yttrium Aluminium Garnet; $Nd:Y_3Al_5O_{12}$
PC	Personal Computer
PLL	Phase Lock Loop
RANSAC	RANdom SAmple Consensus
RISC	Reduced Instruction Set Computer
RS-232	Recommended Standard 232
TTL	Transistor Transistor Logic
UP PINT	Univerza na Primorskem Primorski Inštitut za Naravoslovne in Tehnične vede
USB	Universal Serial Bus
V/I kanal	Vhodno / Izhodni kanal
XML	eXtensible Markup Language

Povzetek

Lidar je naprava podobna radarju. Namesto radijskih valov uporablja svetlobo. Uporabimo ga lahko za zaznavanje delcev v zraku, pretežno različnih onesnaževalcev ali za merjenje vremenskih pojavov. Lidar skupaj s podpornimi elementi tvori lidarski sistem, ki se bo pretežno uporabljal v *Luki Koper, d.d.* Obstaja namreč želja, da se tam kvantitativno ovrednoti stopnjo onesnaževanja.

- **Cilji:**

  (a) Zasnovati, programirati in izdelati vgradni modul za premikanje lidarja v dano prostorsko orientacijo. Poleg neposredne interakcije modula z uporabnikom prek vgrajene tipkovnice, naj obvezno obstajala možnost krmiljenja iz osebnega računalnika.

  (b) Razviti programsko opremo za interaktivni nadzor lidarskega sistema, za zajemanje, prikaz in shranjevanje podatkov (ter nadaljno analizo). Programska oprema naj vključuje podporo *web*-kameri, ki smo jo pritrdili na lidar ter algoritme za računalniški vid, ki bi pripomogli k avtomatizaciji postopka merjenja.

- **Metode:**

Uporaba standardnega zrcalnega teleskopa in polprevodniške fotodiode, ki skupaj sestavljata detektor. Prilagoditev komercialnega večnamenskega Nd:YAG sunkovnega laserja, da postane očem varen in uporaben kot izvor svetlobnih sunkov. Uporaba *Microchip Inc. PIC* mikrokrmilnika, programiranega v *C*. Izdelava modula za gibanje lidarja s pomočjo dveh, v podstavek teleskopa vgrajenih koračnih motorjev. Vgradnja *Hitachi* kompatibilnega prikazovalnika in enostavne tipkovnice v modul za gibanje lidarja. Za komunikacijske namene uporabljamo RS-232 protokol. Kalibracijske algoritme in algoritme za računalniški vid uporabljamo za natančno pozicioniranje. Uporabljajo se algoritmi: Harrisov detektor, opisniki lokalnih značilnih točk, iskanje korespondenc med slikami, Gaussovo filtriranje, izračun gradientov slik, indeksna tabela in algoritem RANSAC. Razvoj temelji na modernemu razvojnemu okolju *C#*, ki ga uporabimo za razvoj uporabniku prijaznega in interaktivnega grafičnega vmesnika. Za shranjevanje podatkov uporabimo bazo podatkov in za njihov prikaz nekaj na novo izdelanih grafičnih kontrol.

- **Rezultati:**

Delujoč mobilni lidarski sistem z interaktivno računalniško podporo.

- **Ključne besede:**

LIDAR, računalniški vid, kamera, mikrokontroler, kalibracija, onesnaženje zraka

Poglavje 1

Uvod

1.1 Kratek pregled po poglavjih

Diplomska naloga v grobem obsega štiri poglavja in dodatek. V tem razdelku bomo na kratko povzeli vsebino poglavij in navedli komu so poglavja v prvi meri namenjena.

Prvo poglavje je uvodno. Poleg tega pregleda bralca seznani s pojmom lidar, poda osnove načina delovanja in opredeli tipične vrste uporabe lidarja. Bralcu predstavi cilje in zaplete pri izvedbi diplomske naloge ter poda nekatere rešitve.

Poglavji 2 in 3 sestavljata jedro pričujočega besedila. Drugo poglavje na kratko poda bistvo za uporabo strojne opreme lidarskega sistema, ki smo jo dobili pred začetkom izvajanja diplomske naloge. Poda načela delovanja in napotke za uporabo senzorja sipane svetlobe in za napajalnik laserja. Pretežni del vsebuje veliko podrobnosti in je namenjen opisu in rešitvam v zvezi z realizacijo in uporabo gibalnega modula na strojnem in na programskem nivoju. Poglavje je predvsem namenjeno bralcem, ki želijo razumeti delovanje gibalnega modula oz. imajo interes uporabiti le-tega v lastni aplikaciji.

Tretje poglavje v prvi polovici poda splošna in podrobna navodila za uporabo programske opreme za rokovanje z lidarskim sistemom. Besedilo je pretežno namenjeno končnemu uporabniku in se smatra kot priročnik za uporabo. Navodila obsegajo napotke za zajemanje in shranjevanje podatkov ter opis nastavitev lidarskega sistema. Poleg splošnih navodil poglavje vsebuje informacijo, ki bo koristila morebitnemu razvijalcu programske opreme, ki bi želel v svoji aplikaciji uporabiti kak del programeske opreme. Pretežno v drugi polovici poglavja so opisani algoritmi, ki jih programska oprema uporablja. V žarišču so algoritmi računalniškega vida in na koncu poglavja še nekaj podrobnosti v zvezi s podatkovno bazo lidarskega sistema.

Zadnje, sklepno poglavje seznanja bralca z doseženimi cilji in različnimi možnostmi za izboljšave lidarskega sistema. Navedeni so plani, ki bodo v prihodnosti realizirani.

V dodatku bo bralec našel načrt strojnega dela gibalnega modula ter vzorčne primere datotek, ki se jih lahko izvozi iz programske opreme. V primeru, da bi uporabnik želel prenesti podatke iz programske opreme v drugo aplikacijo, mu bodo vzorci koristili kot dober zgled. Na koncu je vključena še psevdokoda algoritma RANSAC.

1.2 Splošno o lidarju

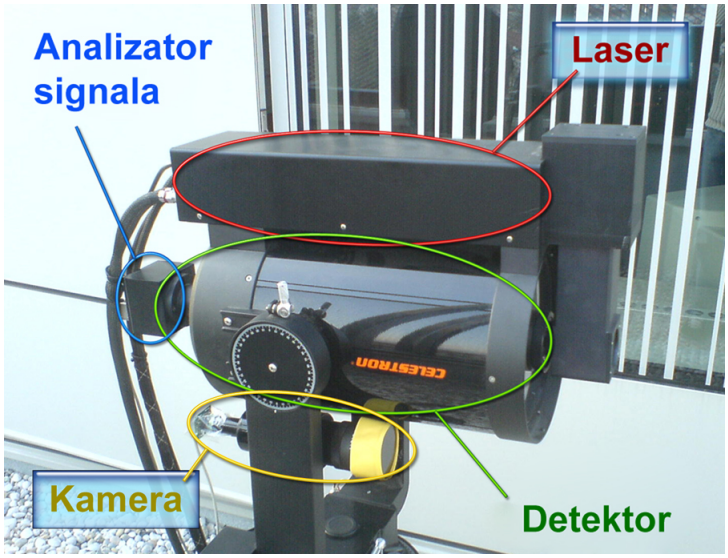
Pri lidarju gre za vrsto tehnologije optičnega zaznavanja oddaljenega cilja. Značilnosti cilja se določajo posredno preko lastnosti sipane oz. odbite svetlobe. Kot vir svetlobe se pri lidarju pretežno uporablja sunkovni laser, za razliko od sorodne naprave, radarja, ki za ta namen uporablja radijsko anteno in radijske valove. Lidar se trivialno uporablja za določanje razdalje do cilja. Po zgradbi sta lidar in radar podobna. Sprejemnik lidarja je navadno neke vrste teleskop, ki svetlobo zbere na senzor. Pri nas v ta namen uporabljamo standardni zrcalni teleskop. Tehnologije senzorjev so različne. V nekaterih primerih se uporabljajo fotopomnoževalke, v našem primeru uporabljamo fotodiodo. Sprejemnik in senzor skupaj tvorita detektor. Pri radarju vlogo detektorja prevzema kar oddajna antena. Tudi pri lidarju se detektor fizično nahaja na istem mestu kot izvor. Optično je laserski žarek usmerjen tako, da sovpada z optično osjo detektorja. Pomemben sestavni del lidarja (in radarja) je analizator signala, ki poskrbi, da se signal iz senzorja vzorči v časovne rezine in posreduje uporabniku. Navedeni trije deli so bistveni za delovanje lidarja, ostali tako ali drugače služijo podpori. Izgled naše naprave je viden na sliki 1.1, opisani deli pa na sliki 1.2.

Slika 1.1: Fotografija merilnega dela našega lidarskega sistema z gibalnim modulom in stojalom.

Poglavitna razlika med radarjem in lidarjem je, da lidar uporablja monogo krajšo valovno dolžino elektromagnetnega valovanja. Navadno uporablja ultra vijolično, vidno ali kratkovalovno infra rdečo svetlobo. Posledično z njim lahko zaznvamo aerosole in oblake ter prašne delce v zraku. Z radarjem teh pojavov ne moremo meriti, saj se radijski valovi zaradi prevelike valovne dolžine v takih medijih ne sipljejo. Lidar je zato uporaben pri raziskovanju atmosfere in za ovrednotenje vremenskih pojavov. (Za informacijo naštejmo pojave, ki so pomembni pri merjenju z lidarjem: Rayleighovo, Mieevo in Ramanovo sipanje svetlobe ter tudi fluorescenca.)

Bistvena razlika je tudi v ločljivosti naprav. Svetlobni izvor lidarja, laser tvori tanek žarek, ki obenem lahko traja zelo kratek čas. Laser zagotavlja veliko prostorsko ločljivost. Velika je tudi časovna ločljivost, saj laser lahko prožimo z veliko frekvenco.

Delovanje je sledeče. Odda se kratek svetlobni sunek časovnega reda nekaj nanosekund. Pri nas 2 ns. Ekvivalent dolžine svetlobne motnje, ki potuje skozi zrak je tako v velikostnem redu enega metra. Ko motnja zadane ob morebitni objekt ali delce se del svetlobe odbije ali siplje nazaj proti sprejemniku, ki jo zbere in usmeri na senzor. Iz časovne razlike med trenutkom oddaje svetlobnega sunka in časom sprejema odboja lahko izračunamo razdaljo. Naša naprava za vsak oddan sunek vzorči signal odbojev s frekvenco $7,37$ MHz v 256 14 bitnih vzorcev, kar omogoča detekcijo do razdalje $5,2$ km z dolžinsko ločljivostjo $20,3$ m.



Slika 1.2: Poglavitni deli merilnega dela lidarskega sistema so: izvor svetlobe – laser, sprejemnik sipane in odbite svetlobe – detektor (vključuje teleskop in senzor) ter analizator signala. Kamera je prikazana, ker ima v naši aplikaciji veliko vlogo.

Slabost lidarja je v tveganju morebitnega opazovalca za poškodbe njegovega vida. Laserski sunek s primerno intenziteto in v vidnem delu spektra, ki bi bil uporaben za praktične meritve zunaj laboratorija, je lahko ob direktem ali indirektnem vstopu v oko že usoden za opazovalčev vid. Cilj razvoja je bil razviti lidar brez te pomanjklivosti. Zato se uporablja svetloba valovne dolžine 1574 nm, ki je daleč izven vidnega dela spektra in obenem v območju, kjer je oko zelo neobčutljivo. Dodatno je žarek s posebnim periskopom

razširjen in s tem zmanjšana njegova gostota energije. Tveganje se s tem še dodatno zmanjša. Težava pri navedeni valovni dolžini je dokaj slaba ponudba primernih senzorjev. Današnja tehnologija še ne nudi senzorjev s podobnim izkoristkom, kot pri tistih v vidnem delu spektra, zato naš lidar ni tako učinkovit, kot bi lahko bil sicer.

Lidar se uporablja na večih področjih. Poleg trivialne uporabe je torej primeren za določanje koncentracij plinov, za merjenje onesnaženosti zraka z različnimi aerosoli. Uporaben je v geologiji za natančno določanje višine terena, celo skozi krošnje dreves. Uporablja se za natančno merjenje ledenikov, za merjenje različnih parametrov vetrov. Sistem lidarjev uporabljajo za milimetrsko določanje položaja lune. Lidar se uporablja za merjenje hitrosti vozil v prometu. Nenazadnje ga lahko uporabimo za zajem 3D informacije predmetov oz. za določanje reliefa površine predmetov...

V naši aplikaciji se bo lidar uporabljal za določanje onesnaženosti zraka s prašnimi delci in dimom. Mesto naj bi našel v mobilni postaji za kvantitativno določanje onesnaženosti zraka v okolici. Težava pri uporabi očem varnega lidarja je v fiksni valovni dolžini izvora svetlobe. Pri običajnem namenskem lidarju se načeloma da spreminjati valovno dolžino oddane svetlobe. Le tako je namreč možno zaznati delce različnih velikosti. Svetloba se najbolj siplje na delcih velikosti, podobne valovni dolžini svetlobe s katero jih obsevamo. V našem primeru smo torej omejeni pretežno na delce velikosti $\approx 1,6\ \mu$m.

Ob začetku izvajanja diplomske naloge nam je bil na voljo že izdelan merilni del lidarskega sistema. Ni vključeval podpore za premikanje, niti kamere. Za uporabo je bila na voljo preprosta programska oprema, ki ni omogočala praktično nobene avtomatizacije merjenja. Vključevala je neposreden prikaz rezultatov osnovnih meritev, nastavitve parametrov senzorja in sprožitev laserja prek napajalnika. Možen je bil izvoz podatkov izmerjene meritve, povprečenje podatkov zaporednih meritev zavoljo zmanjšanja razmerja signal/šum in sprotni prikaz merjenja. Po današnjih merilih uporaba ni bila ravno interaktivna. Za potrebe uporabe aplikacija ni povsem zadoščala. Vsekakor je bila odlična podlaga za izdelavo nove programske opreme. V začetnih stadijih razvoja programske opreme smo se često opirali prav na obstoječo aplikacijo, ki je v precejšni meri skrajšala uvodni del kodiranja.

1.3 Cilji diplomskega dela

Najprej omenimo, da je diplomska naloga del večjega projekta, ki vključuje izdelavo očem varnega lidarja, izdelavo krmilnega modula, razvoj programske opreme za delo z lidarskim sistemom, za analizo podatkov, namestitev celotnega sistema v mobilno postajo, postopka pridobitve patenta, opravljanje meritev in nazadnje, v sodelovanju s sorodnimi projekti, izpolnitev osnovnega cilja. Osnovni cilj dotičnih sorodnih projektov je ob njihovem medsebojnem sodelovanju dokazati, da podjetje, naročnik projekta, *Luka Koper, d.d.* onesnažuje zrak v okolici manj, kot sosednja *Luka Trst*.

Zaradi projektne narave in dokaj neraziskanega področja dela so se cilji diplomske naloge delno spreminjali tekom izvedbe.

Primarni cilj je ostal enak: izdelati solidno programsko opremo za delo z lidarskim sistemom. Izpoljnjevala naj bi zahteve za zajem podatkov oz. njihovo predelavo v uporab-

nika berljivo informacijo, za shranjevanje, prikaz in nadaljno analizo. Vključevala naj bi ogrodje za delo s kamero, ki bi jo montirali na napravo, za boljši nadzor orientacije. Poleg tega naj bi programska oprema nudila podlago za nadaljno vključevanje različnih funkcij, kot na primer krmiljenje orientacije naprave za katerega je bilo potrebno razviti strojno opremo.

Tekom razvoja je prišlo do delnih odstopanj. Opustili smo idejo o analitičnem delu programske opreme, ki so jo kasneje v roke vzeli fiziki in matematiki. Odločitev je bila pravzaprav smiselna, saj mi kot študentu računalništva in informatike manjka precej znanja s področja fizike delcev. Tudi sicer bi morali z algoritmi za analizo počakati na zaključek razvoja programske opreme za osnovne namene. Kasneje se je izkazalo, da bi iz vidika trajanja izdelave vključitev analitičnega dela precej presegla obseg ene diplomske naloge. Namesto tega smo se odločili, da izdelamo strojni modul za gibanje naprave, ki je sam zase prerastel v manjši projekt. Prvotno je bilo mišljeno, da bi ga izdelali študentje pri predmetu elektronike na Fakulteti za matematiko in fiziko. Ker bi se to žal zgodilo šele precej po začetku izvajanja diplomske naloge in bi z implementacijo delov programske opreme za krmiljenje gibalnega modula morali počakati, smo poiskali boljšo rešitev. Modul smo začeli razvijati vzporedno s programsko opremo. Sprva s pretežnim deležem dela na modulu. Ko je bil modul za gibanje končan do mere, da se je z njim dalo izvajati osnovne manevre, smo se pretežno posvetili razvoju programske opreme in modulu vzporedno dodajali funkcije ter odpravljali napake. Prvi cilj, prevesti in predelati obstoječo programsko opremo, je bil do takrat že dosežen. Porajale so se še nove zahteve. Odločili smo se, da v ozadje krmilne aplikacije, v okviru programske opreme postavimo podatkovno bazo, v katero se bodo shranjevali vsi podatki o meritvah. Odločitev se je izkazala za zelo dobro. Pojavili sta se ideji za dva tipa razširjenih meritev in sicer meritev v območju in meritev v rezini (več o tem kasneje), ki v predhodnji programski opremi zaradi odsotnosti modula za premikanje nista bili izvedljivi. Vzporedno s tema dvema je ostal neizpolnjen cilj o uporabi kamere. Za začetek je bil na glavnem oknu aplikacije realiziran le prikaz pogleda skozi kamero. Težnja po boljši ustreznosti diplomskega dela je botrovala odločitvi, da implementiramo še modul za panoramsko merjenje s podporo računalniškega vida. Pri tem smo bolj za izziv in manj za uporabnost razvili modul za avtomatsko kalibracijo kamere. Modul je bil razvit v želji po odpravi optičnih napak kamere ter za določitev njenega zornega kota. Navsezadnje bi tako ali tako morali uporabiti ali implementirati vsaj približno različico kalibracije, da bi lahko konstruirali dobro panoramsko sliko. Ideja panoramskega merjenja je bila v osnovi sledeča: najprej posnamemo panoramsko sliko ozadja in na njej označimo interesne točke merjenja. Kasneje, pri merjenju v izbranem položaju sistem, ob ne povsem točnem položaju iz slike kamere in panoramske slike v bazi, samodejno ugotovi kje točno naj opravi meritev. O netočnem položaju je govor zaradi mobilnosti naprave. Z drugimi besedami bi rekli, da v praksi napravo težko postavimo točno v enak položaj kot ob predhodnji meritvi. In bistvo: manjšo napako v postavitvi naj programska oprema odpravi oz. kompenzira samodejno.

Na tem mestu dodajmo manjšo opombo; pojem prenosnosti bi lahko uporabili tudi v primeru, da bi napravo vsakič nameščali v naprej pripravljena podnožja, ki bi se nahajala na različnih merilnih prostorih. S tem bi enostavno zagotovili ponovljivost meritev v smislu orientacije lidarja. Modul za avtomatsko popravljanje položaja nebi imel več bistvene vloge. Takoj omenimo, da bi vseeno ohranil del svoje namembnosti in sicer v

primeru, da je področje merjenja delno gibljivo. Glede na to, da se bo lidarski sistem uporabljal v luki, lahko pričakujemo, da bo uporabnik izbral merilni položaj na ladji, ki se premika za nekaj metrov v svojem privezu. Bralec bo morda dobil idejo, da bi sistem lahko uporabljali za merjenje premičnih tarč. Kljub temu, da smo razmišljali o ideji, prvotni cilji možnosti niso vključevali. Če se bo v prihodnosti izkazala potreba, lahko programsko opremo ustrezno razširimo.

Poleg naštetih ciljev smo želeli, da bi bila gibalni modul in programska oprema v smislu napak čimbolj robustna in čista ter predvsem praktična in uporabna. Navsezadnje bosta oba služila v dejanski uporabi in ne zgolj kot teoretična osnova, ki jo bo morda nekoč nekdo uporabil. Delovala bosta dnevno, v službi človeka, z željo po čistejšem prostoru in okolju, ki pri današnjih smernicah čedalje bolj predstavlja razkošje.

Poglavje 2

Strojna oprema lidarskega sistema

2.1 Senzor sipane svetlobe

2.1.1 Način delovanja

Detektor vsebuje fotodiodo, ki je občutljiva na del spektra, v katerem oddaja laser. Da čim bolj zmanjšamo ozadje, je pred fotodiodo postavljen filter, ki prepušča le v ozki okolici valovne dolžine laserja ($\lambda = 1574$ nm). Ob laserskem sunku se v senzorju aktivira števec, ki šteje s frekvenco $\nu_s = 7,37$ MHz. Ta je hkrati tudi vzorčevalna frekvenca, s katero se vzorči intenziteto odbite oz. sipane svetlobe, ki pade na fotodiodo. Dolžinska ločljivost je tako $c/(2 \cdot \nu_s = 20,3$ m). Glede na to, da meritev vsebuje 256 vzorcev, to ustreza razdalji $256 \cdot 20,3$ m $= 5,2$ km. Razdalja je morda majhna, vendar pri tej valovni dolžini in jakosti zaradi sipanja žarka v zraku, meritve pri večjih razdaljah niso več smiselne.

2.1.2 Komunikacija z računalnikom

Senzor komunicira s kontrolnim PC preko serijskega RS-232 vmesnika (COM vrata). Za delo z njim uporabljamo množico ukazov, nazaj pa nam pošilja informacije o stanju, meritvah in kontrolne podatke.

Format ukazov in odgovorov

Komunikacija poteka pri 115200 bps, 8 bit, paritete ni in stop bit je eden.

- Ukaz **Read query**: "`CMD0`"
  Odgovor je "`C`". C je kontrolni bajt, ki je xor vsota bajtov prejetega ukaza.

- Ukaz **Run Acqusition**: "`CMD1`"
  Ukaz postavi senzor v stanje, ko le-ta čaka na sprožitev laserja. Ko se laser sproži, senzor opravi meritev in pošlje izmerjene podatke.
  Podatki meritve so oblike: "`CMDXHLHLHL...HLC`"
  X je številka izbranega podatkovnega tipa (4: *Data*, 5: *Background*, 6: *Data – Background*).
  256 parov HL, H visoki in L nizki bajt predznačenega 16bitnega celega števila skupaj

tvorijo vzorec meritve.
C je kontrolni bajt, ki je xor vsota prejšnjih bajtov v poslanem nizu.

- Ukaz **Stop Acquisition**: "`CMD2`"
  Ukaz postavi senzor v stanje, ki prepreči zajemanje podatkov senzorja.

- Ukaz **Force Trigger**: "`CMD3`"
  Podobno kot ukaz "`CMD1`".

- Ukaz **Set Data Type = Data**: "`CMD4`"
  V stanju zajemanja podatkov, bo senzor ob sproženem laserju pošiljal vrednosti meritve, izmerjene pri sipanju laserske svetlobe.
  Odgovor je "`C`". C je kontrolni bajt, ki je xor vsota bajtov prejetega ukaza.

- Ukaz **Set Data Type = Background**: "`CMD5`"
  V stanju zajemanja podatkov, bo senzor ob sproženem laserju pošiljal vrednosti ozadja, izmerjenega tik pred laserskim sunkom.
  Odgovor je "`C`". C je kontrolni bajt, ki je xor vsota bajtov prejetega ukaza.

- Ukaz **Set Data Type = Data – Background**: "`CMD6`"
  V stanju zajemanja podatkov, bo senzor ob sproženem laserju pošiljal vrednosti meritve, izmerjene pri sipanju laserske svetlobe z odštetim ozadjem.
  Odgovor je "`C`". C je kontrolni bajt, ki je xor vsota bajtov prejetega ukaza.

- Ukaz **Send Gain**: "`CMD7HLHLHL..HL`"
  Ukaz vsebuje potake o *gain* profilu v obliki 256 HL parov takoj za glavo ukaza. H predstavlja visoki in L nizki bajt 16-bitnega – predznačenega celega števila. *Gain* profil predstavlja relativne faktorje, s katerimi se po meritvi množijo istoležeči izmerjeni podatki in se vračajo v obliki, ki je navedena pri ukazu Run Acquisition.
  Odgovor je "`C`". C je kontrolni bajt, ki je xor vsota bajtov prejetega ukaza skupaj s podatki o *gain* profilu.

- Ukaz **Get Status**: "`CMD8`"
  Odgovor je "`C`". C je bajt, ki vsebuje podatek o statusu senzorja.

2.2 *Razred Sensor* - visokonivojski vmesnik za uporabo senzorja sipane svetlobe

2.2.1 Splošno

Da je delo s senzorjem na višjem nivoju poenostavljeno smo v jeziku *C#* razvili razred, ki generira RS-232 ukaze za senzor v skladu s klici uporabniku prijaznih metod. Zagotovljena je kontrola parametrov ukazov in kontrola dostave le-teh senzorju. Podatki, ki jih vrača senzor so formatirani v uporabniku prijazno obliko, prilagojeno neposredni nadaljni uporabi. Razred za komunikacijo s senzorjem uporablja razred `System.IO.Ports.SerialPort`, ki je že na voljo v razvojnem okolju.

2.2.2 Konstrukti

Konstruktor

- `public Sensor()`
  Ne naredi nič specifičnega, razen kreiranja razreda.

Delegat dogodka

- `public delegate void StatusAvailableEventHandler(object sender, string status)`
  Predpisana glava metode, ki upravlja z dogodkom StatusAvailable. V spremenljivki status se ob nastopu dogodka nahaja opis statusa.

Lastnosti (vse samo za branje)

- `public int ReceivedValidMeasurements`
  Lastnost vrne število veljavnih meritev od inicializacije senzorja.

- `public int ReceivedBadMeasurements`
  Lastnost vrne število neveljavnih meritev od inicializacije senzorja.

- `public short[] ReceivedMeasurement`
  Prek lastnosti so dosegljivi podatki zadnje veljavne meritve.

Dogodka

- `public event StatusAvailableEventHandler StatusAvailable`
  Dogodek nastopi, ko je na voljo kako statusno obvestilo.

- `public event EventHandler MeasurementReceived`
  Dogodek nastopi, ko so voljo novi podatki meritve. Načeloma se to zgodi takoj po sprožitvi laserja, ko senzor pošlje podatke nove meritev.

Izjemi

- `public class DataTypeException : System.Exception {}`
  Izjemo dobimo, ko smo pozabili nastaviti tip meritve.

- `public class CommandErrorException : System.Exception {}`
  Izjema nastopi, ko ukaz ni dostavljen pravilno.

Množica

- `public enum DataType {Data, Background, DataMinusBackground}`
  Množica določa vse možne tipe meritev.

Metode

- `public void OpenPort(int boudRate, string portName)`
  Inicializira RS-232 kanal, `boudRate` mora biti 115200, `portName` je poljuben, načeloma oblike "COMn", kjer je n ustrezno število.
- `public void ClosePort()`
  Zapre RS-232 kanal.
- `public void ReadQuery()`
  Kaj natančno naredi ukaz ni znano. Pred zajemom podatkov ga je potrebno izvesti.
- `public void RunAcqusition()`
  Pripravi senzor na zajemanje podatkov. Ob naslednjih proženjih laserja bodo meritve dostopne preko lastnosti ReceivedMeasurement.
- `public void StopAcquisition()`
  Ustavi zajemanje podatkov ob proženju laserja. Z drugimi besedami, ob naslednjih sprožitvah laserja ne bomo dobili podatkov meritev.
- `public void ForceTrigger()`
  Ukaz povzroči podobno kot ukaz `RunAcquisition()`, natančnejši podatki žal niso znani.
- `public void SetDataType(DataType type)`
  Nastavi tip `type` naslednje meritve.
- `public byte GetStatus()`
  Vrne status senzorja, predstavljen v enem bajtu.
- `public void SendGain(short [] gainData)`
  Pošlje *gain* profil `gainData` v senzor.

2.3 Napajalnik laserja

2.3.1 Splošno

Vse kar se v tem dokumentu nanaša na laser je v večinoma mišljeno na napajalnik laserja. Tam se nahaja vsa elektronika za krmiljenje laserja in za komunikacijo z računalnikom. Več o delovanju je možno najti v navodilih za uporabo [1]. Na tem mestu bomo navedli le podatke, ki so pomembni za razumevanje komunikacije napajalnika prek RS-232.

2.3.2 RS-232 Komunikacija

Prek vgrajenega vmesnika se lahko krmili prav vse možne nastavitve napajalnika, dodatno je mogoče zakleniti tipkovnico na napajalniku, da nebi prišlo do neželenih težav s programsko opremo.

Komunikacija poteka pri 9600 bps, 8 bit, paritete ni in stop bit je eden.

Sintaksa ukazov

Vsi ukazi napajalniku so oblike "**$**ime XX**C**", $ je znak za začetek ukaza, ime je ime ukaza, XX je pozitivno celo število, ki predstavlja parameter ukaza. C je znak za konec vrstice (ASCII 0xDH). Ukaz se interpretira šele po sprejetem znaku C, v primeru, da pride do napake vmes se ukaz lahko pošlje znova. Znak $ ponastavi vhodni vmesnik znakov v napajalniku.

Ukazi

- `FIRE 01`
  Zažene laser.

- `STOP 00`
  Ustavi laser.

- `MODE 01 – 03`
  Nastavi način delovanja laserja: 01 – zaporedno proženje, 02 – zaporedno proženje, vendar z omejenim številom sunkov, nastavljenih z ukazom "`BURST`", 03 – proženje z zunanjim sprožilecem.

- `LPRF 01 / 02 / 05 / 10 / 20`
  Nastavi frekvenco, s katero pulzira laser.

- `LAMP 00 – 01`
  00 ustavi pulziranje, 01 zažene pulziranje.

- `QSWITCH 00 – 01`
  00 ugasne *Q-Switch*, 01 prižge *Q-Switch*.

- `QFERQ 01 – 99`
  Nastavi *Q-Switch* zagon na vsakih 01 – 99 sunkov.

- `QDLY 00 – 99`
  Nastavi zakasnitev v sunkih do aktivacije *Q-Switch*.

- `EPFN 01 – 20`
  Nastavi energijo sunka. (Nastavitev vpliva na višino stolpca na kontrolni plošči napajalnika.)

- `BURST 00 – 99`
  Nastavi število zaporednih sunkov.

- `STATUS ?`
  Vrne stanje napajalnika odnosno laserja.

- `SAVE 01`
  Shrani trenutne nastavitve v trajni pomnilnik napajalnika.

- `KEYPAD 00 – 01`
  00 zaklene tipkovnico napajalnika, 01 odklene tipkovnico napajalnika.
- `ECHO 00 – 01`
  00 deaktivira pošiljanje ukazov nazaj na računalnik, 01 aktivira pošiljanje ukazov.
- `VERSION 01`
  Vrne verzijo sistema.

Odgovori

Glede na stanje pošiljanja ukazov nazaj na računalnik (*echo*) so odgovori sledeči. Če je *echo* aktiven, potem se nazaj pošlje vsak ukaz v celoti, razen vodilnega znaka "`$`". Za znakom "`C`" se vrne še znak "`L`", L - *line feed* (ASCII 0x0A).

V obeh primerih stanja *echo* se vsak ukaz potrdi z odgovorim "`OK`", če je le-ta veljaven. Odgovor "`?1`" se vrne, če ukaz ni prepoznan, "`?2`" če se ukaz sicer prepozna, a parameter ni v okviru predpisanih vrednosti.

Na ukaz "`STATUS ?`" dobimo odgovor o stanju napajalnika oz. laserja:

- "`00`" - Sistem deluje pravilno.
- "`01`" - Hladilna tekočina je prevroča.
- "`02`" - Oddaljeni spojni priključek ni v kratkem stiku.
- "`04`" - Tok hladilne tekočine je majhen.
- "`08`" - Kabli niso priključeni oz. pokrov ni pričvrščen.

Opomba:

Predvidevamo, da je v primeru več hkratnih napak v delovanju odgovor vsota zgornjih števil. Npr. "`05`" pomeni prevročo hladilno tekočino in majhen tok.

Obstaja še posebna oblika ukaza "`$ime ?C`", ki za odgovor vrne "`##`", kjer ## predstavlja vrednost nastavljeno z ukazom "`$ime ##C`".

2.4 *Razred Laser* - visokonivojski vmesnik za uporabo laseja

2.4.1 Splošno

Razred je implementiran v jeziku *C#* in ga uporabljamo kot vmesnik za krmiljenje laserja iz računalnika prek RS-232. Na ta razred se sklicujejo vse višjeležeče operacije, ki laser uporabljajo. Z vmesnikom je delo poenostavljeno, saj zagotavlja kontrolo nad parametri, ki so predstavljeni v bolj naravni obliki. Strojna oprema namreč prejema vse ukaze v obliki znakovnih nizov. Komunikacija je realizirana s pomočjo razreda `System.IO.Ports.SerialPort`. Komunikacija je obojestranska, da so napake manj vrjetne.

2.4.2 Konstrukti

Konstruktor

- `public Laser()`
  Ne naredi nič specifičnega, razen kreiranja razreda.

Delegat dogodka

- `public delegate void StatusAvailableEventHandler(object sender, string status)`
  Predpisana glava metode, ki upravlja z dogodkom StatusAvailable. V spremenljivki status se ob nastop dogodka nahaja opis statusa.

Dogodek

- `public event StatusAvailableEventHandler StatusAvailable`
  Dogodek nastopi, ko je na voljo kako statusno obvestilo.

Izjema

- `public class ParameterErrorException : System.Exception`
  Izjema nastopi, ko so parametri ukazov neveljavni.

Množica

- `public enum LaserMode {Continuous, Burst, ExternalTrigger}`
  Tukaj so našteti vsi možni načini delovanja laserja.

Metode

- `public void OpenPort(string portName)`
  Inicializira RS-232 kanal, `portName` je načeloma oblike "COMn", kjer je n ustrezno število.

- `public void ClosePort()`
  Zapre RS-232 kanal.

- `public void Fire()`
  Vključi laser, ki začne delovati v skladu z nastavljenimi parametri.

- `public void Stop()`
  Izključi laser.

- `public void SaveConfig()`
  Shrani trenutno konfiguracijo parametrov v interni pomnilnik laserja, ki se bodo naložili ob naslednjem zagonu napajanja.

- `public void SelectMode(LaserMode mode)`
  Izbere način delovanja laserja. Načini mode so našteti v množici `LaserMode`.
- `public void QSwitch(bool enabled)`
  Omogoči oz. onemogoči *Q-Switch* opcijo.
- `public void Keypad(bool locked)`
  Zaklene oz. odklene tipkovnico na napajalniku laserja.
- `public void SetBurstPulseCount(byte pulses)`
  Nastavi število sunkov laserja, `pulses`, ki se bodo zaporedoma izvršili, če je izbran način delovanja `LaserMode.Continuous`. Dovoljeno območje je od 0 – 99.
- `public void SetQSwitchFrequency(byte freq)`
  Zažene *Q-Switch* na vsakih `freq` sunkov. Dovoljeno območje je od 1 – 99.
- `public void SetQSwitchDelay(byte delay)`
  Zakasnitev aktivacije v številu sunkov `delay`. Dovoljeno območje je od 0 – 99.
- `public void SetEnergyLevel(byte energyLevel)`
  Nastavi energijo laserja, `energyLevel` pri posameznem sunku aktivacije. Dovoljeno območje je od 1 – 20.
- `public void SetRepetitionRate(int rate)`
  Nastavi frekvenco, s katero naj se proži laser po aktivaciji. Frekvenca je izražena v sunkih na sekundo, `rate`. Dovoljene vrednosti so 1, 2, 5, 10 in 20.

2.5 Gibalni modul za usmerjanje lidarskega sistema

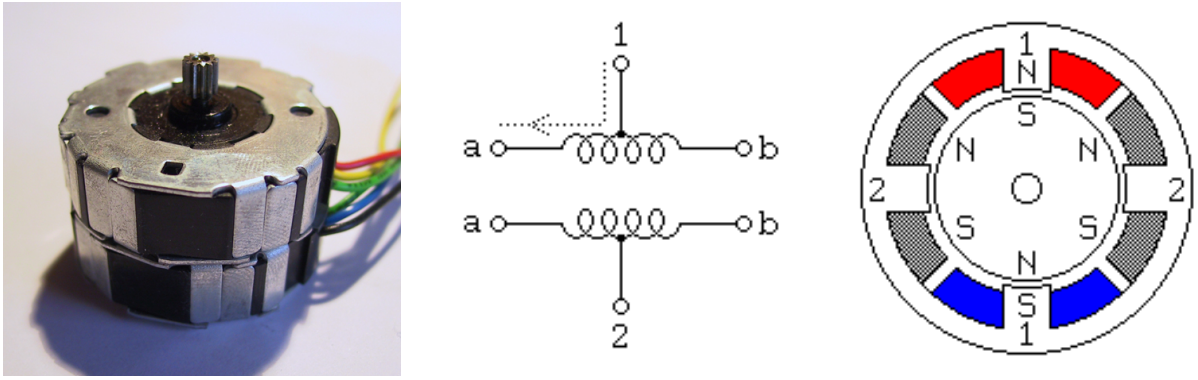
2.5.1 Splošno

Obstoječ sistem za krmiljenje teleskopa, ki ga lidarski sistem uporablja za sprejem sipane svetlobe, je namenjen opazovanju zvezd. Poleg tega žal ne omogoča priklopa preko PC. Da bi ga lahko uporabili za našo aplikacijo, smo se odločili, da izdelamo nadomestno vezje, ki bo komuniciralo preko serijskih RS-232 vrat COM na osebnem računalniku in krmililo originalno vgrajene koračne motorje. Stanje dogajanja se bo izpisovalo na zaslonu, ki bo skupaj z vezjem vgrajeno na mesto prejšnjega vezja. Vgadili smo še tipkovnico za ročno krmiljenje in za morebitne lokalne nastavitve ter daljinsko proženje sistema.

2.5.2 Koračna motorja

Obstoječa koračna motorja sta unipolarnega tipa (slika 2.1), kar poenostavi delo, saj unipolarni tip omogoča krmiljenje z najmanj močnostnimi tranzistorji. Za en motor zadoščajo štirje.

Koračna motorja sta mehansko vezana na pogonske osi preko reduktorjev, kar je nekoliko nerodno, saj za našo aplikacijo ne potrebujemo tako majhnih premikov, kot jih zagotavljata reduktorja. Celo nasprotno. Delo postane težavno, saj so premiki izredno počasni



Slika 2.1: Slika in shema unipolarnega koračnega motorja. **Levo**: koračni motor, ki ga uporabljamo v naši aplikaciji. **Desno**: shematični prikaz unipolarnega koračnega motorja s priključki.

in si moramo najprej pomagati z grobo ročno nastavitvijo. Ena večjih omejitev koračnih motorjev je njihova nizka največja hitrost vrtenja, zato je po pričakovanjih obračanje naprave izredno počasno.

Delovanje:

Unipolarni koračni motor vsebuje 4 tuljave (slika 2.1, desno). Da se rotor vrti, moramo ciklično pošiljati tok na tuljave v sledečem zaporedju: a1, b2, b1, a2, lahko pa uporabimo tudi polovične korake, tako da v vsakem drugem koraku tok teče skozi dve tuljavi naenkrat: a1, a1b2, b2, b2b1, b1, b1a2, a2, a2a1. Za vrtenje v nasprotni smeri se zaporedje korakov obrne. Več o krmiljenju koračnih motorjev naj bralec poišče v [16].

Dokler so preklopi počasni rotor z lahkoto sledi spreminjajočemu magnetnemu polju. Ob višji frekvenci lahko pride do težav, če magnetno polje prehiti rotor. V tem primeru se rotor navadno ustavi in rahlo vibrira na mestu. V našem primeru je pojav izredno pomemben, saj želimo motorje vrteti blizu maksimalne hitrosti, najhitreje kar je možno, ob še sprejemljivem navoru za premik naprave. Težave odpravimo tako, da dolžino korakov postopoma krajšamo do neke spodnje meje, ki zagotavlja najvišjo frekvenco. Ob zagonu motor tako nekaj 10 korakov pospešuje. Druga rešitev, ki jo uvajamo je že zgoraj omenjeno delovanje z vmesnimi koraki.

Implementacija – Strojni del

Za izdelavo modula smo uporabil *Microchip Inc.* mikrokrmilnik *PIC18F458*, [6], ki spada v družino mikrokrmilnikov z RISC jedrom. Zanj smo se odločili zaradi večih razlogov. Je kompakten in preprost za uporabo. Vgrajen ima RS-232 vmesnik, večje število ojačanih V/I kanalov, ki neposredno omogočajo vezavo močnostnih preklopnih tranzistorjev, [7] in tipk brez dodatnih *pull-up* uporov, ki so že vgrajeni. Odporen je proti motnjam v napajanju, tako da s preprostim regulatorjem napetosti dosežemo stabilno delovanje, [2]. Za generator ure podpira neposredno uporabo kristala z dvema pomožnima kondenzatorjema. Priklop *Hitachi* LCD modula je izvedljiv neposredno. Pri vezavi in programiranju prikazovalnika smo se opirali na [3]. Drugi razlog za izbor je dobra programska podpora.

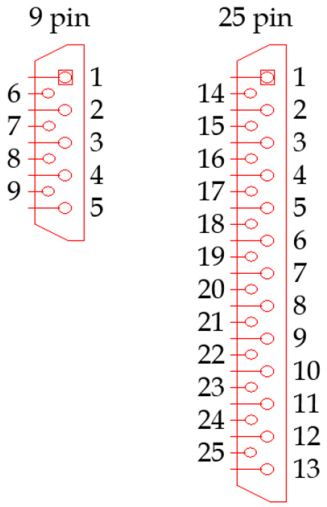
Ta je med drugim močno integrirano razvojno okolje, ki je brezplačno. Prav tako je na voljo dober *C* prevajalnik z obilo knjižnicami, katerega uporaba je za študente za 60 dni brezplačna. Tretji razlog so izkušnje pri delu z njim iz prejšnjih projektov, kjer je bil izdelan tudi programator po navodilih [18], ki ga potrebujemo, da prenesemo kodo iz računalnika v mikrokrmilnik. Za to nalogo smo uporabili program *IC – prog 1.05D*, [19]. (Programator je sicer možno kupiti.) Nenazadnje je tudi cena mikrokrmilnika relativno nizka. Cena podpornih komponent je minimalna.

Priklop na računalnik

Priklop na računalnik je razviden iz spodnje sheme RS-232 priključkov (slika 2.2) in tabele signalov (tabela 2.1), ki jih srečamo pri PC kompatibilnih računalnikih. Podatke o priključkih smo dobili v [22].

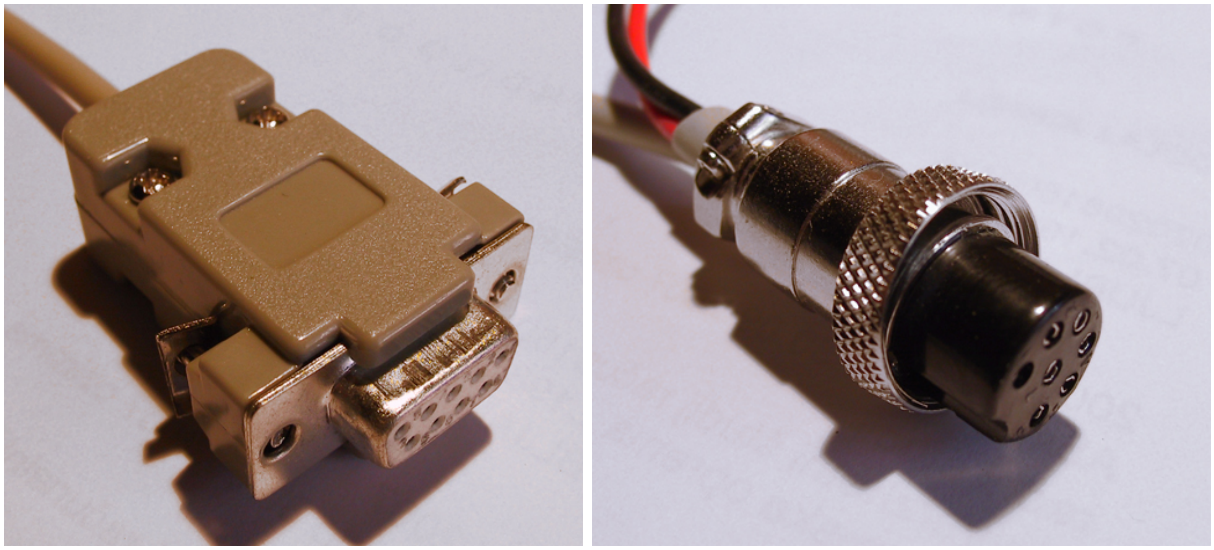
9 kontaktov	25 kontaktov	Signal	Smer	Opis signala
2. nožica	3. nožica	RXD	←	Prejmi podatke
3. nožica	2. nožica	TXD	→	Pošlji podatke
4. nožica	20. nožica	DTR	→	Pod. pripravljeni
5. nožica	7. nožica	GND	↔	Zemlja sistema

Tabela 2.1: RS - 232 signali, ki jih uporabljamo v naši aplikaciji.

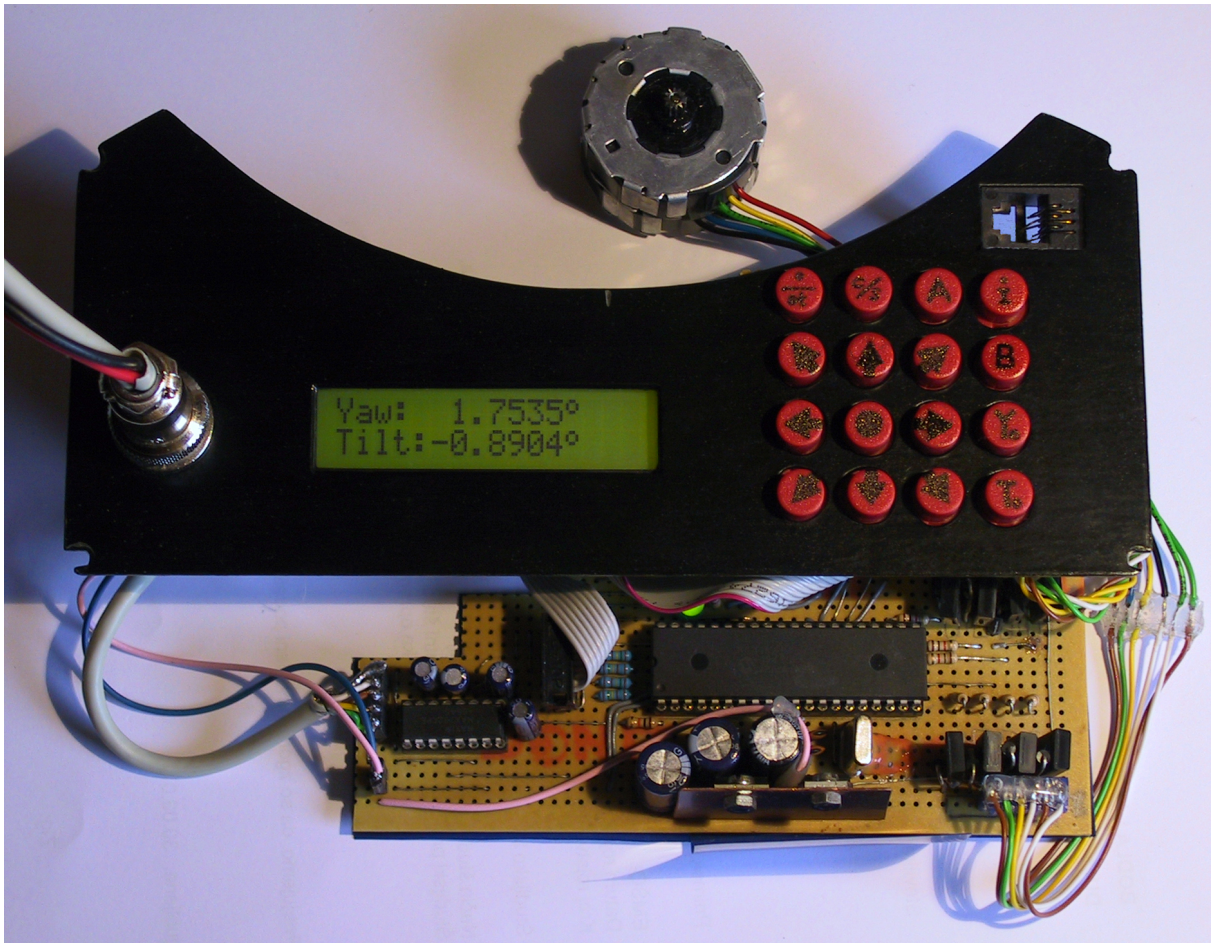


Slika 2.2: Shematični prikaz priključkov RS - 232 vmesnika pri PC kompatibilnih računalnikih.

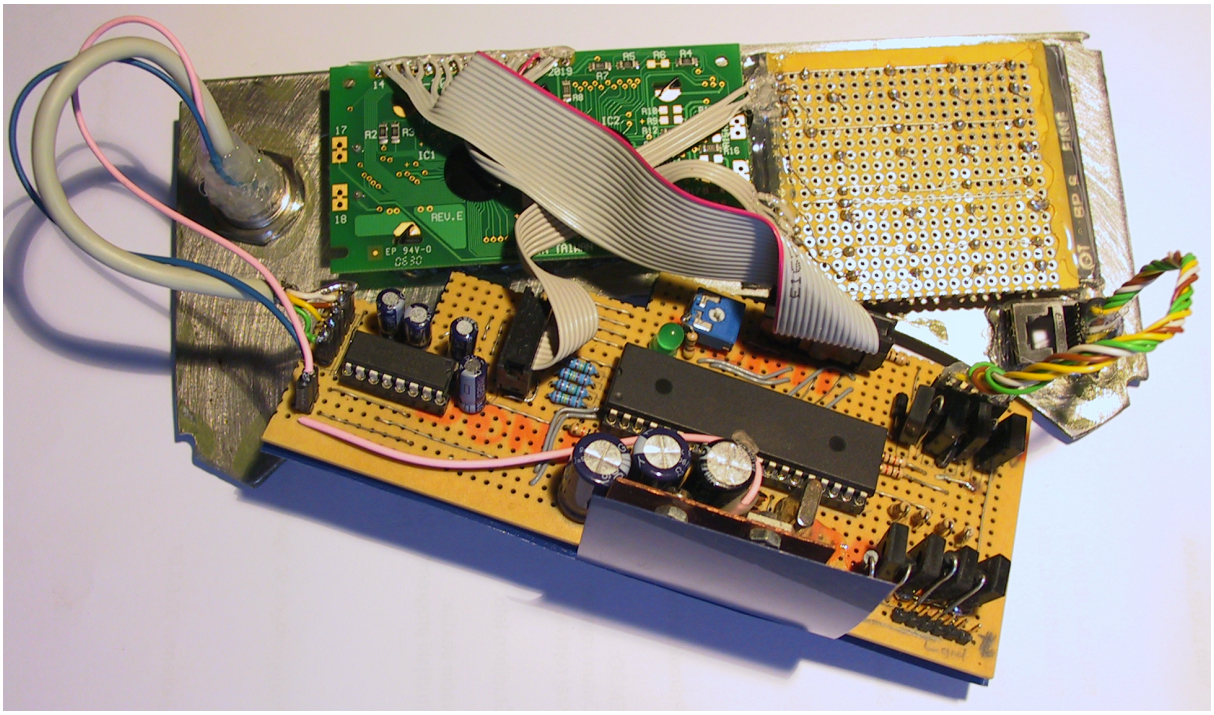
Na tem mestu velja omeniti, da mikrokrmilnik *PIC18F458* deluje na TTL napetostnih nivojih, kar ne sovpada z nivoji RS-232. Zaradi tega moramo povezavo realizirati prek pretvornika nivojev. Uporabili smo vezje *MAX232*, o katerem lahko bralec več prebere v [9].



Slika 2.3: **Levo**: 9 žilni priključek na računalnik. **Desno**: priključek na napravo. (Rdeča in črna žica sta za napajanje.)



Slika 2.4: Končan gibalni modul. Slikano iz prednje strani. Na modul je priključen koračni motor, viden na zgornjem delu slike.



Slika 2.5: Končan gibalni modul, slikan iz spodnje strani.

2.5.3 Realizacija vezja

Vezje smo izdelali na prototipni plošči velikosti 14 x 6 cm, da jo je možno vgraditi v ohišje teleskopa. Izdelali smo tudi nov pokrov, ki pokriva vezje in nudi oporo priključku, prikazovalniku in tipkovnici. Izdelek je prikazan na slikah (2.4, 2.5).

2.5.4 Implementacija programske opreme

Vsa koda je napisana v jeziku *C* v okviru *Microchip Inc.* razvojnega okolja, [5, 4]. Ob nekaterih predpostavkah pisanje rutin neposredno v zbirniku ni bilo potrebno, vendar se analizi v zbirnik prevedene kode ni bilo mogoče povsem ogniti. Časovno kritične dele je bilo potrebno natančno ovrednotiti.

PIC18F458 v gibalnem modulu

Frekvenca delovanja jedra je določena z vhodnim kristalom, ki niha pri 10 MHz. Frekvenca se v vgrajenem PLL vezju mikrokrmilnika množi s 4. Delovna frekvenca jedra tako znaša 40 MHz in je tudi najvišja s katero mikrokrmilnik še zanesljivo deluje, a si zaradi kompleksnosti algoritmov nižje ne moremo privoščiti. Omeniti moramo še, da se za večino ukazov porabijo 4 periode. Podobno velja za vse časovnike. Njihova osnovna časovna enota je enaka 4 urinim periodam. Opomba: višja frekvenca delovanja pomeni tudi večjo porabo energije. V naši aplikaciji s tem nismo bistveno omejeni. Maksimalna poraba mikrokrmilnika je 1 W, ki v razmerah, ki jih nudi avtomobil, kot ciljno mesto namestitve, predstavlja zanemarljivo majhno porabo.

Strojni vmesnik v mikrokrmilniku za komunikacijo prek RS-232 zadošča le za dva bajta. Dejstvo nam povzroča nekaj težav, saj moramo sprejemanju ukazov dodeliti del procesorskega časa in to v točno določenem trenutku.

PIC18F458 vsebuje mehanizem za prekinitve. Pozna dva nivoja prekinitev, visoke in nizke. Za visoke je značilno, da jih ne smemo gnezditi, saj se ob prehodu v ISR kontekst shrani v nabor registrov, ki zadošča le za eno kopijo konteksta. (Tu gre omeniti, da mi je to dejstvo povzročalo nemalo preglavic, saj ni navedeno v dokumentaciji za *C*, žal le v specifikaciji *PIC18F458*, kjer sem po nekaj dneh frustracij le našel rešitev.) Nizke prekinitve lahko gnezdimo, saj se kontekst shranjuje na programski sklad, katerega velikost je možno nastaviti v povezovalnikovi skripti. Vseeno moramo računati z omejitvijo 32 gnezdenih klicev funkcij in prekinitev. Povratni naslovi se namreč shranjujejo na poseben "sklad" 32 registrov. Vsak služi enemu povratnemu naslovu.

Težave

Izpostavimo poglavitne težave, s katerimi smo se morali spopasti pri implementaciji:

- Kako, in če sploh, zagotoviti istočasno sprejemanje ukazov prek RS-232 in premikanje dveh motorjev?
- Kdaj in kako vračati informacijo o uspešnosti izvedbe posameznih ukazov?
- Ali med premikanjem motorjev poročati kake podatke?
- Kako realizirati pospeške in pojemke motorjev?
- Kako uporabiti tipkovnico?

Razmislek

Vedeti moramo, da imamo opravka s sistemom v realnem času, torej morebitne zakasnitve ne pridejo v poštev. Po drugi strani imamo na voljo samo eno jedro, ki ukaze izvaja zaporedno. Najti moramo nek razumen sporazum med zakasnitvami, ki so očitno neizbežne, ter med kompleksnostjo rutin, ki izvedejo zahtevane funkcije in obenem narekujejo dolžino zakasnitev.

Rešitev

Da ne bi komplicirali, smo se neposredno po ugotovitvi narave prekinitev odločili, da motorje krmilimo v okviru nizkih prekinitev, v okviru katerih teče tudi pregledovanje stanja tipkovnice. (Pred ugotovitvijo sem namreč implementiral ISR motorjev v visokih prekinitvah, kasneje sem del servisne rutine prenesel v glavni program, kjer se je ob pomoči semaforjev servisiral zahtevnejši del. Seveda je taka rešitev sila neugodna, če želimo v glavnem programu početi še kaj drugega. Morda bi slednja rešitev celo pravilno delovala, če nebi imel opravka še z eno napako s prekinitvami za tipkovnico, ki sem jo odpravil šele kak teden pozneje.)

Tipkovnica

Tipkovnica ima matrično sestavo, [10]. Deluje po sledečem načinu: ob nizki prekinitvi, ki jo sproži časovnik, se z nizkim signalom na posamezni nožici vezja U1 (slika A.1) (*RA0 – RA3*) aktivira posamezno vrstico. Na nožicah (*RB4 – RB7*) se nato prebere stanje tipke v ustreznem stolpcu. Ob prvi prekinitvi se aktivira prvo vrstico, ob naslednji se prebere stanje prve vrstice ter se jo deaktivira, obenem se aktivira druga vrstica. V tretji prekinitvi se prebere stanje druge vrstice, se jo deaktivira in aktivira tretjo... Stvar teče ciklično. Ideja je v tem, da ne uporabljamo zakasnitvenih funkcij, ki bi po nepotrebnem izkoriščali procesor. Vsekakor potrebujemo neko daljšo zakasnitev med aktivacijo in branjem, saj bi ob kratkih zakasnitvah ob pritisku na tipko zaznali neželjene šume, ki nastanejo ob preklopu vezja tipke. Na podlagi razlike stanja tipkovnice, shranjenega ob predhodni prekinitvi in trenutnega stanja, se določi katera tipka je bila pritisnjena. Tipka oz. njen znak se doda v vmesnik tipkovnice, ki ga praznemo s pomočjo vmesniških funkcij modula. Za vsak pritisk se doda en znak. Stanje vsake tipke je moč spremljati tudi neposredno, kadar nas zanima ali je tipka pritisnjena ali ne.

ISR za pregledovanje tipkovnice reaktivira prekinitve takoj, ko ponastavi zastavico časovnika. S tem je tipkovnica podrejena servisiranju prekinitev za motorje in zakasnitev servisiranja motorjev je tako kvečjemu nekaj ukazov.

PIC18F458, RS – 232 vmesnik

RS – 232 vmesnik se prazni v zanki glavnega programa. Glede na velikost vmesnika smemo pri polni hitrosti iz računalnika poslati le po 2 znaka naenkrat. V iteraciji glavnega programa ($\approx$ 2 ms) se v modulu ti znaki prepišejo v tabelarično spremenljivko, katere vsebina se po sprejemu 16 zaporednih znakov interpretira kot ukaz. V primeru, da je čas med posameznimi sprejetimi znaki prevelik (več kot $\approx$ 250 ms) se vsebina tabelarične spremenljivke zavrže. Ukaz je potrebno vnovič poslati. S tem je zagotovljena odpornost modula proti morebitnim napakam, ki bi nastale npr. ob napaki na računalniku, ki bi preprečila pošiljanje celega ukaza v okviru dovoljenega časa.

Sprejemanje ukazov poteka vzporedno z delovanjem motorjev, kar je občutna prednost pred predhodno implementacijo modula.

Prikazovalnik

Uporabili smo dvo-vrstični, 16 znakovni *Hitachi 44780* kompatibilni pikazovalnik, [3]. Za krmiljenje prikazovalnika uporabljamo prilagojene rutine iz knjižnic paketa *C*, [4]. Po shemi (A.1) smo definirali posamezne vhode in izhode mikrokrmilnika za komunikacijo s prikazovalnikom. Če bi želeli, bi sicer lahko prihranili 4 izhode, tako da bi prikazovalnik krmilili prek 4 krmilnih linij v 4-bitnem načinu, vendar nimamo stiske z izhodi, zato ukrep ni potreben.

Glavni program

Delovanje je sledeče: v zanki stalno pregledujemo vmesnik tipkovnice in navzočnost novega ukaza iz RS-232. V odvisnosti od ukaza oz. stanja tipkovnice se nastavijo spremenljivke,

ki določajo stanje posameznega koračnega motorja. Te so število korakov, smer vrtenja, začetna oz. končna kotna hitrost, največja kotna hitrost, pospešek in pojemek ter več pomožnih. Vsak ukaz iz RS-232 se potrdi z odgovorom.

Na tem mestu se izvaja tudi avtomatsko sporočanje položaja motorjev. Podrobnosti so v poglavju *RS-232 ukazi in povratna sporočila za uporabo s PC*, ukaz *AutoReport*, razdelek 2.5.5.

Koračna motorja in njuni servisni rutini (ISR)

Za vsak motor posebej teče časovnik, ki po preteku sproži nizko prekinitev. Ob klicu ISR se prekinitve avtomatsko maskirajo, ponastavi se zastavica časovnika, v registre izhodov za priključke tuljav motorja se zapišejo podatki za naslednji korak (slika A.1: *RC0 – RC3* oz. *RC4*, *RC5*, *RE0* in *RE1*). S tem se napetosti na izhodih ustrezno spremenijo. Takoj za tem prekinitve omogočimo. (Označimo ta trenutek z Δ.) Posodobijo se spremenljivke stanja motorja. Ta del je časovno zahteven, ker vsebuje računanje v plavajoči vejici. Od trenutka Δ naprej so prekinitve omogočene. Lahko pride do vgnezdene prekinitve drugega časovnika za drugi motor. Začne se izvajati ISR drugega motorja. Prvi motor zaradi tega ne zamudi nič, drugi pa v najslabšem primeru zamudi nekje za red 100 ukazov ≈ 10 μs, kolikor traja prvi del ISR (prvega motorja), ki se ga ne sme prekiniti, kar je pri $\approx 0,73$ ms/korak (izračun sledi kasneje) pri največji možni hitrosti relativno malo $\approx 1,4\%$. Tipično za uporabljene motorje, ki najhitreje delujejo pri $1,5$ ms/korak le $\approx 0,7\%$. Napaka se kompenzira v naslednjem koraku, ki bo nastopil toliko prej, kolikor je trenutni zamudil glede na predviden čas, izračunan v prejšnjem koraku. Paziti je treba samo še, da se ISR časovnika v celoti izvede pred naslednjo zahtevo za servisiranje tega časovnika, t.j. dolžina ISR za motorje sme biti dolga toliko kolikor ukazov je moč izvesti v času enega koraka, odnosno nekaj manj kot polovica te vrednosti, saj imamo 2 motorja. Manjši del se nameni tipkovnici in manipulaciji pri sharanjevanju konteksta pri prekinitvah. Če situacijo obrnemo, kjer poznamo dolžino ISR motorja, ki je po najdaljši poti izvajanja nekje ≈ 3500 ukazov, ISR tipkovnice ≈ 200, in nekaj za manipulacijo konteksta ≈ 100 ukazov. (Podatke smo izmerili s simulacijo.) Skupaj torej $\approx 2 \cdot 3500 + 200 + 100 = 7300$ ukazov. Čas koraka $t_0 = 7300 \cdot 4/40$ MHz $= 0.73$ ms in $\nu = 1370$ Hz. Izračunana frekvenca je približek teoretično najvišje frekvence pri kateri lahko hkrati delujeta oba motorja. Če se premika samo en motor naenkrat je maksimalna frekvenca 2 krat večja.

Premik motorja izgleda takole: motor se začne vrteti z začetno kotno hitrostjo in pospešuje do maksimalne hitrosti ter se nadalje vrti s konstantno hitrostjo, dokler ni potrebno začeti zavirati s pojemkom, ki traja do končne hitrosti (= začetna hitrost), nakar se motor ustavi.

2.5.5 Napotki za uporabo

Uporaba tipkovnice - funkcije

Gibalni modul omogoča vnos podatkov prek lastne tipkovnice (slika 2.6), kot tudi prek RS-232 vmesnika. Tipkovnica služi za ročne premike, za izbor izpisa koordinat na prikazovalniku bodisi v korakih motorjev, bodisi v kotnih stopinjah. Možen je izbor ročnih premikov, ki so bodisi zvezni, bodisi po posameznih korakih. Zvezni premik opišemo

takole: ob pritisku na smerno tipko v izbrani smeri motor najprej enakomerno pospešuje do predefinirane maksimalne kotne hitrosti, s katero se potem nadalje vrti. Ob spustu tipke motor začne enakomerno pojemati, dokler se ne ustavi. Če tipko pritisnemo še preden se motor popolnoma ustavi, ta začne pospeševati od trenutne hitrosti. V primeru tipke v diagonali se premikata oba motorja hkratno. Premik po korakih je podoben zveznemu premiku, le da je tokrat hitrost vrtenja od samega pritiska na smerno tipko konstantna in precej nižja v primerjavi s tisto pri zveznem premiku. Ob spustu tipke se motor hipoma ustavi. Opcija omogoča natančno nastavitev položajev motorjev.

V primeru zagona motorjev prek RS-232 je s pritiskom na tipko *stop* moč akcijo preklicati. Motorja se s pojemkom ustavita.

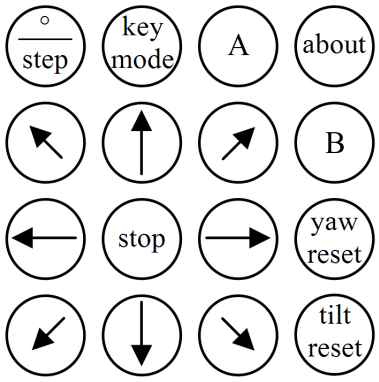
S tipkovnico je možno določiti izhodišče referenčnega koordinatnega izhodišča, t.j. položaj merjenja, v katerem sta relativni nagib in naklon enaka 0°.

Pritisk na tipko *about* prikaže uvodni zaslon z informacijo o avtorju in namembnosti.

Pritisk na tipki *A* oz. *B* povzroči takojšnje pošiljanje sporočila "**KEYA**xxxxxxxxxxxC" oz. "**KEYB**xxxxxxxxxxxC" prek RS-232. x so poljubni znaki, C je kontrolni bajt, ki je xor vsota prejšnjih ASCII kod znakov.

Opomba:

Med avtomatskim krmiljenjem motorjev prek RS-232 se tipkovnica delno zaklene, da nebi prišlo do nevšečnosti. Omogočena je funkcija tipke *stop*.



Slika 2.6: Tipkovnica, kot jo vidimo na gibalnem modulu.

Prikaz na prikazovalniku

Po vklopu ali ponastavitvi gibalnega modula se na prikazovalniku prikaže informativno obvestilo o avtorju in namembnosti. Kasneje prikazovalnik izmenično prikazuje položaj naprave, t.j. naklon in odmik glede na izhodišče referenčnega koordinatnega sistema in stanje v katerem se nahaja modul. Primeri izpisov so na sliki 2.7.

RS-232 ukazi in povratna sporočila za uporabo s PC

Zaradi enostavnosti interpretacije je dolžina vseh ukazov in sporočil enaka 16 znakov. Komunikacija poteka pri 115200 bps, 8 bit, paritete ni in stop bit je eden.

Slika 2.7: Primeri izpisov na prikazovalniku gibalnega modula.

- **Ukaz Move:** "`MSDXXXXXXYYYYYYC`"
  S predstavlja indeks motorja 0 ali 1, D predstavlja smer + ali –, XXXXXX predstavlja število korakov (mora imeti morebitne vodilne ničle), YYYYYY predstavlja maksimalno kotno hitrost v korakih na sekundo (mora imeti morebitne vodilne ničle), C je kontrolni bajt, ki je xor vsota prejšnjih ASCII kod znakov.
  Ukaz nastavi parametre motorja za naslednji premik.
  Odgovor na ukaz s pravilnim C je sporočilo "`KxxxxxxxxxxxxxxC`", z napačnim, sporočilo "`ExxxxxxxxxxxxxxC`", x so poljubni znaki, C kontrolni bajt enako kot pri ukazih.

- **Ukaz Run:** "`RUN C`"
  C je kontrolni bajt, ki je xor vsota prejšnjih ASCII kod znakov.
  Ukaz požene motroja, v skladu s prej navedenimi ukazi *Move*.
  Odgovor na ukaz s pravilnim C je sporočilo "`KxxxxxxxxxxxxxxC`", z napačnim, sporočilo "`ExxxxxxxxxxxxxxC`".
  Po končanem premiku se pošlje sporočilo "`DxxxxxxxxxxxxxxC`". x so poljubni znaki, C je kontrolni bajt enako kot pri ukazih.

- **Ukaz AccelerationSet:** "`AS XXXXXX C`"
  S predstavlja indeks motorja 0 ali 1, XXXXXX predstavlja pospešek v desetinkah koraka na sekundo^2, C je kontrolni bajt, ki je xor vsota prejšnjih ASCII kod znakov.
  Nastavi velikost pospeška za motor S.
  Odgovor na ukaz s pravilnim C je sporočilo "`KxxxxxxxxxxxxxxC`", z napačnim, sporočilo "`ExxxxxxxxxxxxxxC`", x so poljubni znaki, C kontrolni bajt enako kot pri ukazih.

- **Ukaz DecelerationSet:** "`DS XXXXXX C`"
  S predstavlja indeks motorja 0 ali 1, XXXXXX predstavlja pojemek v desetinkah koraka na sekundo^2, C je kontrolni bajt, ki je xor vsota prejšnjih ASCII kod znakov.
  Nastavi velikost pojemka za motor S.
  Odgovor na ukaz s pravilnim C je sporočilo "`KxxxxxxxxxxxxxxC`", z napačnim, sporočilo "`ExxxxxxxxxxxxxxC`", x so poljubni znaki, C kontrolni bajt enako kot pri ukazih.

- **Ukaz ω_0Set:** "`AS XXXXXX C`"
  S predstavlja indeks motorja 0 ali 1, XXXXXX predstavlja začetno (končno) kotno hitrost v desetinkah koraka na sekundo, C je kontrolni bajt, ki je xor vsota prejšnjih ASCII kod znakov.

Nastavi začetno oz. končno kotno hitrost motorja S.
Odgovor na ukaz s pravilnim C je sporočilo "KxxxxxxxxxxxxxxC", z napačnim, sporočilo "ExxxxxxxxxxxxxxC", x so poljubni znaki, C kontrolni bajt enako kot pri ukazih.

- **Ukaz GetPosition:** "PG C"
  C je kontrolni bajt, ki je xor vsota prejšnjih ASCII kod znakov.
  Vrne položaj obeh motorjev.
  Odgovor na ukaz s pravilnim C je sporočilo "G LLLL llll C", z napačnim, sporočilo "ExxxxxxxxxxxxxxC".
  LLLL so bajti longinta (*big endian*), ki predstavlja položaj motorja 0, llll so bajti longinta (*big endian*), ki predstavlja položaj motorja 1, x so poljubni znaki, C kontrolni bajt enako kot pri ukazih.

- **Ukaz SetPosition:** "GS PPPPPPPPPPP C"
  S predstavlja indeks motorja 0 ali 1, PPPPPPPPPPP predstavlja položaj v korakih (lahko negativno), C je kontrolni bajt, ki je xor vsota prejšnjih ASCII kod znakov.
  Nastavi položaj za izbrani motor.
  Odgovor na ukaz s pravilnim C je sporočilo "KxxxxxxxxxxxxxxC", z napačnim, sporočilo "ExxxxxxxxxxxxxxC", x so poljubni znaki, C kontrolni bajt enako kot pri ukazih.

- **Ukaz MotorOff:** "OFFS C"
  S predstavlja indeks motorja 0 ali 1, C je kontrolni bajt, ki je xor vsota prejšnjih ASCII kod znakov.
  Ugasne vse tuljave na motorju S. (Zavedati se moramo, da lahko ob izključitvi pride do neželjenega preskoka kakega koraka. Večina motorjev ob izključitvi ne preskoči koraka, a žal tega ne moremo zagotoviti za vse, ki bi jih bilo mogoče uporabiti skupaj z modulom. V primeru, ko je motor obremenjen z večjim navorom, si izklopa očitno ne moremo privoščiti.)
  Odgovor na ukaz s pravilnim C je sporočilo "KxxxxxxxxxxxxxxC", z napačnim, sporočilo "ExxxxxxxxxxxxxxC", x so poljubni znaki, C kontrolni bajt enako kot pri ukazih.

- **Ukaz LeftKeyPress:** "LP C"
- **Ukaz LeftKeyRelease:** "LR C"
- **Ukaz RightKeyPress:** "RP C"
- **Ukaz RightKeyRelease:** "RR C"
- **Ukaz UpKeyPress:** "UP C"
- **Ukaz UpKeyRelease:** "UR C"
- **Ukaz DownKeyPress:** "DP C"

- **Ukaz DownKeyRelease:** "DR C"
  C je kontrolni bajt, ki je xor vsota prejšnjih ASCII kod znakov.
  Zgornji ukazi se uporabljajo v paru. V bistvu ukazi povzročijo natanko enako akcijo, kot se zgodi pri pritisku na ustrezno tipko na tipkovnici gibalnega modula. Npr. najprej kličemo ukaz *UpKeyPress*, ki simulira pritisk tipke ↑ na modulu. Tako se motor 1 začne vrteti in se vrti, kot bi bila tipka ↑ nenehno pritisnjena. Po sprejetm ukazu *UpKeyRelease* se motor s pojemkom ustavi, podobno kot bi tipko ↑ spustili.
  Odgovor na zgornje ukaze s pravilnim C je sporočilo "KxxxxxxxxxxxxxxC", z napačnim, sporočilo "ExxxxxxxxxxxxxxC", x so poljubni znaki, C kontrolni bajt enako kot pri ukazih.
  Glejte tudi uporaba tipkovnice - funkcije, razdelek 2.5.5.

- **Ukaz MotorStop:** "SI C"
  Ukaz povzroči ustavitev motorja I, če je bil le-ta pognan z ukazom *Move*. Pri tem je ustavljanje pojemajoče in ne trenutno. C je kontrolni bajt, ki je xor vsota prejšnjih ASCII kod znakov.
  Odgovor na ukaz s pravilnim C je sporočilo "KxxxxxxxxxxxxxxC", z napačnim, sporočilo "ExxxxxxxxxxxxxxC", x so poljubni znaki, C kontrolni bajt enako kot pri ukazih.

- **Ukaz ContinuousMove:** "CONT C"
  C je kontrolni bajt, ki je xor vsota prejšnjih ASCII kod znakov.
  Ukaz nastavi premikanje motorjev v hitri način, t.j. način ko se motor ob pritisku na tipko začne vrteti s pospeškom in nadaljuje pri polni hitrosti do ustavitve s pojemkom ob spustu tipke.
  Odgovor na ukaz s pravilnim C je sporočilo "KxxxxxxxxxxxxxxC", z napačnim, sporočilo "ExxxxxxxxxxxxxxC", x so poljubni znaki, C kontrolni bajt enako kot pri ukazih.

- **Ukaz SingleStepMove:** "STEP C"
  C je kontrolni bajt, ki je xor vsota prejšnjih ASCII kod znakov.
  Ukaz nastavi premikanje motorjev v koračni način, t.j. način ko se motor ob pritisku na tipko vrti zelo počasi, tako da je s pritiski na tipko možno premikati motor za posamezen korak.
  Odgovor na ukaz s pravilnim C je sporočilo "KxxxxxxxxxxxxxxC", z napačnim, sporočilo "ExxxxxxxxxxxxxxC", x so poljubni znaki, C kontrolni bajt enako kot pri ukazih.

- **Ukaz LackCorrection:** "CS PPPPPP C"
  C je kontrolni bajt, ki je xor vsota prejšnjih ASCII kod znakov.
  Ukaz nastavi velikost korekcije na motorju S za odpravo mehanske nepopolnosti naprave (*lack*). PPPPPP predstavlja velikost korekcije, izražene v številu korakov. Velikost je lahko negativna (s predznakom je namreč določena smer korekcije). Privzeta vrednost je 0, t.j. korekcija je izklopljena.
  Delovanje: ko pri nastavljeni korekciji, t.j. različni od 0, izberemo premik motorja s korekciji nasprotno predznačemin premikom, se motor premakne za |PPPPPP|

korakov preko izbranega položaja in potem z nasprotno smerjo nazaj na položaj. V primeru dovolj velikega enakopredznačenega premika motor ne bo spreminjal smeri, če pa je premik manjši od korekcije, se motor premakne najprej za razliko premika in korekcije v nasprotno smer in nato v želen položaj s pričakovano smerjo.
Tak način premikanja nam zagotavlja, da se pri vsakem premiku motor v želen položaj premakne vedno v isti smeri, ki je pogojena s predznakom korekcije.
Opombi:

1. Pri premiku za 0 korakov (slepi premik) in nastavljeni korekciji bo motor opravil premik za korekcijo v eno in nato v drugo stran.

2. Na premike, povzročene s tipkovnico in ukaze *LeftKeyPress...*, ta opcija nima vpliva

Odgovor na ukaz s pravilnim C je sporočilo "`KxxxxxxxxxxxxxxC`", z napačnim, sporočilo "`ExxxxxxxxxxxxxxC`", x so poljubni znaki, C kontrolni bajt enako kot pri ukazih.

- **Ukaz AutoReport:** "`PA MMMM C`"
  MMMM je interval poročanja od 0050 ms do 9999 ms. Karkoli drugega povzroči izklop poročanja. C je kontrolni bajt, ki je xor vsota prejšnjih ASCII kod znakov.
  Ukaz nastavi avtomatsko poročanje položajev motorjev, ki se zgodi na vsakih MMMM milisekund.
  Odgovor na ukaz s pravilnim C je sporočilo "`KxxxxxxxxxxxxxxC`", z napačnim, sporočilo "`ExxxxxxxxxxxxxxC`", x so poljubni znaki, C kontrolni bajt enako kot pri ukazih.
  Oblika sporočila s položajem je sledeča: "`AxLLLLxllllxxxxC`". LLLL so bajti longinta (*big endian*), ki predstavlja položaj motorja 0, llll so bajti longinta (*big endian*), ki predstavlja položaj motorja 1, x so poljubni znaki, C kontrolni bajt enako kot pri ukazih.

Primer:
"`M0-003000011000C`"
"`M1+001000016000C`"
"`RUN C`"
(C ima ustrezno vrednost)

Zaporedje ukazov povzroči sinhroni premik motorjev začenši po sprejetju ukaza *Run*. Motor 0 se premakne za 3000 korakov v negativno smer, motor 1 pa za 1000 korakov v pozitivno. Motor 0 pospešuje do maksimalne hitrosti 110 korakov/s, motor 1 pa do 160 korakov/s. Pospešek in pojemek sta taka, kot smo ju naprogramirali v sam čip U1 (slika A.1) kot privzeti vrednosti oz. taka, kot smo ju nastavili z ustreznimi ukazi.

Opombe:

1. Posamezna operacija premika *Move* mora biti krajša od 200 sekund zaradi omejitve velikosti spremenljivk stanja motorja v modulu.

2. V primeru zahteve premika *Move* s frekvenco, za katero bi bila vsota korakov pospeševanja in pojemanja skupaj večja od števila korakov celega premika, se frekvenca ustrezno zmanjša. Kontrola je bila dodana zaradi naknadne implementacije in podpore ukazu *LackCorrection*.

3. Omejitev prve točke je ostala nerešena zaradi enostavnosti v implementaciji in je tako prepuščena višjemu nivoju. Ob neupoštevanju pride do nepredvidljivih rezultatov. Višji nivo jo zato mora upoštevati.

Ponastavitev gibalnega modula prek računalnika

Z negativno fronto signala DTR, ki ga deaktiviramo na PC je možno modul ponastaviti na enak način kot se zgodi pri njegovi priključitvi na napajanje. Uporaba je priročna v slučaju, ko bi se modul nemara iz takega ali drugačnega razloga prenehal odzivati na ukaze.

2.6 *Razred Motion* - visokonivojski vmesnik za uporabo gibalnega modula

2.6.1 Splošno

Razred je implementiran v jeziku *C#* in je namenjen komunikaciji z gibalnim modulom, ki je opisana v razdelku 2.5.5. Implementiran je vmesnik, ki zagotavlja zanesljivo delovanje in dostop z višjega nivoja do vseh funkcij, ki jih modul podpira. Obenem kontrolira še parametre ukazov modula. Za njihovo pravilnost v modulu ni popolnoma poskrbljeno. Komunikacija je realizirana s pomočjo razreda `System.IO.Ports.SerialPort`, ki je na voljo direktno v *C#*. Komunikacija je obojestranska, tako da je detekcija napak boljša.

2.6.2 Konstrukti

Konstruktor

- `public Motion()`
  Razen kreiranja instance razreda ne naredi nič specifičnega.

Struktura

- `public struct Position`
  Struktura vsebuje dve celoštevilčni spremenljivki preko katerih se posreduje informacija o položajih obeh motorjev.

Delegat dogodka

- `public delegate void AutoReportPositionEventHandler(object sender, Position position)`

Predpisana glava metode, ki upravlja z dogodkom `AutoReportPosition`. V spremenljivki `position` se ob nastopu dogodka nahaja lokacija obeh motorjev.

Dogodka

- `public event EventHandler ActionDone`
  Dogodek nastopi po končanem premiku motorjev.
- `public event AutoReportPositionEventHandler AutoReportPosition`
  Dogodek nastopi, ko avtomatično pride informacija o položaju iz gibalnega modula.

Izjeme

- `public class CommandErrorException : System.Exception`
  `CommandErrorException` nastopi, v primeru, če pride do napake pri komunikaciji z gibalnim modulom. Vzrok je, da v določenem času ni odgovora na poslan ukaz.
- `public class ParameterErrorException : System.Exception`
  `ParameterErrorException` nastopi, če so vrednosti parametrov metod vmesnika napačne.

Metode

- `public void OpenPort(int boudRate, string portName)`
  Inicializira RS-232 kanal, `boudRate` mora biti 115200, `portName` je poljuben, načeloma oblike "COMn", kjer je n ustrezno število.
- `public void ClosePort()`
  Zapre RS-232 kanal.
- `public void SetMove0(int steps, float freq, bool adaptFreq)`
- `public void SetMove1(int steps, float freq, bool adaptFreq)`
  Pripravi gibalni modul za naslednji premik motorja 0/1 za `steps` korakov, in maksimalno kotno hitrostjo `freq` v korakih/sekundo. Metoda upošteva pogoj, ki je omenjen v opombah pri ukazih gibalnega modula. S predznakom `steps` se določa smer vrtenja. `freq` mora biti vselej pozitivna. Parameter `adaptFreq` pove, če se v primeru previsoke frekvence ta ustrezno prilagodi.
- `public void Run()`
  Požene motor, za katerega je bil gibalni modul pripravljen. To stori z ustrezno metodo `SetMove(.)`. V primeru, da sta bili klicani obe metodi, se bosta motorja sinhrono začela vrteti, vsak glede na svoje parametre.
- `public void SetInitialAngularVel0(float omega0)`
- `public void SetInitialAngularVel1(float omega0)`
  Metoda nastavi začetno hitrost za motor 0/1, `omega0`. Vrednost mora biti pozitivna, enote so koraki/sekundo.

- `public void SetAngularAcceleration0(float accel)`
- `public void SetAngularAcceleration1(float accel)`
  Metoda nastavi kotni pospešek za motor 0/1, `accel`. Vrednost mora biti pozitivna, enote so koraki/sekunda^2.
- `public void SetAngularDeceleration0(float decel)`
- `public void SetAngularDeceleration1(float decel)`
  Metoda nastavi kotni pojemek za motor 0/1, `decel`. Vrednost mora biti pozitivna, enote so koraki/sekunda^2.
- `public void MotorOff0()`
- `public void MotorOff1()`
  Izključi napajanje za motor 0/1. (Metodo je smiselno uporabiti, če dlje časa ne premikamo motorja in le-ta ni obremenjen.)
- `public Position GetPosition()`
  Metoda vrne relativna položaja motorjev glede na stanje, v katerem sta se nahajala, ko smo gibalni modul priklopili na napajanje oz. od zadnje ponastavitve. Položaja sta izraženi v korakih.
- `public void SetContinuousKeyMovement()`
  Metoda nastavi premikanje motorjev v hitri način, t.j. način ko se motor ob pritisku na tipko začne vrteti s pospeškom in nadaljuje pri polni hitrosti do ustavitve s pojemkom ob spustu tipke.
- `public void SetSingleStepKeyMovement()`
  Ukaz nastavi premikanje motorjev v koračni način, t.j. način ko se motor ob pritisku na tipko vrti zelo počasi, tako da je s pritiski na tipko možno premikati motor za posamezen korak.
- `public void SimulateLeftKeyPress()`
- `public void SimulateLeftKeyRelease()`
- `public void SimulateRightKeyPress()`
- `public void SimulateRightKeyRelease()`
- `public void SimulateUpKeyPress()`
- `public void SimulateUpKeyRelease()`
- `public void SimulateDownKeyPress()`

- `public void SimulateDownKeyRelease()`
  Zgornje metode se uporabljajo v paru. Metode povzročijo natanko enake akcije, kot jih povzroči pritisk in spust ustrezne tipke na tipkovnici gibalnega modula. Npr. najprej kličemo metodo `SimulateUpKeyPress()`, ki simulira pritisk tipke ↑ na modulu. Tako se motor 1 začne vrteti in se vrti, kot bi bila tipka ↑ nenehno pritisnjena. Po sprejetm ukazu `SimulateUpKeyRelease()` se motor s pojemkom ustavi, podobno kot bi tipko ↑ spustili.

- `public void AutomaticPositionReporting(short interval)`
  Vklopi avtomatsko poročanje položaja motorjev na `interval` milisekund. Legalne vrednosti so od 50 – 9999. S parametrom vrednosti 0 se poročanje izklopi.

- `public void ResetControler()`
  Ponovno inicializira gibalni modul.
  Opomba: Nekateri USB na RS-232 vmesniki ne podpirajo signala DTR, preko katerega je operacija realizirana, tako da obstaja možnost, da ukaz pod temi pogoji ne bo deloval.

- `public void StopMotor0()`

- `public void StopMotor1()`
  V primeru, da se motor 0/1 premika posledično na ukaz prek RS-232, `StopMotor`0/1`()` ukaz prekine premik in povzroči ustavitev motorja 0/1. Ustavitev je v skladu s pojemkom postopna in ne takojšna.

- `public void SetPosition0(int position)`

- `public void SetPosition1(int position)`
  Nastavi relativni položaj `position` motorja 0/1, izraženo v korakih.

- `public void SetLackCorrection0(short steps)`

- `public void SetLackCorrection1(short steps)`
  Nastavi velikost korekcije na motorju 0/1 za odpravo mehanske nepopolnosti krmiljene naprave. Velikost korekcije, izražene v korakih, `steps`, naj bo večja od same velikosti mehanske nepopolnosti naprave.

Poglavje 3

Programska oprema *jeVel* za delo z lidarskim sistemom

Slika 3.1: Začetno okence programa za delo z lidarjem. (Ideja o imenu *jeVel* izhaja iz angleške besede *jewel*, dragulj...)

3.1 Uvod

Pri razvoju programske opreme za lidarski sistem smo izhajali iz predhodno že razvite programske opreme, ki je bila pomanjkljiva in nezanesljiva. Za informacijo navajamo: pisana je v jeziku *Delphi*, avtor je žal neznan. V približku je obsegala to, kar sedaj ponuja osnovna meritev v novi programski opremi. Uporabili smo jo le v toliko, da smo razrešili nejasnosti pri krmiljenju senzorja pri lidarju, saj literatura o protokolih senzorja žal ni bila dostopna. Nova programska oprema je sicer napisana povsem znova.

Novosti obsegajo vse od boljše preglednosti, vključitve računalniškega vida prek *web*-kamere, kontrole modula za gibanje, različnih tipov meritev, shranjevanja in iskanja podatkov, podrobnih nastavitev naprav lidarskega sistema, ki se uveljavijo avtomatsko

ob zagonu, preglednega izvoza podatkov, do, nenazadnje, interaktivnega in privlačnega pogleda na vse skupaj.

Programska oprema je razvita v programskem jeziku *Microsoft C#*. Zanj smo se odločili iz večih razlogov. Med prvimi je bila želja, naučiti se novih tehnologij in jezika kot takega. Ima izredno močno razvojno okolje z enim najboljših razhroščevalnikov. Naslednji razlog je v modernosti, tako s strani prenosljivosti aplikacije v novejše operacijske sisteme, kot po izgledu. *C#* ima tudi izredno preprosto podporo večnitnosti in razvoju kontrol. Oboje se uporablja v novi aplikaciji.

Pri uporabi knjižnic za laser smo se opirali na dokumentacijo [1], za senzor, kot že omenjeno, nismo imeli na voljo virov, zato smo delovanje razbrali iz kode obstoječe programske opreme. Za programiranje *web*-kamere so nam v veliki meri pripomogli viri na internetu: [20] in drugi. Uporaba *web*-kamere praktično v celoti temelji na *Microsoft Windows* gonilniku video naprav, dostopnem preko API klicev. Modul za gibanje lidarskega sistema je bil v celoti razvit v tem diplomskem delu. Glede na to svobodo so prilagoditve modula in programske opreme obojestranke in razvoj obeh je potekal pretežno vzporedno.

3.2 Osnovno okno aplikacije

Osnovno okno v aplikaciji služi za osnovno navigacijo lidarskega sistema in predstavlja oporo za vsa ostala okna. Prikazano je na sliki 3.2.

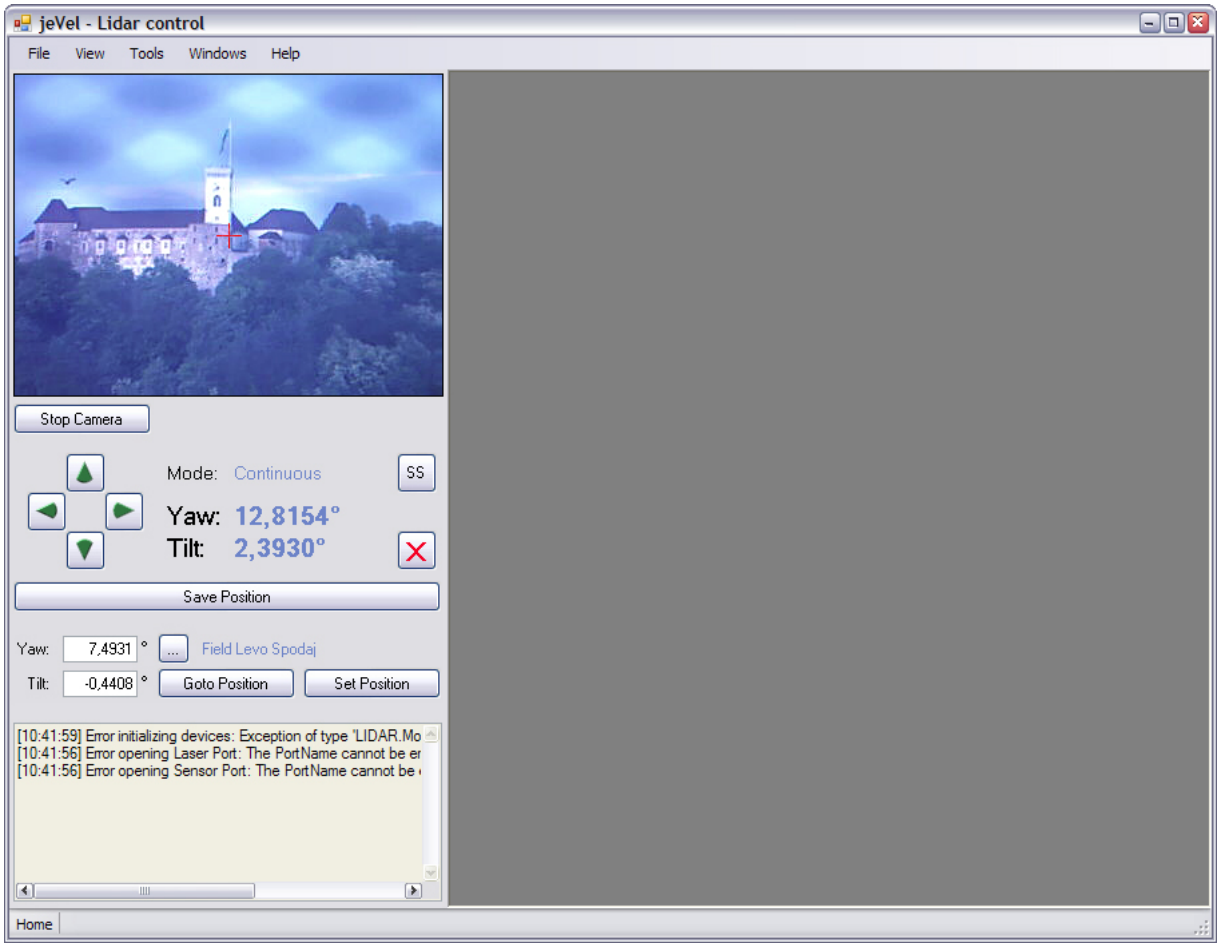
Skrajno zgoraj so meniji, ki nam služijo za izbor akcij:

- *File.* V tem meniju imamo možnosti *New* in *Open*, s katerima ustvarimo ali odpremo različne meritve:
  - Hitra, osnovna meritev (*Quick Measurement*)
  - Meritev v pravokotnem območju (*Field Measurement*)
  - Meritev v rezini (*Slice Measurement*)
  - Panoramska meritev (*Panorama*)

  Poleg tega v tem meniju lahko zamenjamo aktivno domeno delovanja aplikacije in urejamo *gain* profile ter aplikacijo zapremo.
- *View.* Vklopimo ali izklopimo statusno vrstico.
- *Tools.* Tukaj se nahaja meni za nastavitve lidarskega sistema.
- *Windows.* V tem meniju lahko urejamo pogled oken.
- *Help.* Vsebuje značko o avtorju programa.

Pod meniji se nahaja pogled kamere, ki nam služi za vpogled orientacije naprave. Rdeč križ na sredini nam omogoča natančno nastavitev položaja, kjer bo opravljena meritev. Kamero zaženemo s pritiskom na gumb *Start Camera*, ki se po zagonu preimenuje v *Stop Camera* in služi zaustavitvi kamere.

Spodaj imamo več krmilnih elementov za premikanje naprave. Gumbi s smernimi puščicami nam omogočajo premikanje naprave levo/desno/gor/dol. Pri oznaki *mode* je napisano na kakšen način se bo ob pritisku na smerne gumbe naprava premikala. *Continuous* predstavlja hitro, zvezno in približno premikanje s pospeševanjem in zaviranjem, način *Step* pa nam omogoča natančne premike za posamezen korak motorja. Z gumbom *SS* je možno preklapljati med načinoma.



Slika 3.2: Osnovno okno programa *jeVel*.

Poleg zgornjih oznak *Yaw* (odklon) in *Tilt* (nagib) sta napisani ustrezni vrednosti trenutnega položaja. Obliko izpisov je v nastavitvah naprav možno nastaviti na izpis v korakih ali na izpis v stopinjah.

Gumb z rdečim *X* nam v vsakem trenutku samodejnega premikanja naprave omogoča prekinitev gibanja.

Gumb *Save Position*, omogoča shranjevanje trenutnega položaja naprave za kasnejši priklic.

V okenci z oznakama *Tilt* in *Yaw* lahko po želji vnesemo položaj in s pritiskom na tipko *Goto Position* izvedemo premik na ta položaj.

S pritiskom na gumb [...] prikličemo shranjen položaj, katerega ime se izpiše poleg gumba [...], vrednost pa prenese v okenci.

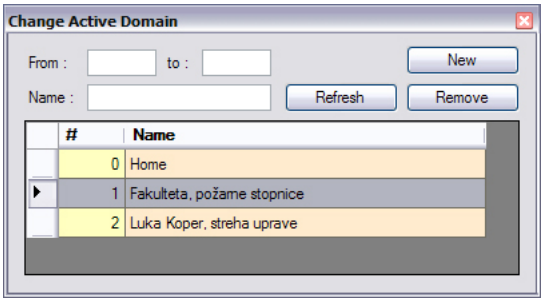
Gumb *Set Position* omogoča nastavljanje položaja naprave brez premikov. Vrednost okenc enostavno postane trenutni položaj naprave.

Na dnu se nahaja okence, v katerem se izpisujejo morebitne napake v delovanju naprave. Najprej se izpiše čas nato opis napake.

V spodnjem robu okna je vrstica stanja. V levem delu vrstice stanja je izpisano ime aktivne domene, v kateri teče aplikacija. Desno se izpisujejo priložnostna obvestila o dogajanju.

3.2.1 Izbira aktivne domene delovanja

File : Change Active Domain



Slika 3.3: Okno za izbor aktivne domene.

Aktivna domena delovanja je domena, v kateri poteka merjenje. Naprava je zasnovana kot mobilna, kar pomeni, da bomo merili na več lokacijah - domenah. Ob izbrani aktivni domeni se vsi potaki, t.j. meritve, položaji, rezine... shranjujejo pod njeno okrilje. Kjerkoli v aplikaciji bomo priklicali podatke, bodo vidni le pripadajoči aktivni domeni. S tem se ognemo preveliki zmedi.

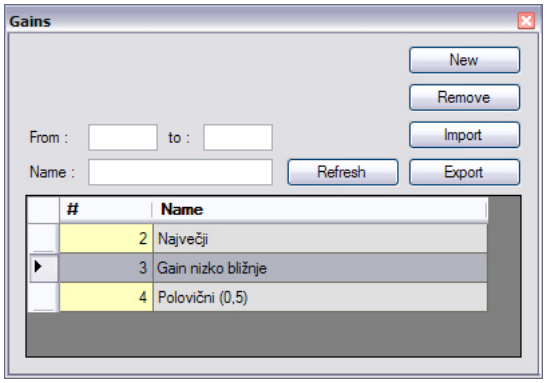
Ob prvem zagonu aplikacije se nam bo okno za izbor, slika 3.3, prikazalo samodejno. Ustvariti bomo morali novo domeno, jo izbrati in nato lahko nadaljujemo z delom. Izberemo jo z dvoklikom na vnos v tabeli oz. s pritiskom na *Enter*, ko smo v tabeli.

Levo zgoraj vidimo filtre, s katerimi je možno izbirati domene po zapisni številki od, do in po imenu. Vnose potrdimo z gumbom *Refresh* ali s pritiskom na *Enter*, ko smo v vnosnem okencu. Gumb *New* nam omogoča vstavljanje nove domene preko okna za vnos imena domene. (Potrdimo s *Save* ali prekličemo s *Cancel*). Gumb *Remove* služi brisanju izbranih domen. Brisanje je možno le, kadar se v izbranih domenah ne nahajajo podateki.

3.2.2 Urejanje *gain* profilov

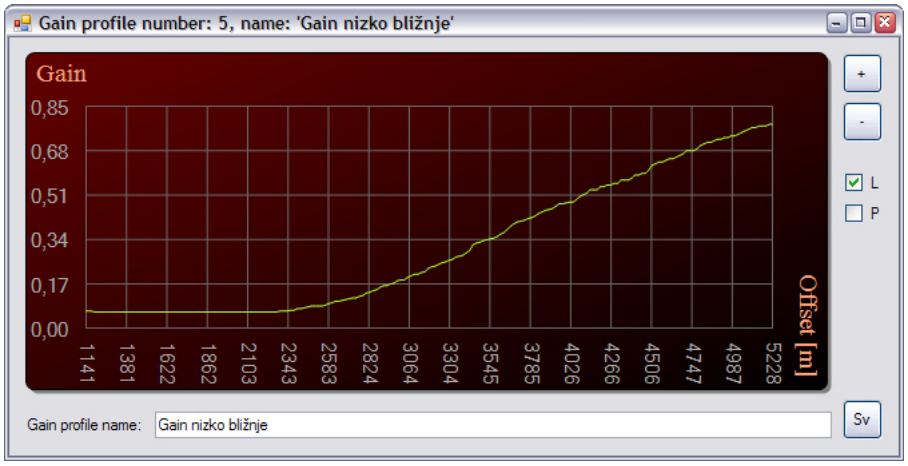
File : Manage Gain Profiles

Najprej splošno o *gain* profilu; *Gain* profil je zaporedje faktorjev (število faktorjev se določi v nastavitvah naprav), ki se ga pred vsako meritvijo pošlje v senzor. Veljavne vrednosti faktorjev so od 0 do 1. Z *gain* faktorji se v bistvu nastavi občutljivost senzorja za dani trenutek med potekom sprejemanja signala. Ta del se v celoti odvija v logiki senzorja.



Slika 3.4: Okno za delo z *gain* profili.

Z oknom na sliki (3.4) urejamo *gain* profile. Dvoklik na zapis tabele ali pritisk *Enter* v tabeli nam odpre izbran profil v posebnem oknu za prikaz in urejanje *gain* profila, slika (3.5). Z gumbom *New* ustvarimo nov profil, ki ga v istem oknu po želji oblikujemo in poimenujemo.



Slika 3.5: Prikaz in urejanje *gain* profila.

Gumb *Remove* odstrani v tabeli izbrane profile, če le-ti niso v uporabi pri kakšni meritvi.

S pritiskom na *Export* izvozimo izbrane profile v datoteke, katerih predpono xxx, njihovo končnico in njihovo ciljno mapo določimo v porajajočem pogovornem oknu. Po potrditvi se bo za vsak *gain* profil zgenerirala datoteka oblike xxxNNNNNNNN.txt, (ali z drugo izbrano končnico), kjer NNNNNNNN predstavlja številko *gain* profila. Primer izvoza *gain* profila v datoteko gn00000003.txt je v dodatku (razdelek A.2.1).

Gumb *Import* nam omogoča uvoz *gain* profila iz datoteke, ki naj bo po obliki enaka izvoženim datotekam. V primeru, da prvi dve vrstici manjkata, bo uvožen *gain* profil dobil ime po svoji datoteki. Preimenujemo ga lahko z urejanjem.

Pri pregledovanju *gain* profilov v tabeli si lahko pomagamo z vnosnimi polji filtrov po številki od, do, ter po imenu. Z *Enter* ali s pritiskom na *Refresh* potrdimo vnos.

Opomba: V primeru, da je profil uporabljen pri kateri meritvi, ga lahko le pregledujemo. Poleg ostalih podatkov se v glavi okna izpiše "*Read Only*" (Samo za branje).

V naslovni vrstici okna za prikaz in urejanje *gain* profila (slika 3.5) se nahajajo številka *gain* profila in njegovo ime.

Za grafični prikaz uporabljamo kontrolo *Graph*. Na ordinatni osi se nahaja velikost *gain* faktorja, na abscisi razdalja za katero se določen faktor uporablja. Z gumboma + in – lahko graf povečujemo ali zmanjšujemo. Isto dosežemo z dvoklikom leve tipke, odnosno dense tipke na miški. Kljukica pri L pomeni prikaz vezne črte med posameznimi točkami profila, kljukica pri P odebeli točke profila.

S pritiskom in držanjem desne tipke na miški lahko graf "primemo" in ga premikamo, sicer pa se nam v statusni vrstici osnovnega okna prikazuje položaj kurzorja na grafu. Ob preletu posameznih točk se te odebelijo in prikaz vrednosti se ustali, da je odčitavanje enostavnejše. S pritiskom na levo tipko miške lahko premikamo posamezne točke na grafu.

V okencu *Gain profile name* lahko *gain* profil preimenujemo.

Vsakršne spremembe uveljavimo s pritiskom na gumb *Sv*.

Kontrola *Graph*

Za prikazovanje osnovnih meritev je bila razvita posebna grafična kontrola *Graph*, katere izgled lahko vidimo npr. na slikah (3.5, 3.7, 3.8,...).

Kontrola omogoča prikaz različnih diagramov zveznih krivulj, podanih z zaporedjem točk $T = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$, ki se posredujejo s klicem metod kontrole. Pred uporabo je potrebno prilagoditi privzete intervale neodvisne in odvisne osi. Kontrola omogoča povečevanje in pomanjševanje grafa v poljubni točki. Zavoljo tega se nastavi intervale pogleda ob največji in najmanjši povečavi. Glede na velikost diagrama se v ozadju grafa velikost skale samodejno prilagaja. Graf se lahko prikazuje s točkami, s povezano lomljeno črto ali z obema. Pri vključenem urejevalnem načinu lahko uporabnik z miško premika posamezne točke prikazane krivulje. Pri tem se izbrana točka odebeli. Iz gostiteljskega okna so podatki o spremembah dostopni preko metod kontrole. Preko sporočil gostiteljsko okno dobiva informacije o položaju kurzorja na kontroli. Za ozadje grafa je možno izbirati med več stili (enobarvno, gradientno obarvanje, slika) in pri tem izbirati med različini barvami. Možno je izbirati barvo robov in zaobljenost ter debelino in barvo sence. Pri skali (mreži) lahko izbiramo barvo, odmik od robov kontrole in pisavo za prikaz vrednosti ter natančnost izpisa vrednosti. Določamo lahko imena osi in nastavljamo njihovo pisavo ter barvo. Ostale nastavitve so podedovane iz kontrole `UserControl`.

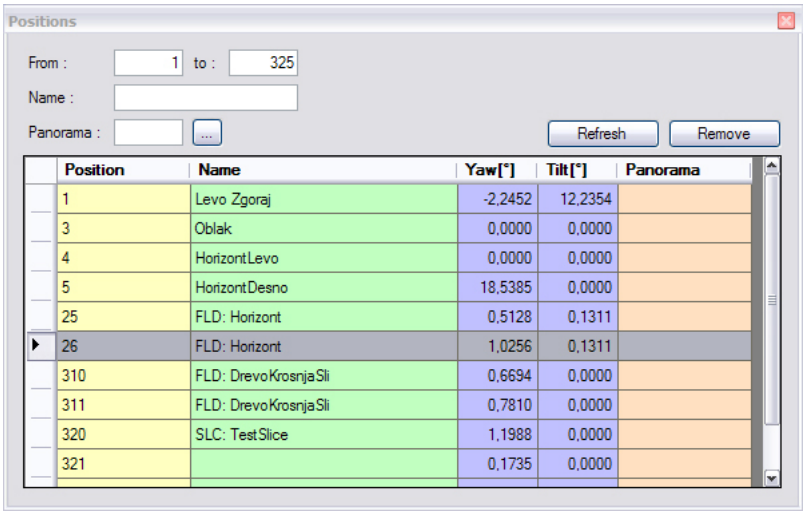
Pred izrisovanjem grafa se uporablja algoritem za rezanje daljic (*clipping algorithm*). Težava namreč nastane, ko je graf povečan in obsega območje, ki je večje od razpoložljivega prostora na mreži. Takrat se prikažejo le deli grafa, ki ležijo v območju pravokotnika robov mreže. Algoritem tekom izrisovanja za vsako daljico (del grafa med dvema točkama) izračuna tisti del, ki leži znotraj robov mreže. Ako cela daljica leži znotraj robov mreže, rezanje ni potrebno in izriše se cela. V nasprotnem primeru jo algoritem "oklesti" in izriše le del, ki leži znotraj robov mreže. Če daljica nima področja znotraj robov mreže, se je ne izriše.

3.2.3 Priklic položaja iz baze (gumb [...])

Na veliko mestih v aplikaciji lahko položaj prikličemo iz baze. Pri tem se nam odpre okno za izbiro položaja, ki je prikazano na sliki (3.6). Z dvoklikom na ustrezeno vrstico ali s pritiskom na *Enter* se nam ustrezni položaj kopira v polja, kamor smo želeli položaj vnesti.

Pri pregledu položajev si lahko pomagamo s tremi filtri. Prvi omeji področja glede na njihovo številko od, do. Z drugim izbiramo področje po imenu in s tretjim glede na panoramo, na kateri se področje nahaja. (Ni nujno, da področje sploh pripada kateri panorami.) S pritiskom na *Refresh* ali na tipko *Enter* uveljavimo kriterije.

Če želimo, lahko v tem oknu z gumbom *Remove* odstranimo kako področje.



Slika 3.6: Okno za priklic položajev iz baze.

Splošno o položajih

Položaji nam služijo kot podlaga za shranjevanje informacije o orientaciji naprave. Primarno jih shranimo za kasnejšo rabo. Bodisi jih shranimo iz osnovnega menija, bodisi na panorami. V slednjem primeru bo v pregledu položajev v stolpcu panorama zapisano ime dotične panorame.

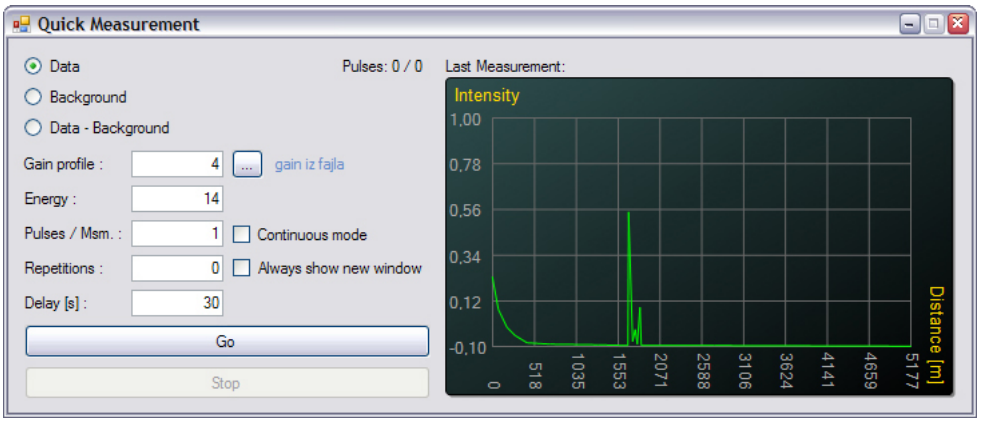
Pri meritvah, ki obsegajo več področij, t.j. pri (1) meritvah v pravokotnem območju in pri (2) meritvah v rezini se področja med merjenjem samodejno dodajajo v bazo. V primeru (1) se shranjena področja imenujejo "*FLD: <ime območja>*", v primeru (2) pa "*SLC: <ime rezine>*". Če za obstoječo orientacijo naprave področje že obstaja, se le-to ne bo shranilo.

Področja so relativna glede na ročno nastavljeno orientacijo naprave. O ročni nastavitvi orientacije naprave govorimo, ko jo premikamo z uporabo vijakov. Ob zagonu aplikacije se področje naprave niti ne spremeni, niti ne ponastavi. Le če naprave ne premikamo ročno, smemo področja jemati kot absolutna.

3.3 Merjenje (tipi meritev)

3.3.1 Hitra, osnovna meritev (*Quick Measurement*)

File : New : Quick Measurement...



Slika 3.7: Izgled okna za hitro, osnovno meritev.

Osnovno meritev uporabljamo takrat, ko želimo meriti v določeni smeri, brez avtomatske podpore za obračanje naprave. Načeloma ročno nastavimo položaj in izvedemo meritev. Podatki, ki jih prejmemo se najlaže predstavi v grafu intenzitete v odvisnosti od razdalje, ki ga lahko vidimo na desnem delu slike (3.7).

Posamezne meritve se avtomatsko shranjujejo v bazo podatkov in kadarkoli kasneje jih lahko prikličemo nazaj na zaslon.

Opomba: Vsi ostali tipi meritev temeljijo na osnovnih meritvah.

Postopek merjenja

1. Izberemo tip meritve. Bodisi zajamemo podatke, bodisi ozadje, bodisi se odločimo za razliko podatkov in ozadja. Načeloma uporabljamo zadnjo možnost.

2. Nadalje nastavimo *gain* profil. Bodisi vnesemo njegovo zaporedno številko zapisa v bazi, ali pa ga s klikom na gumb [...] prikličemo iz tabele *gain* profilov. V primeru priklica iz baze se nam desno od vnosnega polja izpiše tudi njegovo ime.

3. V polje *Energy* vpišemo intenziteto laserskega sunka (od 1 do 20).

4. Izberemo število laserskih sunkov za meritev (*Pulses / Msm.*). Vnose več kot 1 uporabimo, ko želimo imeti meritev z manj šuma. V tem primeru se namreč meritev obravnava kot aritmetična sredina podatkov, pridobljenih pri posameznem sunku.

5. Alternativo zgornji možnosti dobimo, če odkljukamo *Continuous mode* način, ki deluje na sledeč način. Laser se nastavi na avtonomno, ponavljajoče proženje, brez časovne omejitve, do preklica. Podatki vsake meritve se takoj prikažejo na grafu *Last Measurement*, pri tem pa se ne shranijo.

6. V polje ponovitev (*Repetitions*) vpišemo koliko meritev želimo opraviti poleg prve. V primeru ene same torej vnesemo 0.

7. Izbira *Always show new window* nam za vsako ponovitev meritve odpre novo okno z grafom meritve. Okna se sicer ne odpirajo. Izbiro je možno spreminjati med delovanjem. V tem primeru se odprejo okna le za tiste meritve, ki so bile opravljene tekom aktivne izbire. V vsakem primeru se nam meritve prikazujejo na grafu *Last Measurement*, kjer v vsakem trenutku vidimo zadnjo.

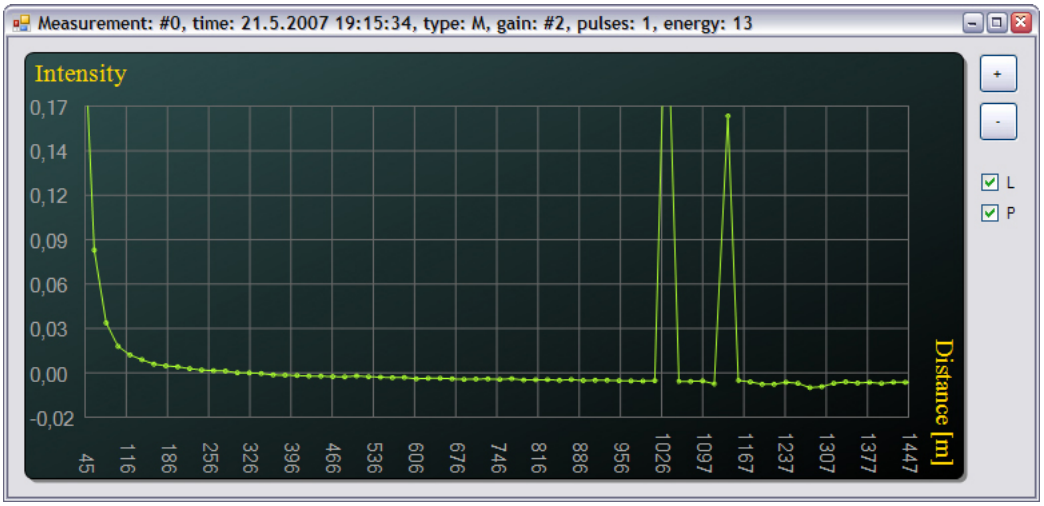
8. Vnos *Delay* pove koliko sekund naj aplikacija čaka med posameznimi ponovitvami.

9. Proces izvedemo s pritiskom na gumb *Go*.

10. Med zajemanjem podatkov lahko opazujemo števec sunkov (*Pulses*). Prvo število pove koliko sunkov je že opravljenih, drugo število koliko jih še manjka do zaključka.

11. Med delovanjem lahko proces prekinemo s pritiskom na gumb *Stop*. V tem primeru bo meritev temeljila na podlagi le do trenutka prekinitve prejetih podatkov. Tudi zapis za opravljeno število sunkov za prekinjeno meritev se v bazo zapiše, kot ustrezno manjši od predvidenega.

Prikaz osnovne meritve



Slika 3.8: Okno grafičnega prikaza osnovne meritve.

V naslovni vrstici okna osnovne meritve (slika 3.8) se nahajajo podatki o meritvi. Njena zaporedna številka, čas merjenja, tip meritve: M – podatki zmanjšani za ozadje, B – samo ozadje, D – samo podatki, uporabljen *gain* profil, število sunkov (*pulses*) in energija zajemanja (*energy*).

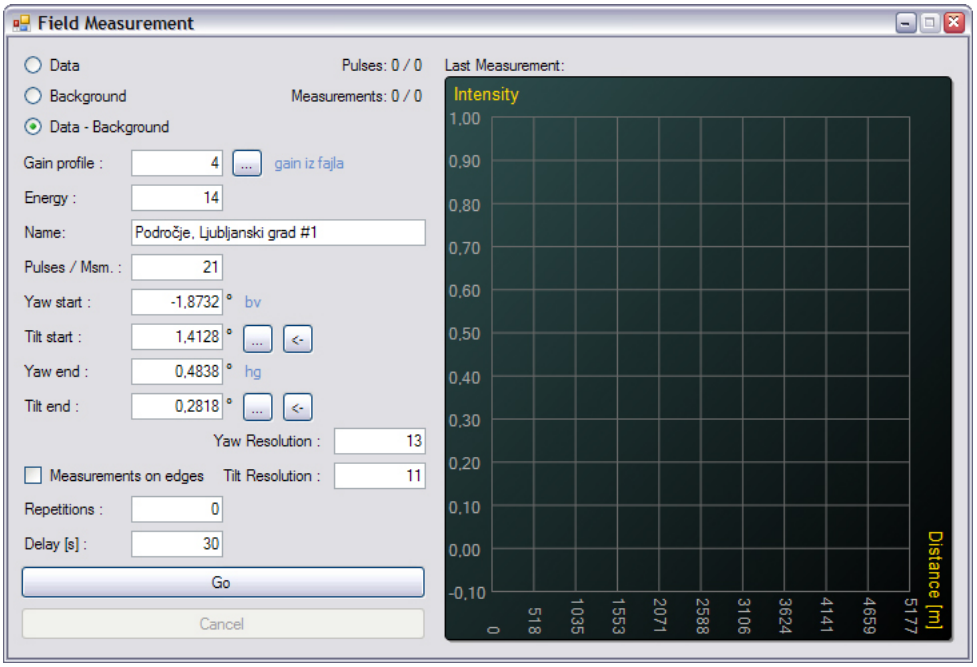
Za grafični prikaz uporabljamo kontrolo *Graph*. Na ordinatni osi se nahaja intenziteta signala, na abscisi razdalja od izvora laserskih sunkov. Intenziteta signala je preračunana na interval [0,1]. Zaradi procesiranja podatkov v senzorju smo včasih priča tudi rahlo negativnim vrednostim.

Z gumboma + in – lahko graf povečujemo ali zmanjšujemo. Isto dosežemo z dvoklikom leve miškine tipke, odnosno dense. Kljukica pri L pomeni prikaz vezne črte med posameznimi točkami meritve, kljukica pri P odebeli točke meritve.

S pritiskom in držanjem desne tipke na miški lahko graf "primemo" in ga premikamo, sicer pa se nam v statusni vrstici osnovnega okna prikazuje položaj kurzorja na grafu. Ob preletu posameznih točk meritev se te odebelijo in prikaz vrednosti se ustali, da je odčitavanje enostavnejše.

3.3.2 Meritev v pravokotnem območju (*Field Measurement*)

File : New : Field...



Slika 3.9: Izgled okna za meritev v pravokotnem območju.

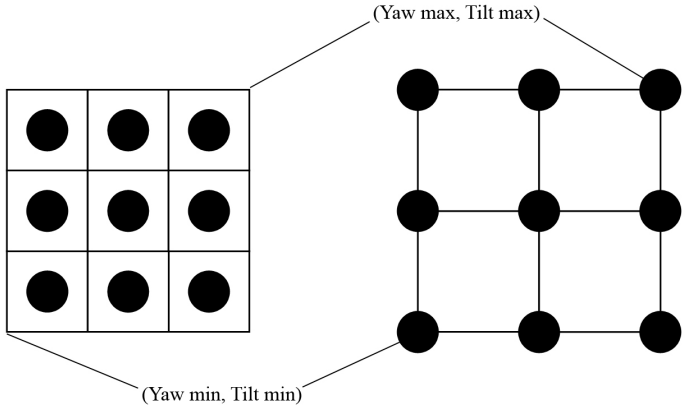
Namen

Meritev v pravokotnem območju uporabljamo, ko želimo opraviti več enostavnih meritev v pravokotnem območju.

Posamezna področja in njihove pripadajoče osnovne meritve se avtomatsko shranjujejo v bazo podatkov.

Postopek

1. Najprej izberemo tip meritve. Bodisi zajamemo podatke, bodisi ozadje, bodisi se odločimo za razliko podatkov in ozadja. Načeloma uporabljamo zadnjo možnost.



Slika 3.10: Področje meritev (črnih pik) v odvisnosti od izbire *Measurements on edges*: levo neoznačeno, desno označeno.

2. Nadalje nastavimo *gain* profil. Bodisi vnesemo njegovo zaporedno številko zapisa v bazi, ali pa ga s klikom na gumb [...] prikličemo iz tabele *gain* profilov. V primeru priklica iz baze se nam desno od vnosnega polja izpiše tudi njegovo ime.

3. V polje *Energy* vpišemo intenziteto laserskega sunka (od 1 do 20).

4. *Name*, navedemo ime področja, ki nam kasneje služi, da ga identificiramo.

5. Izberemo še število laserskih sunkov za posamezno meritev (*Pulses / Msm.*). Vnose več kot 1 uporabimo, ko želimo imeti meriteve z manj šuma. Računa se povprečna meritev.

6. Vnesemo omejitve področja, na katerem želimo meriti. Pri tem lahko prikličemo vrednosti iz baze s pritiskom na gumba [...] ali z gumboma [<-] kopiramo trenutni položaj naprave v vnosna polja.

7. Z izborom (*Measurements on edges*) se bodo meritve nahajale neposredno na robovih izbranega področja in enakomerno vmes (slika 3.9, desno). V naspotnem primeru bodo zamaknjene za pol enote (slika 3.9, levo).

8. Vnesemo še števili meritev v smeri odklona in nagiba, tako bo skupaj opravljenih (*Yaw Resolution*) x (*Tilt Resolution*) meritev.

9. V polje ponovitev, *Repetitions* vpišemo koliko področij želimo zajeti poleg prvega. V primeru le enega vnesemo 0.

10. Vnos *Delay* pove koliko sekund naj aplikacija čaka med posameznimi ponovitvami.

11. Merjenje zaženemo s pritiskom na gumb *Go*.

12. Med zajemanjem podatkov lahko opazujemo števec sunkov (*Pulses*). Prvo število pove koliko sunkov je že opravljenih, drugo število, koliko jih še manjka do zaključka trenutne meritve.

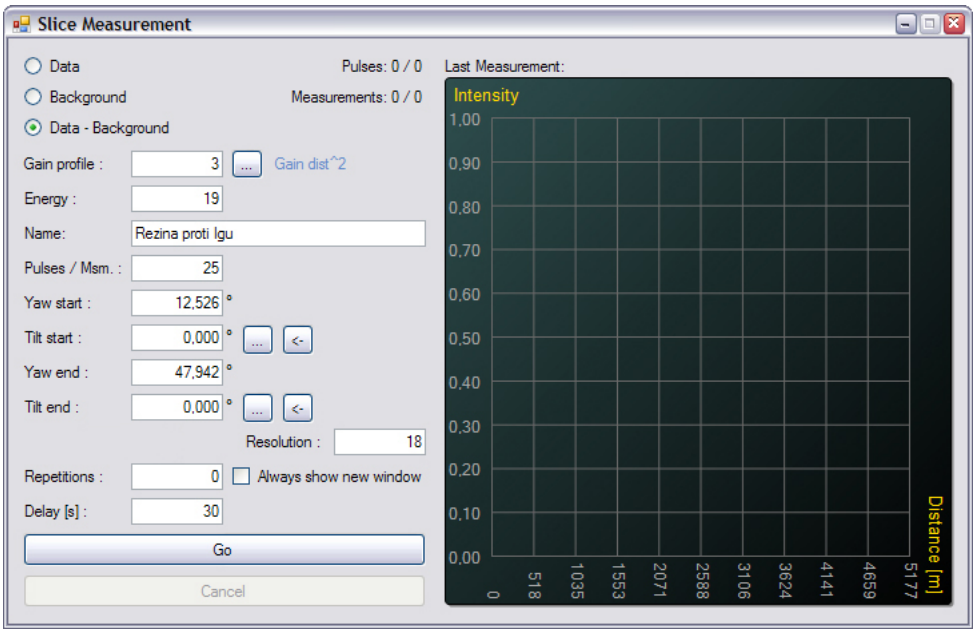
13. Opazujemo lahko še števec meritev (*Measurements*), ki ima podobno vlogo kot števec sunkov, le da ta šteje posamezne meritve.

14. Na desni se nahaj grafični prikaz zadnje meritve.

15. Med delovanjem lahko proces prekinemo s pritiskom na gumb *Cancel*. V tem primeru se bo področje v bazo zapisalo z oznako "nedokončano". Zapisale se bodo vse opravljene meritve. Zadnja bo morda zaradi prekinitve temeljila na manjšem številu sunkov kot je bilo zahtevano.

3.3.3 Meritev v rezini (*Slice Measurement*)

File : New : Slice...



Slika 3.11: Izgled okna za meritev v rezini.

Namen

Meritev v rezini je priročna, ko želimo iz meritev konstruirati sliko, podobno radarski. Zajame se podatke na poti od začetne do končne točke in sicer v enakomernih kotnih razmakih, začenši v začetni točki in z zadnjo meritvijo v končni točki. Prejeti podatki se avtomatsko shranjujejo v bazo ter se na koncu prikažejo v oknu, ki je posebej oblikovano za prikaz rezine.

Postopek

1. Najprej izberemo tip meritve. Bodisi zajamemo podatke, bodisi ozadje, bodisi se odločimo za razliko podatkov in ozadja. Načeloma uporabljamo zadnjo možnost.

2. Nadalje nastavimo *gain* profil. Bodisi vnesemo njegovo zaporedno številko zapisa v bazi, ali pa ga s klikom na gumb [...] prikličemo iz tabele *gain* profilov. V primeru priklica iz baze se nam desno od vnosnega polja izpiše tudi njegovo ime.

3. V polje *Energy* vnesemo energijo sunkov.

4. Navedemo ime rezine, ki nam kasneje služi za identifikacijo.

5. Izberemo še število laserskih sunkov za posamezno meritev (*Pulses / Msm.*). Vnose več kot 1 uporabimo, ko želimo imeti meriteve z manj šuma.

6. Vnesemo začetno in končno točko. Pri tem si lahko pomagamo s priklicom vrednosti iz baze s pritiskom na gumba [...] ali iz trenutnega položaja naprave [<-].

7. Vnesemo število meritev v rezini, (*Resolution*).

8. Za več zaporednih merjenj rezin vpišemo v *Repetitions* število ponovitev.

9. Izbira *Always show new window* nam za vsako ponovno izmerjeno rezino odpre novo okno s prikazom. Sicer se okna ne odpirajo. Izbiro je možno spreminjati med delovanjem. V tem primeru se odprejo okna le za tiste rezine, ki so bile izmerjene tekom aktivne izbire.

10. Vnos *Delay* pove koliko sekund naj aplikacija čaka med posameznimi ponovitvami.

11. Meritev izvedemo s pritiskom na gumb *Go*.

12. Med zajemanjem podatkov lahko opazujemo števec sunkov (*Pulses*). Prvo število pove koliko sunkov je že opravljenih, drugo število pa koliko jih še manjka do zaključka trenutne meritve.

13. Opazujemo lahko števec meritev (*Measurements*), ki ima podobno vlogo kot števec sunkov, le da ta šteje posamezne meritve.

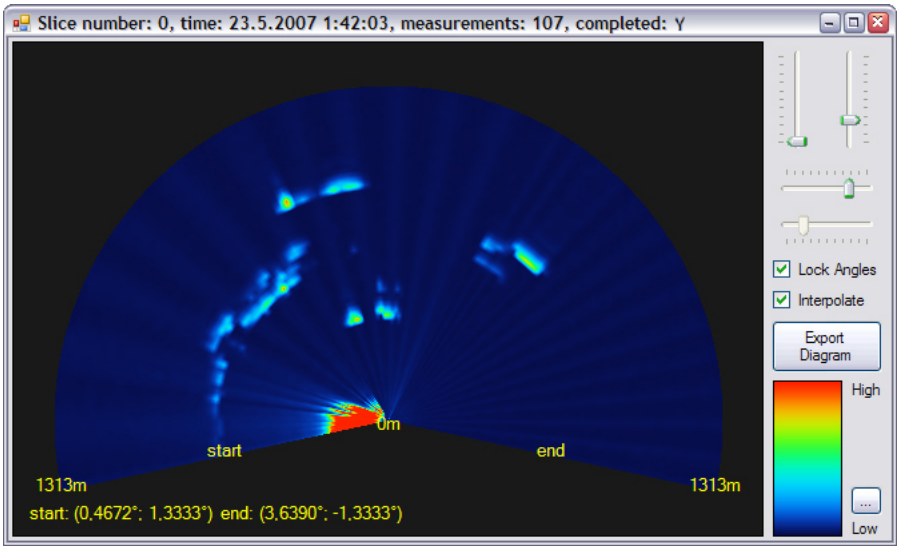
14. Na desni se nahaj grafični prikaz zadnje meritve.

15. Med delovanjem lahko proces prekinemo s pritiskom na gumb *Cancel*. V tem primeru se bo rezina v bazo zapisala z oznako "nedokončana". Zapisale se bodo vse opravljene meritve. Zadnja bo morda zaradi prekinitve temeljila na manjšem številu sunkov kot je bilo zahtevano.

Okno za prikaz rezine

V zgornji vrstici okna za prikaz rezine so podatki o prikazani rezini. Njena številka – *Slice number*, čas izmere – *time*, število osnovnih meritev – *measurements*, ki sestavljajo rezino in podatek ali je bilo zajemanje rezine dokončano – *completed* (Y – da, N - ne).

Diagram rezine v sredini okna je realiziran s kontrolo *LidarScan* in ima sledeč pomen; prva meritev rezine je označena na robu s *start* in je bila zajeta v položaju, spodaj označenim s *start: (yaw$_s$, tilt$_s$)*. Vmes so enakomerno prikazane vmesne meritve. Zadnja je označena z *end* in ji ustreza položaj *end: (yaw$_e$, tilt$_e$)*.



Slika 3.12: Okno za prikaz rezine.

Z navpičnimi drsniki zgoraj desno omejujemo prikaz meritev na določeno vzdolžno območje, ki je označeno na robovih in v sredini diagrama. Z vodoravnimi drsniki izbiramo začetni in končni kot prikaza rezine. *Lock angles* poskrbi, da sta kota simetrična in s tem diagram. (V opozorilo: koti merjenja nimajo nobene povezave s koti prikaza.)

Možnost *Interpolate* linearno interpolira vrednosti med posameznimi meritvami, tako radialno kot angularno. Na diagramu je posledica vidna kot gladkejši prikaz.

Z gumbom *Export Digram* lahko izvozimo diagram v slikovno datoteko. Ime ji določimo v pogovornem oknu, ki se prikaže ob pritisku na dotični gumb.

Desno spodaj je prikaz intenzitetne skale. V teh barvah je izrisan tudi diagram. Zgornji del predstavlja vrednost največjega vzorca katerekoli meritve rezine, spodnji del vrednost najmanjšega. Prikaz rezine je v tem smislu relativen. Z gumbom [...] lahko barvno skalo zamenjamo. V pogovornem oknu odpremo bitno sliko z barvnim prelivom, ki ponazarja skalo. Velikost slike naj bo 1x1000 slikovnih točk.

Kontrola *LidarScan*

Izgled kontrole vidimo na sliki (3.12), na levem delu okna. Kontrolo *LidarScan* smo razvili za interaktivni prikaz meritev.

Za zanimivost navedimo, kako se podatki izrisujejo. Izrisovalni algoritem preleti vse piksle kontrole in zanje izračuna barvo. Deluje v obratni smeri; kartezijske koordinate pretvori v polarne. Ne deluje v nasprotni smeri, kot bi nemara bralec sprva pomislil. Posredno, z uporabo funkcije $\arg(x - x_c + (y - y_c)i)$ se izračuna relativni kot piksla (x, y) glede na središče (x_c, y_c) in robove prikaza rezine. Izračuna se tudi Evklidsko razdaljo med pikslom in središčem rezine. Z dobljenima podatkoma se določi štiri najbližje vzorce iz dveh, izbranemu pikslu sosednjih meritev. Z dvakratno linearno interpolacijo se v angularni in radialni smeri določi vrednost piksla. Njegovo barvo se nato določi preko izbrane barvne skale.

Z implementacijo ni bilo večjih težav. Nekaj jih je povzročala le periodičnost kot-

nih funkcij. Za zanimivost navedimo, da moramo zato za poljubno izbiro začetnega in končnega kota prikaza obravnavati kar šest različnih poti algoritma. Podrobnosti naj si bralec pogleda v kodi.

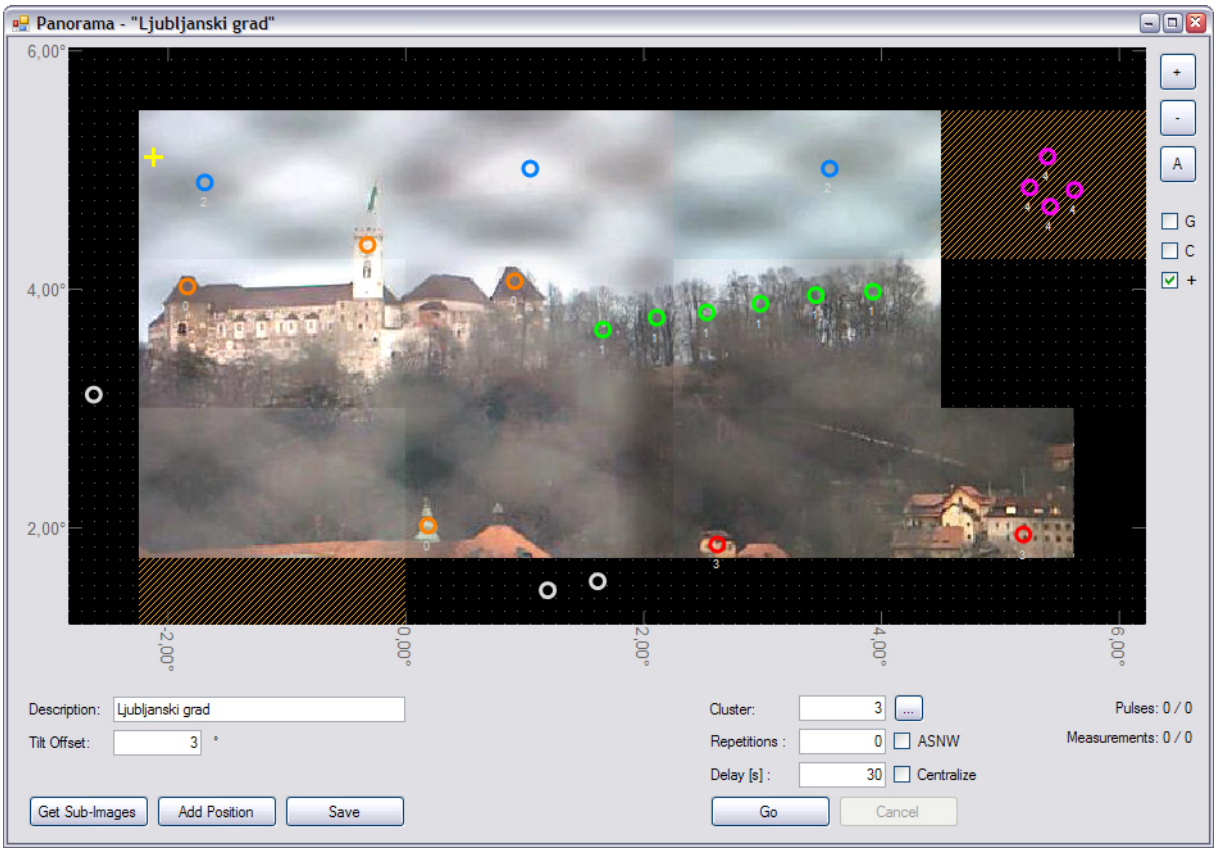
O lastnostih kontrole *LidarScan* na tem mestu ne gre razglabljati, saj je v namenu dokaj specifična in za razliko od kontrole *Graph* vrjetno ne bo našla uporabnega mesta zunaj aplikacije *jeVel*.

3.3.4 Panoramska meritev (*Panorama*)

Splošno

Panoramsko merjenje je verjetno najbolj uporabna vrsta meritve, ki jo ponuja programska oprema. Uporablja se za avtomatsko opravljanje enostavnih meritev in sicer na vnaprej določenih položajih. Implementirana je tudi podpora računalniškega vida, ki manjše odmike od prednastavljenih položajev samodejno popravi in meritev, glede na panoramsko sliko izvede na pravem mestu. Panoramsko sliko predhodno posnamemo.

Okno panorame



Slika 3.13: Okno za kreiranje in merjenje na panorami.

Na sliki (3.13) je prikazano okno panorame v tipičnem stanju, ki ga vidimo med uporabo. Pretežni del okna zajema kontrola *panorama*. Na njej so s krogci prikazani

merilni položaji in v ozadju je panoramska slika. Šrafirano območje je območje, ki ga želimo zapolniti z dodatnimi panoramskimi slikami. *Panorama* prikazuje tudi trenutni položaj orientacije naprave. Spodaj in levo se nahaja skala, s katero se lahko orientiramo v merilnem prostoru naprave.

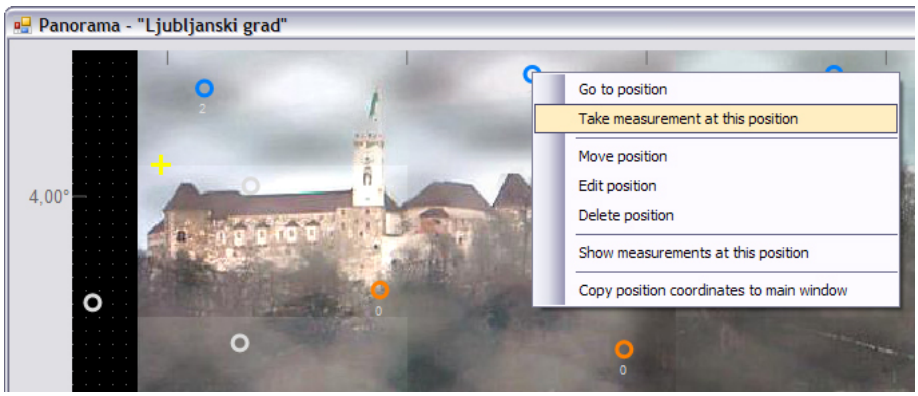
Opišimo sedaj namen in podrobnosti posameznih lastnosti. Merilni položaji so označeni kot krogci v različnih barvah. Pod vsakim krogcem je številka gruče, če ima ustrezni merilni položaj gručo določeno. Pripadnost gruči in položaj z natančno vrednostjo koordinat je moč videti v statusni vrstici osnovnega okna, če se na merilni položaj postavimo z miško. Tudi barva je odvisna od pripadnosti gruči. Pravzaprav barvo za vsako gručo posebej določi uporabnik. Na zahtevo se v merilnih položajih izvedejo enostavne meritve in sicer za posamezno gručo naenkrat. Gručo se določi v okencu *Cluster*. Pritisk gumba [...] nam pomaga pri izbiri gruče. Odpre se seznam vseh gruč, ki so določene za trenutno odprto panoramo. Ko s tipko *Go* sprožimo postopek merjenja, naprava sistematično obišče vse merilne položaje, ki izbrani gruči pripadajo. V vsakem trenutku je položaj naprave na panorami viden z rumenim križcem. Z desnim klikom miške na križec lahko le tega "premaknemo" na drugo mesto. S tem dejanjem interaktivno spremenimo položaj naprave. Ko sprostimo gumb, se bo naprava samodejno premaknila na izbrani novi položaj.

Parametre meritve, podobno kot barvo, določimo za vsako gručo posebej. V merilnih položajih, ki pripadajo dani gruči se torej izvajajo povsem enake meritve. Le-te se shranjujejo v tabelo enostavnih meritev. Obiskovanje merilnih položajev poteka zaporedno po požrešni metodi, upoštevajoč minimalno Evklidsko razdaljo med trenutnim položajem in merilnim položajem. Pravzaprav gre za problem trgovskega potnika, le brez obveznega povratka v izhodišče. V primeru, da meritve izvajamo periodično, nastavitev ponovitev *Repetitions* na več kot 0, pa gre za pravi problem trgovskega potnika. V tem primeru se namreč meritve opravlja ciklično. Ko se merjenje zaključi v zadnjem položaju, po določenem vmesnem premoru, določenim z *Delay* sekundami, nastopi ponovna meritev v prvem položaju.

Merilne položaje na panoramo dodajamo s tipko *Add Position*. Takoj ko kliknemo na izbrano mesto se nam prikaže okence za vnos podatkov novega položaja, kjer lahko poleg znanih koordinat določimo še ime položaja in pripadnost gruči. Obe polji sta za vnos neobvezni in obe lahko kasneje popravimo. S tem ko dodamo in shranimo novi položaj, se le-ta ustrezno obarva in prikaže na panorami.

Dodatne možnosti v zvezi z merilnimi položaji dosežemo z levim klikom na enega izmed njih. Odpre se meni, kot ga prikazuje slika (3.14). *Go to position* napravo premakne v izbran položaj. *Take measurement at this position* najprej premakne napravo v izbran položaj, nato odpre okno za enostavno meritev in nastavi parametre v skladu z gručo, ki ji merilni položaj morebiti pripada. *Move position* omogoča premik merilnega položaja. Ob izbiri se prikaže križ, kurzor miške, s katerim kliknemo na neki drugi lokaciji na panorami. Tja se preseli izbrani merilni položaj. Možnost *Edit position* odpre okence za urejanje parametrov merilnega položaja (položaj, ime, gruča). *Delete position* nas vpraša, če želimo položaj resnično in trajno izbrisati in ga v primeru potrditve odstrani iz baze. *Show measurements at this position* prikaže pregledno okno meritev (slika 3.21) z aktivnimi filtri, ki omejujejo poregled meritev na tiste, ki so bile izvedene v izbranem položaju. *Copy position coordinates to main window* prekopira koordinate položaja v

okenci *Yaw* in *Tilt* na osnovnem oknu.



Slika 3.14: Meni za delo z merilnimi položaji.

Če želimo na panorami videti le merilna področja, ki pripadajo določeni gruči, vnesemo njeno številko v okence *Cluster* ali jo prikličemo z gumbom [...] ter s kljukico označimo možnost *C*. Lahko prikažemo tudi področja, ki ne pripadajo nobeni gruči. To storimo, da vnos *Cluster* pustimo prazen.

Izbrana možnost *G* na panorami prikaže mrežo, s katero si lahko pomagamo za boljšo orientacijo. Gumba + in − se uporabljata za povečavo in pomanjšavo panorame. S pomočjo gumba *A* popravimo razmerje med naklonom in nagibom panorame, če smo s ga povečavo in pomanjšavo pokvarili. Po pritisku panorama dobi naravne proporce.

S pritisnjenim levim miškinim gumbom in premikom se na panorami bodisi označuje, bodisi briše območja za snemanje panoramske slike. Akcija je pogojena z izborom v okencu +. Kljukica narekuje označevanje, prazno polje narekuje brisanje. Preden se s pritiskom na gumb *Get Sub-Images* posname ali dopolni panoramsko sliko so področja na panorami označena z oranžno šrafuro. Nasprotno z desnim gumbom miške in premikom spreminjamo položaj pogleda na panoramo.

Snemanje panorame poteka v smeri najbližjega označenega področja glede na trenutni položaj naprave, t.j. po načinu trgovskega potnika s požrešno metodo. V sredini izbranih področij se zajame slike, odpravi se jim napake in jih začasno shrani v pomnilnik. Ob pritisku na *Save* se shranijo na disk.

Panorama omogoča zajem delno prekrivajočih se slik. Če bi nemara hoteli implementirati gladke prehode med slikami, bi nam možnost prišla prav. Za uporabo v naši aplikaciji to sicer nima velike teže. Videli bomo, da so opisniki značilk na slikah dokaj robustno zasnovani.

Nastavitev *Tilt Offset* določa odmik sredine panorame od vodoravne smeri. Včasih namreč (v skrajnem primeru) želimo vse meritve panorame izvajati pod ali nad obzorjem. Na primer iz visokega stopla proti nižje ležečim področjem. Več o tem v razdelku 3.3.4.

S kljukico pri *ASNW* (*Always Show New Window*) izbiramo, ali se ob meritvi v vsakem položaju odpre novo okno z njenim prikazom. Izbor je možno spreminjati dinamično med merjenjem.

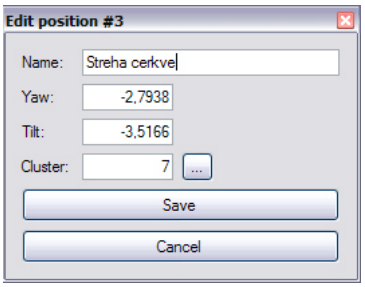
Najnaprednejša možnost panorame je možnost *Centralize*. Ko je aktivna, bo program samodejno poizkušal prilagoditi položaj naprave, da se slika iz kamere kar najbolj poravna

z vnaprej posneto sliko panorame. Obseg razlike med pravilnim položajem in shranjenim položajem sme biti največ za polovico zornih kotov kamere. (Ob optimizaciji modula za računalniški vid bi lahko območje zvečali.)

Kdaj bi možnost sploh lahko uporabili? Ne smemo pozabiti, da je naprava premična. Takoj, ko napravo premaknemo iz danega položaja, je precej verjetno, da ko jo postavimo nazaj, ne bo več natanko enako orientirana. Če smo pred premikom shranili položaje, bodo po premiku zaradi nenatančne postavitve netočni. Za razliko od orientacije s shranjenimi položaji, je orientacija s pomočjo slike vedno pravilna. Tako nam shranjeni položaji nudijo okvirno orientacijo, natančen položaj pa se določi s pomočjo primerjave shranjene slike s sliko na kameri. Uporabljajo se natanko isti algoritmi, kot pri kalibraciji kamere. Išče se korespondence med sliko kamere in po površini 4 krat večjo sliko, ki obkroža shranjen položaj. Če slika kamere leži znotraj slike okolice shranjenega položaja, se iz korespondenc lahko določi potreben premik naprave, da se sliki ujameta oz. zlijeta. Po premiku naprave nastopi meritev.

Delo s položaji na panorami (in v splošnem) ter z gručami

Pri dodajanju ali urejanju položaja se nam prikaže okence, ki ga vidimo na sliki (3.15). V primeru urejanja položajev v splošnem polje *Cluster* ne bo vidno. Pomen ima samo pri položajih na panorami.



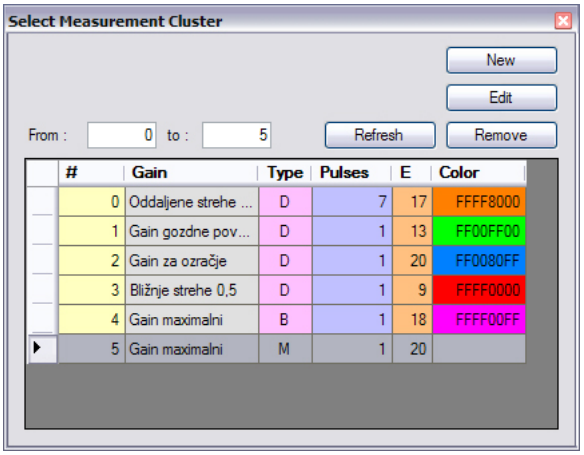
Slika 3.15: Urejanje podatkov merilnega položaja.

V prvo polje vnesemo ime položaja, v drugo odklon v kotnih stopinjah in v tretje nagib, tudi v kotnih stopinjah. Polje *Cluster* lahko pustimo prazno. V tem primeru položaj nima določene gruče na panorami. Na tak način lahko položaje odstranimo iz gruč, brez da bi jih morali zbrisati. V primeru, da želimo položaju prirediti gručo, v polje vpišemo zaporedno številko gruče, ki mora obstajati v bazi podatkov. Vse obstoječe gruče za izbrano panoramo prikažemo s pritiskom na gumb [...]. Gumb *Save* shrani položaj v bazo, *Cancel* prekliče vnose in/ali dodajanje novega področja.

Na sliki (3.16) je prikazano okno za izbiranje, dodajanje in urejanje gruč. *New* ustvari novo gručo in zanjo odpre okno za urejanje. Podobno kot *Edit*, ki odpre okno za urejanje izbrane gruče. *Remove* odstrani izbrane gruče, če le-te niso nikjer dodeljene. Prikaz gruč v tabeli je vedno omejen na dano panoramo in na vnos filtra od, do (*From, To*), ki narekuje interval indeksa gruč. *Refresh* osveži tabelo.

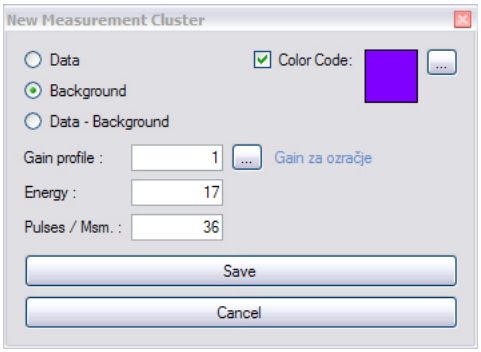
Na tabeli je zanimiv zadnji stolpec *Color*, ki prikazuje barvo gruče. S tako barvo bodo na panorami obarvana vsa področja, ki imajo dodeljeno dotično gručo. Ostali stolpci

prikazujejo podatke o gruči. Dvoklik na vrstico tabele ali pritisk na tipko *Enter* izbere označeno gručo in njen indeks vrne v polje, kjer smo želeli gručo izbrati.



Slika 3.16: Izbor gruče, kateri pripada področje merjnja.

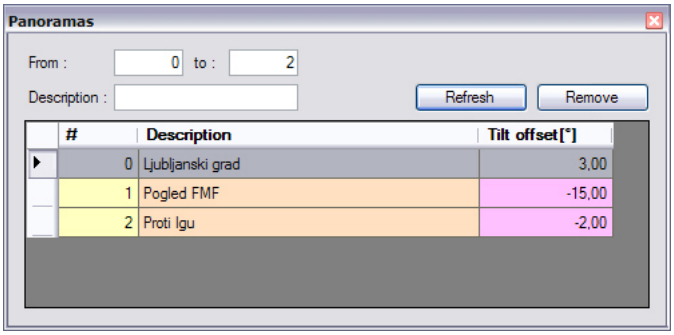
Slika (3.17) prikazuje urejanje podatkov gruče. Bodisi nove, bodisi obstoječe. Gruča položajem določa tip meritve (*Type*). Izbiramo lahko med *Data*, *D* – samo podatki, *Background*, *B* – ozadje ali *Data* – *Background*, *M* – razlika podatkov in ozadja. Določa še *gain* profil, energijo in število sunkov na meritev. Pomen je enak kot pri enostavni meritvi. S kljukico v polju *Color Code* gruči priredimo barvo, ki je prikazana poleg, v okvirčku. Barvo izberemo iz palete s klikom na gumb [...]. V primeru, da *Color Colde* ni odkljukano, gruča ne bo imela določene barve. *Save* shrani podatke o gruči, *Cancel* razveljavi vnose in/ali dodajanje nove gruče.



Slika 3.17: Urejanje podatkov o novi ali obstoječi gruči merilnih položajev.

Odpiranje panorame

Panoramo lahko odpremo preko okna za odpiranje panoram (slika 3.18) z dvoklikom na izbrano panoramo ali s tipko *Enter* na vrstici izbrane panorame. Na voljo imamo dva kriterija za prikaz panoram. Bodisi po indeksu od, do (*From, to*) ali po opisu *Description*. Gumb *Refresh* osveži tabelo panoram, gumb *Remove* odstrani izbrane panorame.

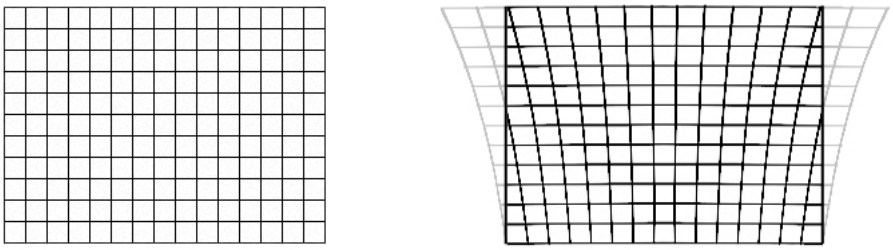


Slika 3.18: Okno za odpiranje panorame.

Panoramska slika je plašč valja s projiciranimi slikami kamere

Slika panorame je projekcija slike kamere na površino valja. Pri panoramah z velikim *Tilt Offset* zato pride do izkrivljanja slike. Vhodno sliko se po odpravi napak predela še po projekcjski enačbi (3.1). ((x, y) je točka na sliki kamere, (x', y') je točka na projekciji. Središči obeh slik sta $(0, 0)$, H je višina slike v pikslih, Fov_y je navpični zorni kot kamere in φ je nagib naprave.) Primer oz. shema projekcije je prikazana na sliki (3.19).

$$(x', y') = \left(x \cdot \cos\left(\frac{y}{H} \cdot Fov_y - \varphi\right), y\right) \tag{3.1}$$

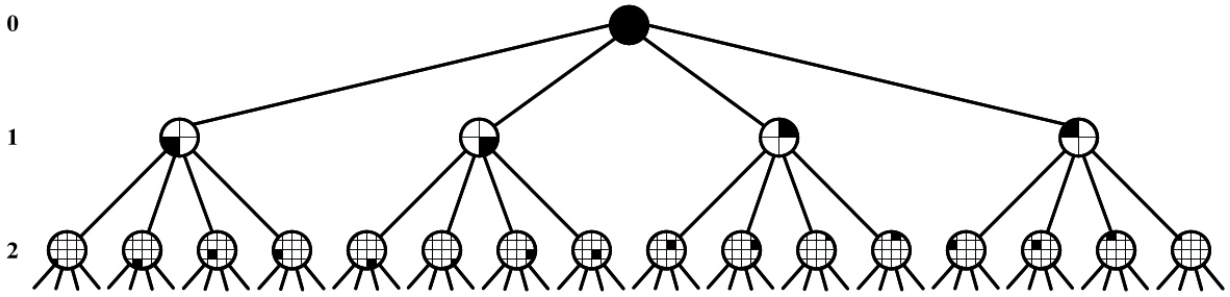


Slika 3.19: Projekcija slike iz kamere (**levo**) na valj - panoramo, (**desno**). Da bi dobili podobno situacijo bi moral biti *Tilt Offset* nastavljen na polovico vertikalnega zornega kota kamere in njen zorni kot okoli 45°. Sliko bi zajeli ravno pri nagibu *Tilt Offset*.

Shranjevanje panoramske slike

Shranjevanje panoramske slike v delovnem pomnilniku je realizirano rekurzivno, s pomočjo drevesne strukture, ki je prikazana na sliki (3.20). Prikazano drevo ima v trenutni različici programske opreme največjo globino 8. V njegovih listih se nahajajo bitne sličice, ki so nedeljive enote panoramske slike. Ko označujemo območje za pripravo panoramske slike, v bistvu določimo, v katere liste naj se shranijo slike iz kamere. Pri globini 8 je možno shraniti do 256x256 sličic, kar zadošča za celotno panoramsko sliko.

Struktura omogoča enostavno dodajanje in dostop do posameznih sličic v logaritemskem času. Pri tem velja omeniti, da drevo ni nujno popolno. Posamezne veje so lahko prazne. Tam panoramska slika poreprosto ni definirana in s tem prihranimo prostor.



Slika 3.20: Drevesna struktura za shranjevanje panoramske slike. Osenčeni deli vozlišč v drevesu predstavljajo dele slike, ki so dosegljivi iz danega vozlišča.

Shranjevanje v podatkovno bazo je realizirano nekoliko drugače. V bazi ne potrebujemo hitrega dostopa do posameznih sličic, zato se tja shranjujejo tabelarično, ena za drugo ter z dodanima koordinatama lege na panoramski sliki. Pri nalaganju sličic iz baze v pomnilnik se drevesna struktura ponovno zgradi.

3.4 Odpiranje meritev

3.4.1 Odpiranje osnovne meritve

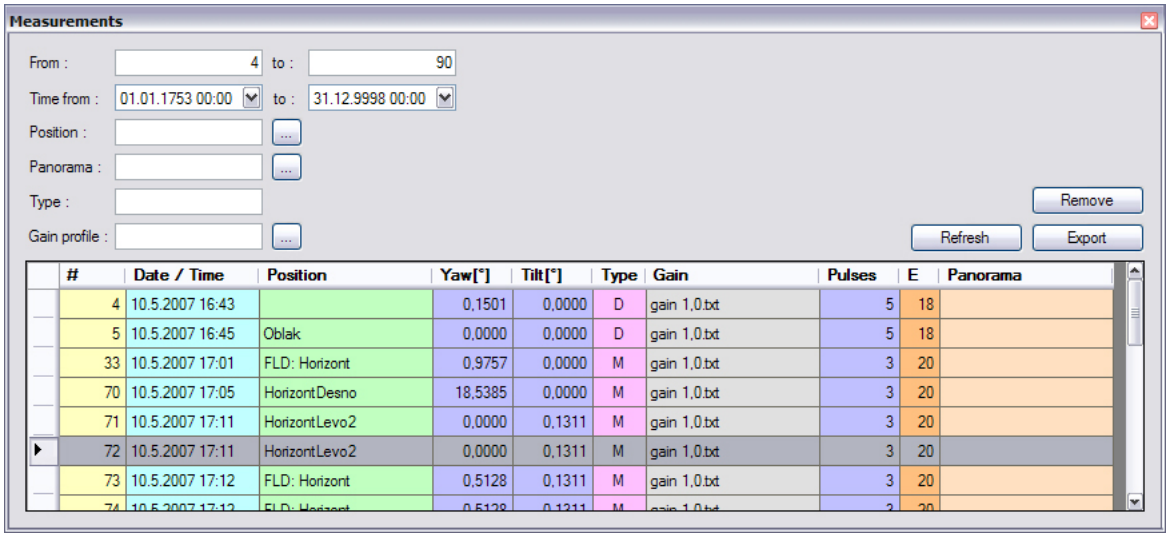
File : Open : Measurement...

Z uporabo okna na sliki (3.21) lahko odpremo katerokoli osnovno meritev, četudi je del meritve območja ali meritve v rezini. Izbor potrdimo z dvoklikom na izbrano meritev ali s pritiskom na tipko *Enter*, ko se nahajamo v tabeli.

Označene meritve lahko z gumbom *Export* izvozimo v datoteke, katerih ime se določi tekom izvažanja.

Pri pregledu meritev si lahko pomagamo z različnimi kriteriji, ki jih uveljavimo s pritiskom *Refresh*:

- Zaporedna številka meritve od, do (*From, to*).
- Časovna omejitev od, do (*Time from, to*).
- Zaporedna številka področja, v katerem je bila meritev opravljena (*Position*). (Z gumbom [...] lahko številko področja prikličemo iz baze.)
- Zaporedna številka panorame, na kateri je bila meritev opravljena (*Panorama*). (Z gumbom [...] lahko številko panorame prikličemo iz baze.)
- Izbiramo lahko med tipi (*Type*) meritev: podatki (D), ozadje (B), razlika podatkov in ozadja (M).
- Zaporedna številka *gain* profila, s katerim je bila meritev izvedena. Tudi tukaj lahko številko prikličemo iz baze s pritiskom na [...].



#	Date / Time	Position	Yaw[°]	Tilt[°]	Type	Gain	Pulses	E	Panorama
4	10.5.2007 16:43		0,1501	0,0000	D	gain 1,0.txt	5	18	
5	10.5.2007 16:45	Oblak	0,0000	0,0000	D	gain 1,0.txt	5	18	
33	10.5.2007 17:01	FLD: Horizont	0,9757	0,0000	M	gain 1,0.txt	3	20	
70	10.5.2007 17:05	HorizontDesno	18,5385	0,0000	M	gain 1,0.txt	3	20	
71	10.5.2007 17:11	HorizontLevo2	0,0000	0,1311	M	gain 1,0.txt	3	20	
72	10.5.2007 17:11	HorizontLevo2	0,0000	0,1311	M	gain 1,0.txt	3	20	
73	10.5.2007 17:12	FLD: Horizont	0,5128	0,1311	M	gain 1,0.txt	3	20	

Slika 3.21: Izbor osnovne meritve za vpogled.

Z gumbom *Remove* je možno odstraniti izbrane meritve.

Primer izvoza enostavne meritve v datoteko msm00000070.txt je v dodatku, razdelek A.2.2.

3.4.2 Odpiranje meritve območja

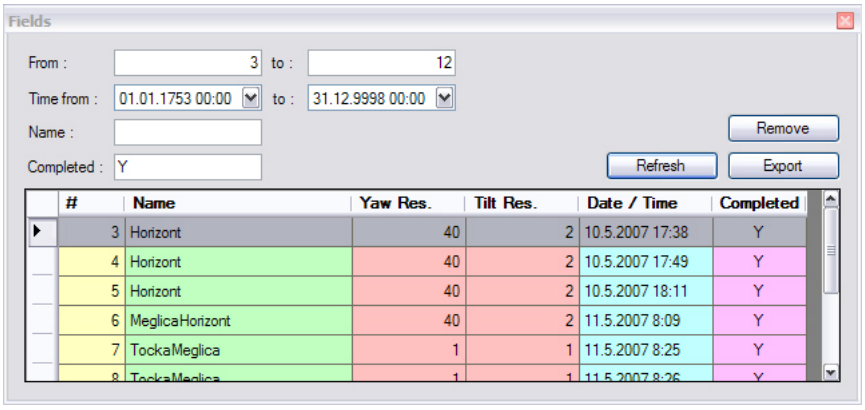
File : Open : Field...

Odpiranje meritev območja v trenutni različici programske opreme še ni mogoče. Uporabniku je na voljo le izvoz meritve področja, ki je dostopen preko okna, prikazanega na sliki (3.22). V tabeli označene meritve področja se izvozi s pritiskom na gumb *Export*. Glede na to, da je posamezna meritev področja sestavljena iz več enostavnih meritev so na voljo tri načini izvoza, ki so dostopni preko pogovornih oken v postopku izvoza. Vmes se določi tudi pot in ime tarčnih datotek. Možni izvozi so:

- Izvoz v eno samo datoteko skupaj s podatki pripadajočih enostavnih meritev.
- Izvoz v več datotek; ena za meritev področja in več datotek za pripadajoče enostavne meritve.
- Izvoz v eno samo datoteko brez pripadajočih osnovnih meritev.

Pri prikazu obstoječih meritev območij si lahko uporabnik pomaga s kriteriji izbora, ki jih potrdi z gumbom *Refresh*. Kriteriji so:

- Zaporedna številka meritve območja od, do (*From, to*).
- Časovna omejitev od, do (*Time from, to*).
- Izbor po imenu (*Name*).
- Izbor po uspešnem zaključku (*Completed*), ki je bodisi da (*Y*), in ne (*N*).



Slika 3.22: Izbor področja za odpiranje.

Z gumbom *Remove* je možno odstraniti izbrane meritve.

Primer izvoza meritve območja v datoteko fld0000000.txt je v dodatku, razdelek A.2.3.

3.4.3 Odpiranje meritve v rezini

File : Open : Slice...

Odpiranje meritev v rezini je uporabniku na voljo preko okna, prikazanega na sliki (3.23). Z dvoklikom na zapis v tabeli ali s tipko *Enter* se izbere meritev, ki se prikaže v novem oknu za prikaz meritve v rezini, (slika 3.12).

S pritiskom na gumb *Export* je v tabeli označene meritve možno izvoziti. Podobno kot pri meritvah območja so meritve v rezini sestavljene iz več enostavnih meritev. Na voljo so tri načini izvoza. Dostopni so preko pogovornih oken v postopku izvažanja. Pri tem se določi tudi pot in ime tarčnih datotek. Možni izvozi so:

- Izvoz v eno samo datoteko skupaj s podatki pripadajočih enostavnih meritev.
- Izvoz v več datotek; ena za meritev v rezini in več datotek za pripadajoče enostavne meritve.
- Izvoz v eno samo datoteko brez pripadajočih osnovnih meritev.

Pri prikazu obstoječih meritev v rezini si lahko uporabnik pomaga s kriteriji izbora, ki jih potrdi z gumbom *Refresh*. Kriteriji so popolnoma enaki kot pri meritvah območja.

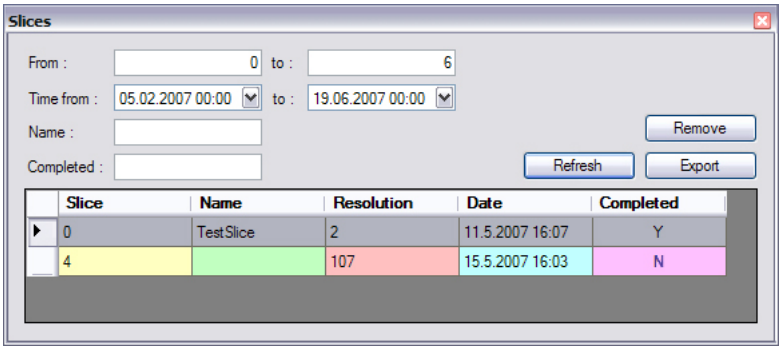
Z gumbom *Remove* je možno odstraniti izbrane meritve.

Primer izvoza meritve v rezini v datoteko slc00000000.txt je v dodatku, razdelek A.2.4.

3.5 Nastavitve lidarskega sistema

Tools : Device Settings

V nastavitvah naprav nastavimo vse potrebne parametre za pravilno delovanje aplikacije in naprave. Nastaviti moramo parametre napajalnika za laser, parametre senzorja, parametre gibalnega modula in parametre kamere.

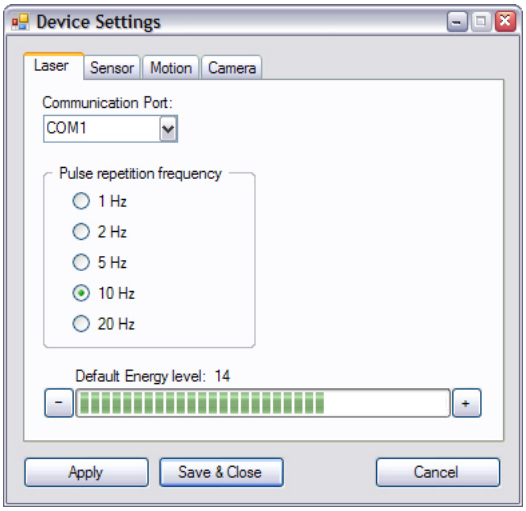


Slika 3.23: Izbor meritve v rezine za odpiranje.

Nastavitve, za razliko od ostalih podatkov v aplikaciji, niso shranjene v bazi podatkov. Shranjene so v posebni XML datoteki.

Okno za nastavitve ima mapo z zavihki. Vsakega za eno napravo (*Laser*, *Sensor*, *Motion* in *Camera*). Spodaj se nahajajo splošni gumbi *Apply*, *Save & Close* in *Cancel*. Gumbi imajo sledeče funkcije; prvi preveri ustreznost novih nastavitev, jih shrani in uveljavi. Drugi ima enako vlogo, vendar dodatno zapre nastavitve. Tretji zgolj prekliče vse spremembe nastavitev in zapre okno.

3.5.1 Nastavitve napajalnika za laser



Slika 3.24: Nastavitve napajalnika za laser.

Komunikacija z napajalnikom laserja poteka preko serijskih vrat računalnika, v okence *Communication Port* vnesemo pripadajoča COM vrata.

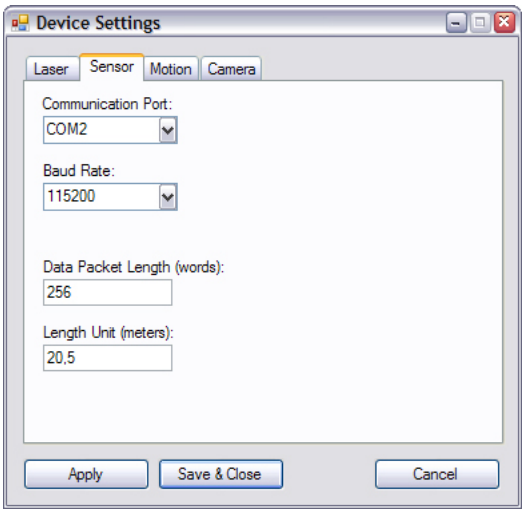
Nastavljamo lahko frekvenco proženja laserja *Pulse repetition frequency*. Ta parameter vpliva predvsem na trajanje merjenja.

Opozorilo: če merimo enostavne meritve in imamo vključeno možnost *Continuous mode*, z visoko frekvenco tvegamo hitrejše staranje žarnice laserja. Poleg tega v primeru

odpovedi aplikacije iz takega ali drugačnega razloga le-ta ne bo ustavila delovanja laserjevega napajalnika. Napajalnik laserja v avtonomnem delovanju, ki je značilno za *Continuous mode* način, ne zazna napake v aplikaciji in neovirano naprej proži laser. Edina možnost ustavitve je ročna ali s ponovno vzpostavitvijo aplikacije.

Pri vseh meritvah imamo okence za vnos količine energije. V ta namen lahko določimo privzeto vrednost, ki se nam v okencih vnaprej pojavlja.

3.5.2 Nastavitve senzorja



Slika 3.25: Nastavitve senzorja.

Tudi senzor z računalnikom komunicira prek serijskih vrat, ki jih izberemo v polju *Communication Port*.

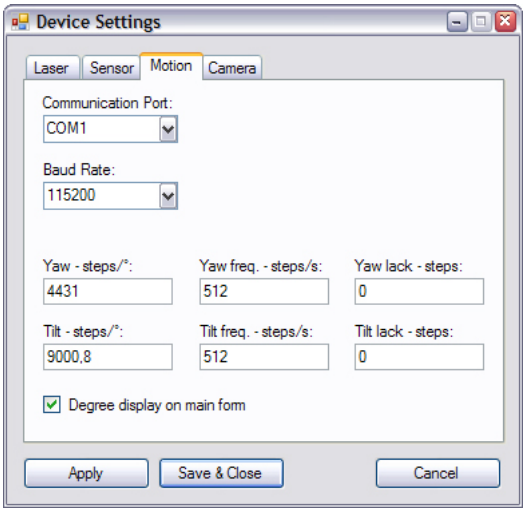
Glede na to, da je senzor prototip, obstaja možnost, da se bo v prihodnosti spremenilo hitrost komunikacije. V ta namen je hitrost (v bps) možno spreminjati v polju *Baud Rate*. Trenutno se uporablja vrednost 115200.

Polje *Data Packet Length (words)* naj bo nastavljeno na dolžino zlogov (1 zlog = 2 bajta), ki jih senzor pošlje po opravljeni meritvi. Obenem je tako dolg tudi paket, v katerem je shranjen *gain* profil, ki ga pošljemo senzorju pred začetkom merjenja.

Length Unit (meters), η polje je namenjeno vnosu dolžine enote (v metrih) dolžinske ločljivosti naprave. Tudi ta podatek, vključno z dolžino podatkovnega paketa je moč spreminjati iz razlogov morebitne spremembe na senzorju, vendar za razliko od dolžine paketa ne vpliva na delovanje naprave. Od njega so odvisni predvsem prikazi na diagramih in podatki v izvoženih datotekah. Določimo ga po enačbi (3.2), kjer je c svetlobna hitrost in $\nu_s$vzorčevalna frekvenca, ki je pri trenutnih pogojih enaka $7,37$ MHz. Podatek je tako $20,34$ m.

$$\eta = \frac{c}{2 \cdot \nu_s} \tag{3.2}$$

3.5.3 Nastavitve gibalnega modula



Slika 3.26: Nastavitve gibalnega modula naprave.

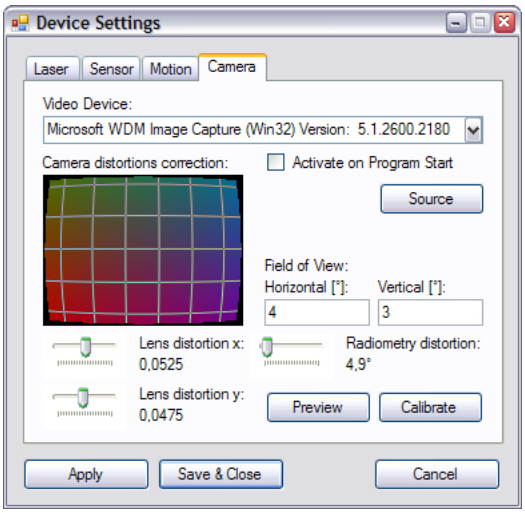
Gibalni modul naprave je bil zasnovan zelo splošno, zato tudi nastavitev ni malo. Podobno kot senzor tudi gibalni modul komunicira prek serijskih vrat. Izberemo jih v polju *Communication Port*. Nastavljamo lahko hitrost komunikacije v bps, *Baud Rate*. Modul trenutno komunicira pri 115200 bps.

Nastavitev *Yaw – steps/* $^\circ$ določa število korakov koračnega motorja za premik naprave v smeri odmika, horizontalno, za eno kotno stopinjo. Podobno *Tilt – steps/* $^\circ$ določa število korakov koračnega mtorja za premik v smeri nagiba naprave, vertikalno, pravtako za eno kotno stopinjo. Vrednosti najenostavneje odmerimo. Prikaz položaja na napravi spremenimo v koračni prikaz, nato s tipkovnico na napravi premikamo napravo toliko časa, da lahko z določeno natančnostjo odčitamo premik v stopinjah. Podatka izračunamo tako, da število korakov s prikazovalnika naprave delimo s številom stopinj premikov. Trenutno sta podatka 4431,0 in 9000,8 *korakov/* $^\circ$.

Yaw freq. – steps/s in *Tilt freq. – steps/s* sta najvišji frekvenci "korakanja" motorjev (korakov/sekundo) pri premikih. (Pospeška in pojemka motorjev v aplikaciji nista nastavljiva. Primerne vrednosti so trdno zakodirane v programu.) Za frekvenci želimo, da bi bili čim višji, saj je hitrost premikov naprave neposredno povezana z njima. Težava nastopi, ker ima motor omejen navor, ki kmalu postane prešibak za vrtenje pri visokih frekvencah. Izberemo taki frekvenci, ki zagotavljata zanesljivo vrtenje, brez izpuščenih korakov. Okvirna maksimalna vrednost pri naši napravi je 600 v smeri odklona in 500 v smeri nagiba.

Yaw lack – steps in *Tilt lack – steps* podatka služita za odpravo mehanske nepravilnosti pri obračanju naprave v dano točko iz različnih smeri. Poskrbita preprosto za to, da se vsak premik konča v isti smeri s toliko koraki, kolikor jih navedeno v ustrezni okenci. Predznak števila korakov določa smer. Za našo napravo vrednosti 200 zadoščata, sicer pa vrednosti toliko časa povečujemo, da je pri premikih – povratkih v dano točko napake čim manj, neglede na smer vrnitve.

3.5.4 Nastavitve kamere



Slika 3.27: Nastavitve kamere.

Pričujoči zavihek nastavitev kamere, slika (3.27), nas sprašuje za obilo parametrov. Temu je tako zaradi narave diplomske naloge, ki vključuje tudi računalniški vid.

Zgoraj se nahaja polje za izbiro gonilnika *web*-kamere, *Video Source*. Na večini sistemov bo edina možnost izbire le gonilnik *Microsoft WDM Image Capture*, ki jo izberemo.

Polje Activate on Program Start nam v aktivnem stanju omogoča avtomatski zagon kamere pri vsakem zagonu aplikacije, sicer moramo za to poskrbeti sami.

Gumb *Source* prikliče nastavitve izbranega gonilnika. V primeru, da kamera ni aktivna jo aktivira, saj dostop do gonilnika sicer ni mogoč. Več o tem v razdelku o nastavitvah gonilnika kamere.

Vnosni polji *Field of View* določata velikost zornega kota kamere v horizontalni in vertikalni smeri. Izražena sta v kotnih stopinjah. Odvisna sta od tipa kamere in iskalnega daljnogleda na napravi.

Slika 3.28: Primer odprave optičnih napak objektiva. **Levo**: slika brez popravkov. **Sredina**: odpravljeno popačenje leče. **Desno**: popravljena osvetlitev.

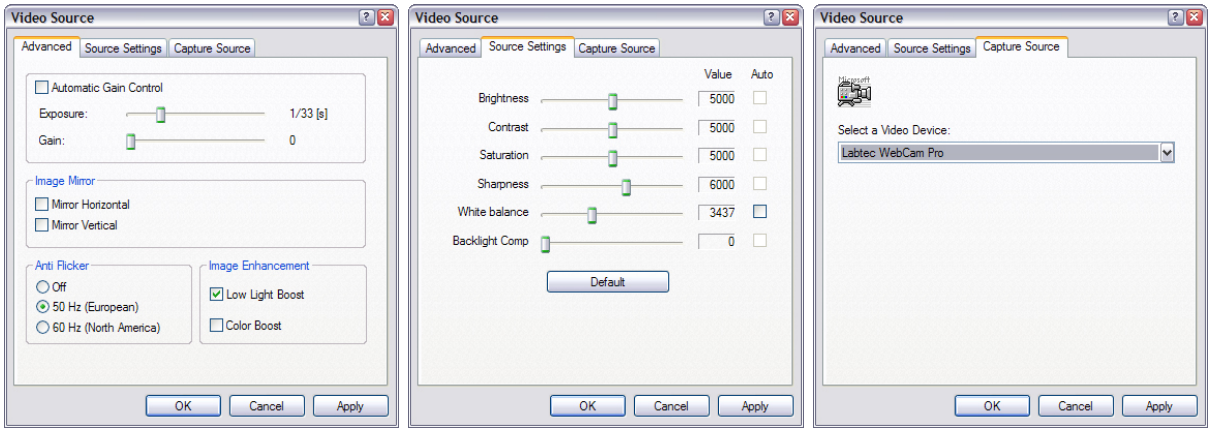
Najbolj v oči bode shematični prikaz za odpravo optične napake objektiva kamere, *Camera distortions correction*, ki služi zgolj za vizualizacijo učinka nastavitev *Lens distortion* x ter y in *Radiometry distortion*. Ti trije parametri poskrbijo za odpravo optičnih

napak, ki nastanejo v vsakem refraktorskem objektivu. *Lens distortion* parametra izničita učinek popačenja leče [17], *Radiometry distortion* pa problem neenakomerne osvetlitve. Več lahko bralec najde v [11]. Podrobnejši vpogled v temo pa sledi v poglavju 3.6.

Gumb *Preview* prikaže sliko iz kamere, popravljeno z danimi parametri. (Slika 3.28, desno.)

Gumb *Calibrate* odpre okno za avtomatsko nastavitev parametrov kamere.

Nastavitve gonilnika kamere



Slika 3.29: Nastavitve gonilnika kamere. **Levo**: napredne nastavitve. **Sredina**: nastavitve pogleda. **Desno**: izbira kamere.

Nastavitve gonilnika kamere obsegajo več nastavitev, ki jih lahko vidmo na sliki (3.29).

Opozorilo: nastavitve, ki jih naredimo v pričujočih oknih se shranjujejo posebej, zunaj aplikacije. Poleg tega so globalne za posamezno *web*-kamero, kar pomeni, da če isto kamero uporabljamo še kje drugje in njene nastavitve spremenimo tam, bodo vidne tudi tukaj.

Glede na to, da kamera služi pretežno za zajemanje panoramskih posnetkov in orientiranje, bomo navedli optimalne nastavitve pri teh pogojih delovanja.

Pomembna nastavitev pri kameri je *Automatic Gain Control*, ki mora biti za našo uporabo izklopljena. Gre za avtomatsko prilagajanje parametra *Gain*, ki uravnava intenziteto slike kamere. V primeru, da želimo posneti panoramski posnetek s skladno osvetlitvijo na vseh področjih, moramo zagotoviti, da se intenziteta slike ne spreminja. *Exposure* parameter določa čas osvetlitve senzorja v kameri. Svetlost slike je sorazmerna s časom osvetlitve. Želimo imeti čim svetlejšo sliko, vendar znotraj merilnega območja senzorja. Z drugimi besedami: noben del slike ni presvetljen. Ročna nastavitev *Gain* se ne shranjuje. Priporočeno je, da se ga ohrani na privzeti vrednosti 0.

Možnosti *Mirror Horizontal / Vertical* naj ostaneta izklopljeni. (Uporabljata se za zrcaljenje pogleda kamere.)

Možnosti *Anti Flicker* služijo pri odpravi motenj, ki bi nastale pri snemanju pod utripajočo svetlobo, ki se npr. pojavlja pri fluorescentnih svetilkah. Ker se naprava uporablja zunaj in podnevi, je nastavitev poljubna. Če bi napravo uporabljali pri javni cestni razsvetljavi, je pametno izbrati možnost *50Hz (European)*.

Možnosti *Image Enhancement* za izboljšavo slike sta dve: *Low Light Boost* uporabimo, če delamo npr. v mraku oz. pri šibkih osvetlitvah. *Color Boost* uporabljamo za izboljšavo barv. Nastavimo ga po občutku.

Source Settings ali nastavitve vira načeloma prilagodimo, da slika izgleda čim bolj naravna. Pozorni bodimo na to, da so pri vseh avtomatske nastavitve (*Auto*) izklopljene. Razlog je isti kot pri *Automatic Gain Control*.

Z možnostjo *Select Video Device* izberemo *web*-kamero lidarskega sistema. Do dileme utegne priti le v primeru, ko imamo na računalnik priključenih več *web*-kamer.

Okno za avtomatsko nastavitev parametrov kamere

Preden opišemo kalibracijski algoritem naj bralca opomnimo, da so v algoritmu uporabljena nekatera načela, ki so opisana v razdelku 3.6. Zaradi celovitosti opisa nastavitev lidarskega sistema algoritem navajamo na tem mestu.

Kalibracijski algoritem je sestavljen iz dveh faz:

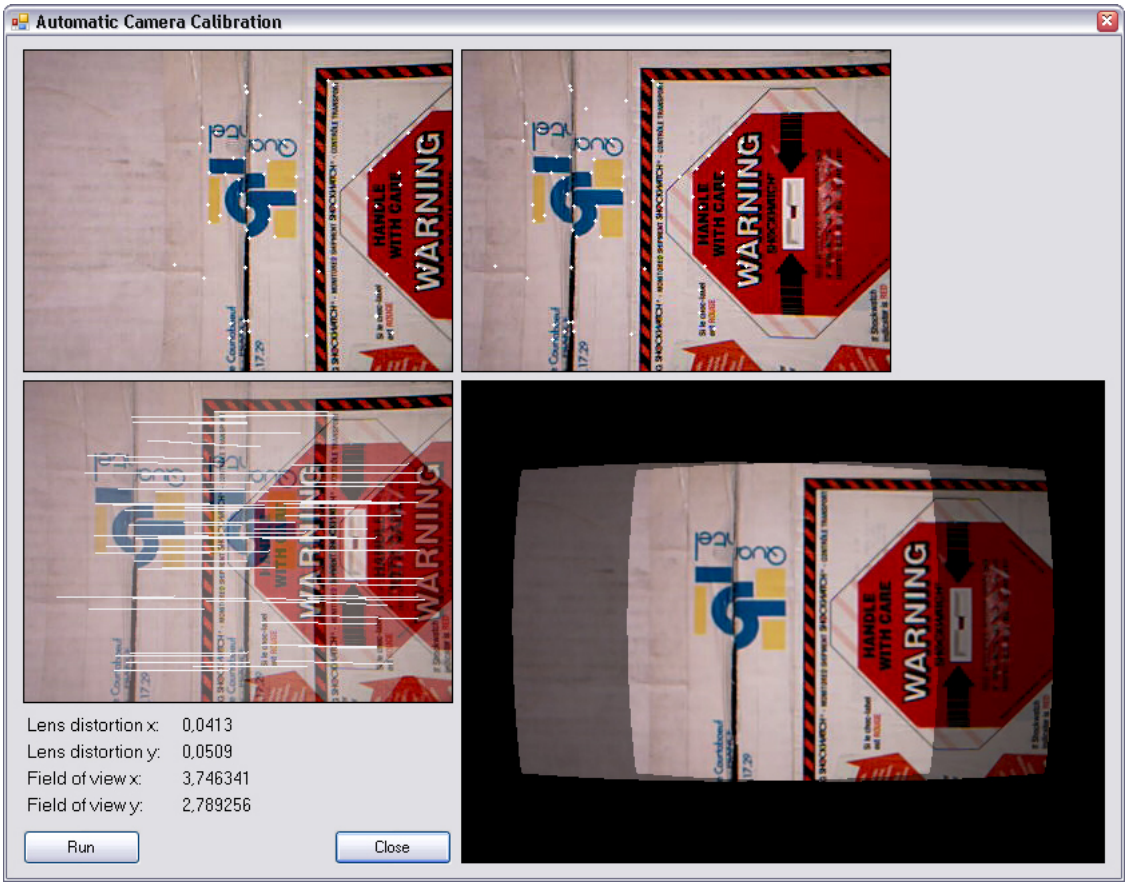
- Prva faza kalibracije - določitev parametrov *Lens distortion x / y*. Potek:

  1. Pri dveh različnih (in točno znanih) odklonih zajamemo sliki iz *web*-kamere. Razlika odklonov je približno ena tretjina horizontalnega zornega kota kamere. Na sliki (3.30) zgoraj levo in desno. Naklon naprave nastavimo tako, da je ta čimbolj vodoravno.
  2. Z uporabo algoritma za iskanje korespondenčnih točk para prekrivajočih se slik (razdelek 3.7) poiščemo korespondenčne točke na obeh slikah. Bele točke na sliki (3.30) zgoraj levo in desno so korespondence. Spodaj desno so prikazane povezave ustreznih korespondenčnih točk.
  3. *Lens distortion x* in *y* nastavimo na 0.
  4. Stari standardni deviaciji razlik koordinat parov korespondenčnih točk v smeri x in y nastavimo na neskončno.
  5. Koraka spremembe parametrov *Lens distortion x* in *y* nastavimo na neko primerno začetno vrednost $(0,01)$.
  6. Dokler ne presežemo maksimalnega števila iteracij (100) ali pa je kateri od korakov spremembe parametrov *Lens distortion* še prevelik (*>0,00001*):
     (a) S trenutnima *Lens distortion* koeficientoma iz originalnih vrednosti korespondenčnih točk na obeh slikah z uporabo enačb (3.3) izračunamo njihov novi položaj. Enačbi na (3.3) sta funkciji za generiranje radialne popačenosti slike. Parametra a_x in a_y sta v naši aplikaciji imenovana *Lens distortion x* in *Lens distortion y*.
     (b) Izračunamo novi standardni deviaciji razlik koordinat parov korespondenčnih točk v smeri x in y.
     (c) Če je nova standardna deviacija v smeri x večja od stare, nastavimo korak spremembe *Lens distortion x* na negativno vrednost polovice prejšnje vrednosti koraka in staro deviacijo nastavimo na vrednost nove.

(d) Če je nova standardna deviacija v smeri y večja od stare, nastavimo korak spremembe *Lens distortion y* na negativno vrednost polovice prejšnje vrednosti koraka in staro deviacijo nastavimo na vrednost nove.

(e) *Lens distortion x* povečamo za korak x

(f) *Lens distortion y* povečamo za korak y



Slika 3.30: Avtomatska kalibracija kamere - primer in rezultati.

7. Rezultat kalibracije parametrov *Lens distortion x* in *y* prikažemo v okence (slika 3.30) desno spodaj. Razmak med prikazanima slikama smo izračunali že kot stranski produkt pri točki 6 (b) in sicer kot povprečje razlik parov, z *Lens distortion x* in *y* popravljenih korespondenčnih točk. Pred prikazom slikama še odpravimo optične napake (razdelek 3.6).

8. Napravo postavimo v prvotno orientacijo.

Sledi druga faza - določitev parametrov zornih kotov kamere. Napravo pričakujemo v prvotnem položaju. Le-ta se torej nahaja v približno vodoravnem položaju, kar je pomembno tudi v drugi fazi.

Omenimo sedaj, da bi tekom prve faze enostavno izračunali tudi horizontalni zorni kot. V prvi fazi (točka 1) namreč natanko poznamo razliko odklonov pri katerih smo sliki zajeli. V predzadnjem koraku prve faze (točka 7) že tudi natančno vemo koliko pikslov

sta sliki narazen. (Zanima nas le horizontalna smer.) Iz razlike odklonov in razmika v pikslih pri znani horizontalni resoluciji kamere je izračun zornega kota trivialen. Dodatno bi morali postopek ponoviti še v vertikalni smeri, da bi določili še vertikalni zorni kot.

Določanje zornih kotov sicer poteka na podoben način, kot je opisano v delu [12]. Metodo smo namreč implementirali že nekoliko pred implementacijo prve faze kalibracije. Preizkusili smo jo ter jo zaradi primernosti rezultatov tudi ohranili. Po točnosti sta obe metodi primerljivi.

- Druga faza kalibracije - določitev parametrov zornih kotov kamere:

  1. V trenutnem (izhodiščnem) položaju naprave se zajame slika.
  2. V horizontalni smeri premaknemo napravo za nek minimalni odmik. Okvirno za 1/10 pričakovanega zornega kota in nikakor ne več kot za pol zornega kota.
  3. Slikama odpravimo optične napake (razdelek 3.6).
  4. V sredini prve slike zajamemo neko majhno kvadratno podsliko in jo na drugi sliki poskušamo smiselno umestiti. Z drugimi besedami: primerjamo jo z vsemi enako velikimi podslikami iz druge slike, zajetimi s središčem na vodoravni prečnici druge slike. Razlika v pikslih med središčem slike in področjem, ki daje najmanjšo različnost primerjave, predstavlja prvi približek zornega kota (glede na dani kot odmika in resolucijo kamere).
  5. Upoštevajoč izračunan približek zornega kota iz prejšnje točke premaknemo napravo še za toliko, da je po premiku glede na izhodiščni položaj obrnjena praktično za polovico zornega kota.
  6. Točko 4 ponovimo. Tokrat je napaka izračunanega zornega kota manjša, saj je relativni premik, kot podlaga za določanje zornega kota, tokrat največji oz. če smo natančni, malo manj kot polovico največjega. Seveda bi v primeru skoraj največjega kota morali jemati podsliko iz prve slike na skrajnem robu, s središčem na vodoravni prečnici. V implementaciji smo izboljšavo opustili.
  7. Napravo obrnemo v izhodiščni položaj.
  8. Identičen scenarij ponovimo še v vertikalni smeri, da določimo vertikalni zorni kot.

3.6 Odprava optičnih napak slike kamere

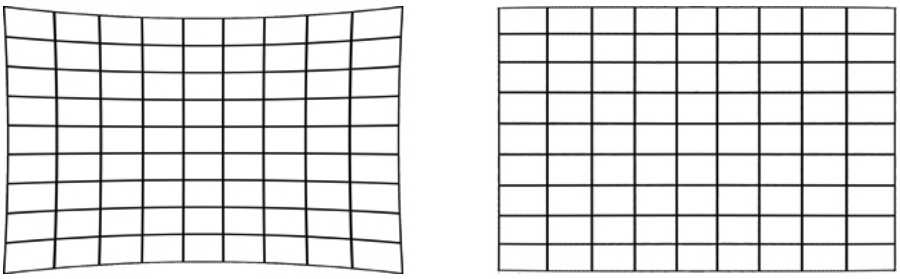
3.6.1 Popačenje leče

Znano je, da pri vsakem objektivu, ki uporablja leče, obstaja določeno radialno popačenje. Naša kamera ima relativno veliko radialno popačenje in zato si slike brez korekcije ne moremo privoščiti. Programsko je sliko enostavno popraviti. Na sliki (3.31) lahko vidimo shemi popačene in popravljene slike. Algoritem za izračun popravljene slike je sledeč; [17]: z uporabo prvega približka inverzne funkcije (3.3) za tvorjenje radialne popačenosti (3.4) za vsak piksel popravljene slike (P'_x,P'_y) določino izhodiščno točko (P_x,P_y) na popačeni izvorni sliki. Ker koordinati točke na izvorni sliki nista nujo celi števili, interpoliramo

vrednosti štirih okoliških sesednjih pikslov. Tako dobimo vrednost piksla popravljene slike. (Nato popravljeno sliko še ustrezno povečamo, da postane njena končna velikost prek sredine v horizontalni in vertikalni smeri enaka velikosti popačene slike. Natančneje naprej izračunamo faktor povečanja s in sproti iz popačene slike jemljemo piksle v okolici točke ($P_x \cdot s$, $P_y \cdot s$)).

$$P'_x = P_x \cdot \left(1 - a_x \cdot \|P\|^2\right) \quad P'_y = P_y \cdot \left(1 - a_y \cdot \|P\|^2\right) \quad P = (P_x, P_y) \tag{3.3}$$

$$P_x = \frac{P'_x}{1 - a_x \cdot \left\| \frac{P'}{1 - a_x \cdot \|P'\|^2} \right\|^2} \quad P_y = \frac{P'_y}{1 - a_y \cdot \left\| \frac{P'}{1 - a_y \cdot \|P'\|^2} \right\|^2} \quad P' = \left(P'_x, P'_y\right) \tag{3.4}$$



Slika 3.31: Slika iz kamere in odpravljanje njenega radialnega popačenja. **Levo**: slika iz kamere z radialnim popačenjem. **Desno**: popravljena slika.

3.6.2 Napaka radiometrije

Zaradi dejstva, da so robovi slike vedno slabše osvetljeni kot sredina, moramo pri sliki odpraviti tudi to težavo. Piksle vhodne nepravilno osvetljene slike $E(p)$ množimo s faktorji na desni strani enačbe (3.5). Enačba (3.5) je osnovna enačba radiometrične formacije slike, [11]. Povezuje osvetlitev slike v točki p na sliki, $E(p)$ in izsev scene (v točki P), ki jo slikamo, $L(P)$. Večji kot je kot α, kot med optično osjo in premico, ki povezuje točki p in P skozi gorišče objektiva, večja bo razlika. d je premer aperture objektiva, f njegova goriščna razdalja.

V naši aplikaciji se enačba (3.5) uporablja tako, da za $E(p)$ vzamemo intenziteto piksla nepopravljene slike, za intenziteto popravljenega piksla pa vzamemo $L(P)$. Skratka intenzitete R, G, B ustrezno množimo po enačbi (3.5). Kot α teče od 0 (v sredini slike) do *Radiometry distortion*, (v kotu slike). In sicer sorazmerno s tangensom razdalje, normirane na polovico diagonale slike, med sredino slike in pikslom, ki ga popravljamo. Za izračun faktorja (3.6) postopamo takole: izračunamo ga ob predpostavki, da na sredini slike ni korekcije. Z drugimi besedami: sredinski piksel nepopravljene slike se ne spremeni. V enačbi (3.5) sredini slike ustreza kot $\alpha = 0 \rightarrow L(P) = E(p)$.

V nastavitvah naše aplikacije maksimalno vrednost α neposredno nastavljamo prek *Radiometry distortion*, ki naj bi sicer ustrezal zornemu kotu prek diagonale slike, vendar

se zaradi nekoliko drugačne rabe enačbe (3.5) in drugih lastnosti kamere nekoliko razlikuje od njenega dejanskega zornega kota.

$$L(P) = E(p) \cdot \frac{4}{\pi} \cdot \left(\frac{f}{d}\right)^2 \cdot \cos^{-4}\alpha \tag{3.5}$$

$$\left(\frac{f}{d}\right)^2 = \frac{\pi}{4} \tag{3.6}$$

3.7 Algoritem za iskanje korespondenčnih točk para delno prekrivajočih se slik

Za potrebe kalibracije kamere, t.j. za določitev njenih parametrov popačenja leče in za določitev zornih kotov v navpični in vodoravni smeri ter za potrebe panorame smo implementirali algoritem, ki poišče korespondence med dvema slikama. Korespondenca je povezava med ustreznima točkama na paru slik. Pri tem na slikah dopuščamo določeno količino šuma. Prostostne stopnje za določanje korespondenc so tri: dve za translacijo in ena za rotacijo. Rotacija za našo aplikacijo nima koristne vloge, saj slike zajemamo pod istim kotom, a smo jo zaradi popolnosti algoritma vseeno vključili. Iz opisa algoritma bo bralec lahko sklepal, da s tem nismo izgubili veliko na časovni zahtevnosti, niti na natančnosti.

Končni cilj iskanja korespondenc je v določitvi vrednosti omenjenih prostostnih stopenj. Z vrednostmi namreč določimo kako moramo eno sliko premakniti in zavrteti, da se poravna s prvo.

3.7.1 Analiza slik s Harrisovim detektorjem značilnih točk, [13, 14]

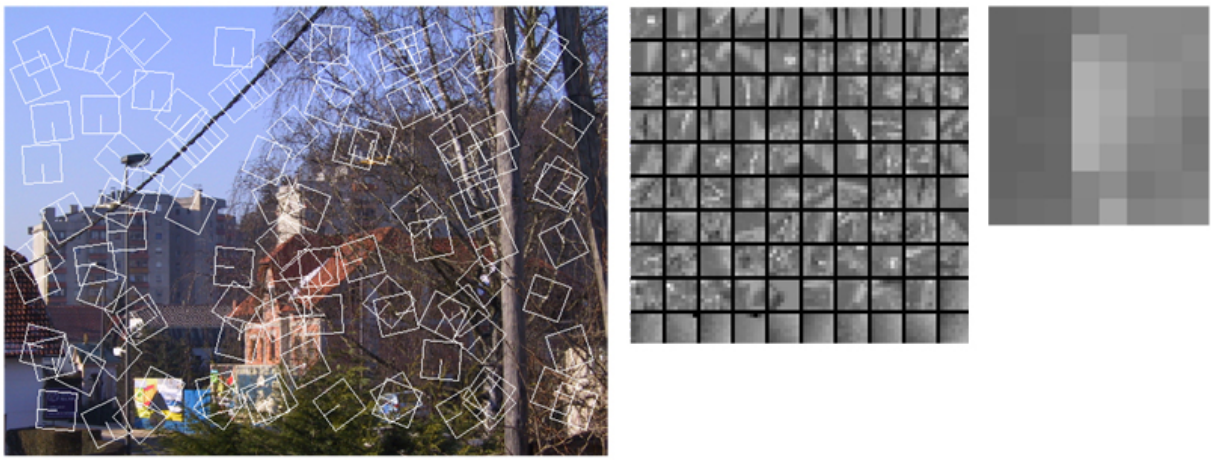
Slika 3.32: Postopek zaznavanja Harrisovih kotov. Slike **od leve proti desni** so: vhodna slika, intenzitetna slika, filtrirana intenzitetna slika, slika smeri gradientov filtrirane intenzitetne slike in slika jakosti Harrisovega detektorja filtrirane intenzitetne slike.

Vhodno sliko pretvorimo v intenzitetno sliko, ki jo nato filtriramo z nizko-prepustnim Gaussovim filtrom $d = 6$, σ=*1*. Filtriramo, da omilimo morebitne motnje, ki nastanejo

pri zajemanju slike na CCD senzorju v kameri. V nadaljevanju izračunamo smeri gradientov posameznih točk filtrirane intenzitetne slike, ki jih potrebujemo za izračun jakosti Harrisovega detektorja filtrirane intenzitetne slike in kasneje še za referenčno orientacijo pri usmerjenem zajemu zaplat. Postopek je prikazan na sliki 3.32. V naslednjem koraku izračunamo sliko jakosti Harrisovega detektorja filtrirane intenzitetne slike. Lokalni maksimumi na območju 3x3 piksle v sliki jakosti Harrisovega detektroja so t.i. Harrisovi koti. Harrisovi koti so "značilne točke" slike in nam služijo kot središča zaplat, ki jih, podvzorčene, zajamemo v nadaljevanju.

Pri implementaciji smo se opirali na knjižnjice z implementacijo v *Matlabu*, [15].

3.7.2 Zajem usmerjenih zaplat s središči v značilnih točkah

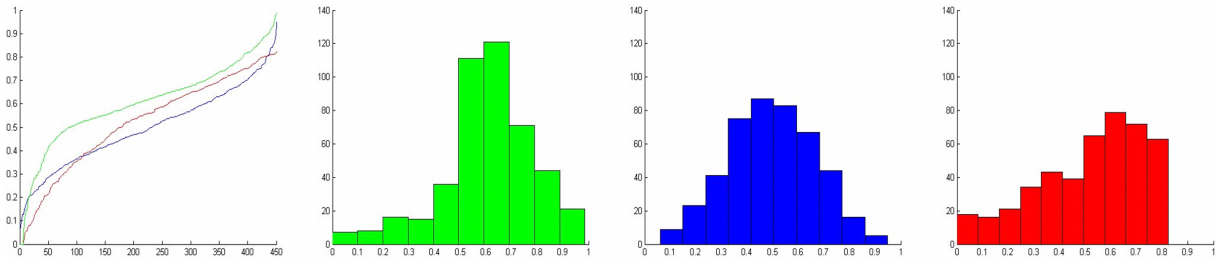


Slika 3.33: Zajemanje zaplat in tvorjenje opisnikov slike. **Levo**: območja kvadratov so zaplate s središči v Harrisovih kotih. Rotacija zaplat, t.j. kot črtice v kvadratu okoli njegovega središča glede na navpično smer je pogojena s smerjo prej izračunanega gradienta filtrirane intenzitetne slike v ustreznem Harrisovem kotu. **Sredina**: Podvzorčene usmerjene in normalizirane zaplate oz. opisniki razvrščeni po jakosti pripadajočih Harrisovih kotov. **Desno**: primer opisnika.

Z *adaptive non-maximal suppression* algoritmom, [13], izberemo k (≈ 1000) najprimernejših Harrisovih kotov. Dobimo neko podmnožico Harrisovih kotov, ki so enakomerno razporejeni prek vhodne oz. intenzitetne slike. Vsak izmed njih je izbrana značilna točka in predstavlja središče pripadajoče zaplate. Toda zaplat v nadaljevanju iz intenzitetne slike ne zajemamo neposredno. Razlog tokrat tiči v tem, da bi ob neposrednem zajemanju dobili prevelike zaplate (40x40). Intenzitetno sliko zato z ustreznim filtrom, (ki je navadno drugačen kot tisti pri zaznavanju Harrisovih kotov), ($d = 3$, σ *=1,5*) ponovno filtriramo. Iz filtrirane slike zajamemo zaplate, tako da pod kotom, skladnim s predhodno izračunano smerjo gradienta v pripadajočem Harrisovem kotu izberemo le vsak n-ti ($n = 5$) piksel. Dobimo podvzorčene in usmerjene zaplate. (Velikost zaplat je 8x8 = 64 vrednosti.) Vsako zaplato normaliziramo, da postane povprečna vrednost njenih pikslov $\overline{x} = 0$ in njihova standrardna deviacija $\sigma = 1$. Pod temi pogoji zaplatam pravimo opisniki, (slika 3.30, sredina). Pozor: barva pikslov ne predstavlja dejanske vrednosti in

je izbrana le za vizualizacijo.) Vsakemu opisniku izračunamo še tri neodvisne indeksne karakteristike s katerimi jih bomo kasneje razporedili v posebno primerjalno tabelo, ki pohitri iskanje korespondenc. Karakteristike so: povprečni gradient v smeri x, (zaradi poravnave opisnika ni popolnoma normalno porazdeljen), povprečni gradient v smeri Y ter vsota absolutnih vrednosti pozameznih pikslov. (Zaradi svoje narave zopet ni prav lepo normalno porazdeljena).

Dejanski primer porazdelitev indeksnih karakteristik je prikazan na sliki (3.34).



Slika 3.34: Konkretni primer porazdelitev indeksnih karakteristik s katerimi razporejamo opisnike v razporeditveno tabelo. **Levo**: kumulativni prikaz porazdelitev karakteristik. **Zelen**: povprečni gradient opisnika v prečni smeri. (Zaradi poravnave opisnika ima asimetrično porazdelitev). **Moder**: povprečni gradient opisnika v vzdolžni smeri. **Rdeč**: vsota absolutnih vrednosti posameznih pikslov opisnika. (Zaradi svoje narave tudi ta ni normalno porazdeljen).

3.7.3 Iskanje domnevnih korespondenc delno prekrivajočih se slik na podlagi ujemanja opisnikov

Ujemanje slik temelji na zadostnem in dovolj natančnem sovpadanju množic opisnikov dveh primerjujočih slik. Iskanje korespondenc med opisniki dveh slik je izvedeno prek primerjalne tabele, da ni potrebno primerjati vsak opisnik z vsakim. Izberemo tabelo primerne velikosti (10x10x10 košev) za obe sliki. Polnjenje tabele je realizirano prekrivajoče, kar pomeni, da se vsak opisnik, ki pripada centralnemu košu nahaja tudi v enem nad/pod, levo/desno ter pred/za njim. Verjetnost, da med primerjavo istoležnih košev zgrešimo ujemajoč par opisnikov, je manjša. Za indeksiranje opisnikov v primerjalni tabelo se uporabljajo tri neodvisne približno normalno porazdeljene karakteristike opisnikov, ki smo jih pripravili in izračunali že v prejšnji točki pri zajemu.

3.7.4 Izboljšava množice najdenih korespondenc opisnikov

Ko imamo domnevne korespondence zaženemo algoritem, ki izloči nepravilne. Korespondence med opisniki iz prejšnje točke namreč niso nujno pravilno določene. Primer nepravilnih je prikazan na sliki (3.35), spodaj levo. Korespondence, ki občutno izstopajo so napačne. Za njihovo izločitev uporabimo algoritem RANSAC (razdelek A.3), ki predvidi dober model korespondenc. Tiste, ki modelu ne ustrezajo dovolj dobro, zavržemo. Primer ustreznih korespondenc je na sliki (3.35), spodaj desno.

Slika 3.35: Primer izboljšave danih korespondenc z algoritmom RANSAC. **Levo zgoraj**: leva vhodna slika, **desno zgoraj**: desna vhodna slika. **Levo** in **desno spodaj**: prekrivajoči vhodni sliki ter prikaz ujemajočih značilk pred RANSAC izboljšavo in po njej.

Naš model sestavljajo trije parametri. Vsaka korespondenca povezuje dve točki. Prva se nahajata na eni, druga na drugi sliki. Zamislimo si sedaj dva koordinatna sistema. Točke iz prve slike postavimo v prvega, točke iz druge v drugega. Nato ustrezne pare točk povežemo z navideznimi elastikami ter sistem potopimo v viskozno tekočino. Sistem pustimo, da preide v stanje z najmanjšo energijo. Z drugimi besedami bi rekli, da elastike usmerijo koordinatna sistema tako, da se točke posameznih parov čimbolj zbližajo oz. da se elastike čimbolj skrčijo. Gibanje dovolimo le v ravnini kjer ležita koordinatna sisema. Parametra translacije in parameter rotacije drugega koordinatnega sistema glede na prvega so parametri opisanega modela.

Dejanske vrednosti parametrov izračunamo s simulacijo opisanega modela. Iterativno izračunavamo silo in navor elastik na težišče točk v drugem koordinatnem sistemu. Koordinatni sistem v vsaki iteraciji delno premaknemo in rotiramo v smeri sil in navora, dokler se rezultatni sil in navorov znatno manjšata in števec iteracij še ne preseže maksimalnega števila iteracij. Model postopoma preide v ravnovesje, ki je doseženo pri najnižji energiji elastik. Vrnemo parametre ravnovesnega stanja modela.

Kakovost prileganja modela množici korespondenc – elastik izračunamo s standardno deviacijo dolžin elastik v ravnovesnem stanju modela.

V zadnjem koraku moramo še izbrati korespondence, ki bodo nastopale v rešitvi oz. izločiti moramo nepravilne.

Korespondence – elastike, ki jih sprejmemo za pravilne, morajo ustrezati pogoju, da se njihova dolžina v ravnovesnem stanju najboljšega modela od njihove povprečne dolžine v ravnovesju najboljšega modela razlikuje za manj kot standardna deviacija dolžin elastik v

ravnovesju najboljšega modela. S tem ohranimo približno 2/3 korespondenc, ki so tvorile najboljši model.

3.7.5 Težave in možne izboljšave opisanega algoritma

Rezultati algoritma na testnih primerih so zadovoljivi, vendar se bo moral algoritem v praksi še potrditi. Algoritem je dokaj občutljiv na njegove parametre, ki jih ni malo. Na rezultate občutno vpliva število Harrisovih kotov, ki služijo za iskanje korespondenc. Več ko jih je, boljše je. Žal je tudi bolj potratno. Pomembeni parameteri so meje, s katerimi ločimo hipotetično pravilne korespondence od napačnih. Bodisi pri določanju ujemanja opisnikov, bodisi pri ujemanju dolžin "elastik". Parametre smo določili s poizkušanjem različnih vrednosti. Število iteracij v algoritmu RANSAC precej vpliva na rezultat. Število iteracij prilagajanja modela korespondencam nekoliko manj. Manj pomemben je tudi izbor parametrov filtrov pri detekciji Harrisovih kotov. Večji vpliv ima izbor parameterov filtra pri zajemu opisnikov in njihova velikost. Število košev pri ugotavljanju ujemanj opisnikov je boljše če je manjše, a v tem primeru se poveča časovna zahtevnost. Pri uporabljenih vrednostih rezultati praktično niso slabši od tistih pri uporabi enega samega koša. Iz opisa je razvidno, da je parametrov algoritma res veliko. Za optimalno delovanje bi zato vrjetno morali še precej testirati.

Uporabnik bo nemara opazil, da algoritem deluje sprejemljivo hitro, vendar vseeno dokaj počasi. Implementacija algoritma bi lahko bila boljša. Zaradi pomanjkanja časa smo precej naravnost implementirali algoritem in sicer po matematičnih zapisih iz literature. Pomanjklivosti smo dopolnili z algoritmi, ki so se zdeli primerni. Beseda leti predvsem na model pri algoritmu RANSAC. Za namen modela bi lahko uporabili Prokrustov algoritem in morda nekoliko povečali hitrost. Veliko operacij v algoritmu je matričnega tipa in zanje bi lahko napisali hitrejše funkcije. Predvsem funkcija konvolucije je kritična.

Možne izboljšave algoritma bi bile na primer večresolucijski pristop, t.j. s primerjavo opisnikov različnih velikosti. Dobra izboljšava algoritma bi nemara bila dosežena z upoštevanjem barvne informacije slik pri zajemu opisnikov in ne samo na podlagi intenzitetne slike. Pri algoritmu RANSAC bi lako dodali ponovno iskanje ujemanj med opisniki izločenih korespondenec in jih preizkusili na najdenem modelu in ga s tem še izboljšali. Z uporabo delne informacije o vrednostih parametrov modela bi lahko določili hipotetična mesta opisnikom, katerim nismo uspeli najti oz. pravilno določiti par. Iz košev za iskanje ujemanj bi obenem izločili opisnike, ki ležijo v predelu slik, ki se po trenutnih podatkih ne prekrivata. Nadalje bi lahko večkrat ponovili postopek 3.7.3, vendar le na množici opisnikov, ki na tem mestu domnevno ležijo v področju prekrivanja slik... Nenazadnje je metod za iskanje značilnih točk slik mnogo. Npr. s *Haar wavelet transformacijo*. Morda bi z uporabo katere druge metode dobili boljše rezultate...

3.8 Baza podatkov

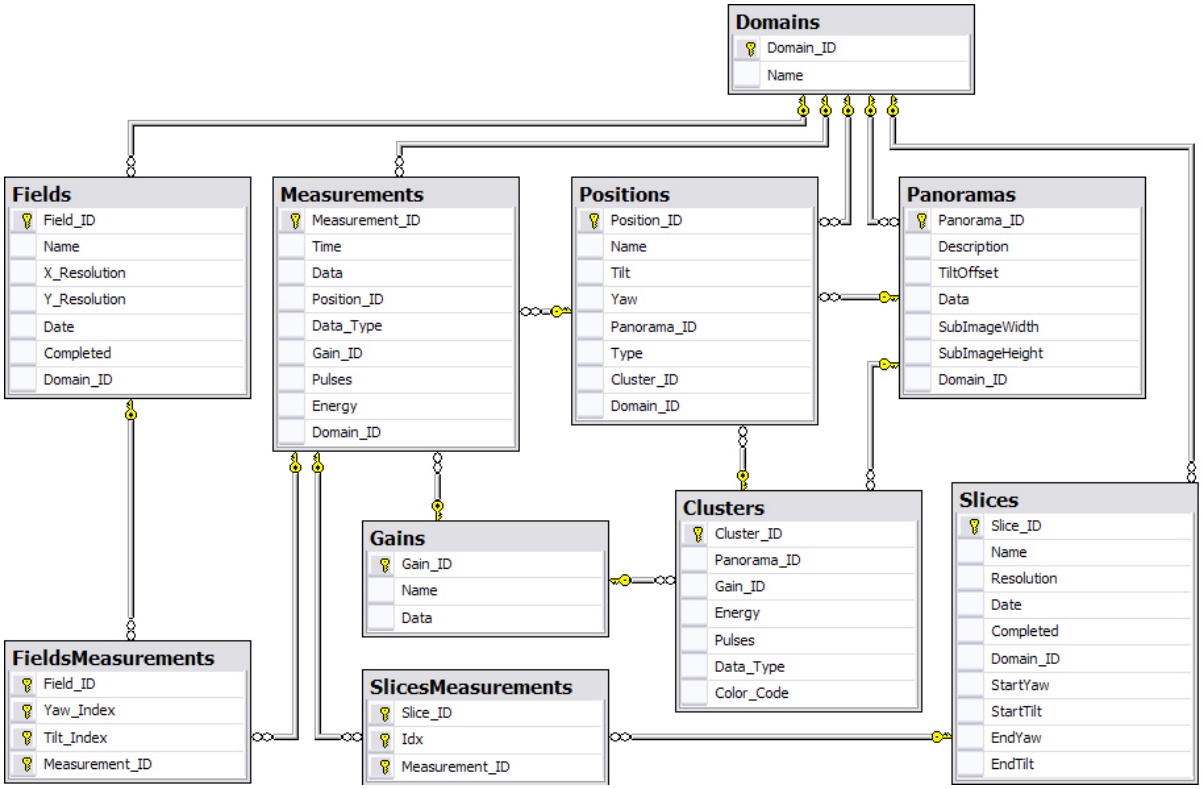
Prva zamisel shranjevanja podatkov v programski opremi je vključevala navadne datoteke, v katere bi se podatki shranjevali. Bodisi sproti, bodisi ko bi aplikacijo zaprli. Dejansko se na tak način shranjujejo nastavitve aplikacije. Le-te se ob uveljavitvi nastavitev shranijo

v nastavitveno XML datoteko. Druga odločitev je bila v prid podatkovni bazi. (Med drugim tudi zaradi celovitosti področja diplomske naloge.) Razen nastavitev se vsi podatki shranjujejo v podatkovno bazo.

Glede na količino in naravo podatkov se je druga odločitev tekom dela izkazala za pravilno. V naslednjem razdelku bomo videli, da imamo opravka s kar pestrim naborom podatkov.

Uporabili smo podatkovno bazo, ki temelji na strežniku *Microsoft SQL Server 2005 Express, database file*, [21]. Prednost izbrane podatkovne baze je v preprostosti, saj je celotna podatkovna baza shranjena v eni sami podatkovni datoteki, ki se nahaja v isti mapi kot zagonska datoteka aplikacije. Če želimo aplikacijo prenesti na drugo mesto, je vse kar potrebujemo, da poleg aplikacije premestimo še podatkovno datoteko. Druga prednost uporabe te podatkovne baze je v dostopnosti strežnika. Strežnik je prosto dostopen na internetu in njegova uporaba je brezplačna.

3.8.1 Opisi tabel podatkovne baze



Slika 3.36: Diagram tabel podatkovne baze s primarnimi ključi, označenimi s ključki na levi pred imeni polj tabele in tujimi ključi, označenimi s povezavami med tabelami. Povezave se nanašajo na istoimenska polja v povezanih parih tabel.

Na sliki (3.36) so prikazane podatkovne tabele in povezave med njimi:

- *Domains*; vsebuje imena domen (`varchar[100]`). Uporablja se za izbor aktivne domene, v kateri deluje aplikacija. Na vseh področjih v aplikaciji se vedno prikazu-

jejo le podatki, ki so vezani na izbrano domeno (`int`). To velja za podatke iz *Panoramas*, *Positions*, *Measurements*, *Fields* in *Slices*.

- *Measurements*; tukaj so podatki vseh osnovnih meritev: čas merjenja (`datetime`), podatki meritve (`image`), veza na položaj merjenja (neobvezno) (`int`), tip meritve (`varchar[1]`) ('M': podatki – ozadje, 'B': ozadje, 'D': podatki), veza na *gain* profil (`int`), ki je bil uporabljen (obvezno), število sunkov (`int`) in energija (`int`) s katerima je bila meritev izvedena ter domena (`int`) v okviru katere je bila meritev opravljena (obvezno).

- *Positions*; položaji meritev (`int`). Tabela vsebuje podatke o imenih položajev (`varchar[100]`), naklonu (`float`) in nagibu (`float`), vezo na panoramo (`int`) (neobvezno) v primeru, da se položaj nahaja na panorami, tip položaja (`varchar[1]`) ('A': avtomatsko zgeneriran pri merjenju, 'P': ročno shranjen v panorami, 'S': ročno shranjen v osnovnem oknu), povezava na gručo (`int`) (neobvezno), ki vsebuje parametre za avtomatsko meritev v tem položaju. Funkcija se uporablja pri merjenju v panorami. Ostane še povezava na domeno (`int`) (obvezna), kjer se položaj nahaja.

- *Gains*; tabela *gain* profilov (`int`), ki se uporabljajo pri meritvah. Tabela vsebuje ime profila (`varchar[100]`) in podatke profila (`image`).

- *Panoramas*; v tej tabeli so podatki posamezne panorame (`int`): opis panorame (`varchar[100]`), sredinski odmik nagiba od vodoravnega položaja (`float`), množica slik (`image`), ki tvorijo panoramsko sliko, višina (`int`) in širina (`int`) teh slik ter veza na domeno merjenja (`int`) (obvezna).

- *Clusters*; zapisi tabele gruč (`int`) so vezani na panoramo (`int`) (obvezno). Vsebujejo povezavo na *gain* profil (`int`) (obvezno), energijo (`smallint`), število sunkov (`int`) in tip meritve (`varchar[1]`) s katerim se meri na merilnih položajih ('M': podatki – ozadje, 'B': ozadje, 'D': podatki), ki uporabljajo dotično gručo. Barvna koda (`int`) vsebuje podatek o barvi. Z njo se na panorami obarvajo merilni položaji.

- *Fields*; tabela meritev v pravokotnem območju (`int`) vsebuje ime meritve (`varchar[100]`), število osnovnih meritev v vodoravni (`int`) in navpični smeri (`int`), čas pričetka merjenja (`datetime`), podatek o uspešnem zaključku merjenja (`varchar[1]`) ('Y': izmerjene vse meritve, 'N': nekatere meritve niso bile izvedene) in veza na domeno merjenja (`int`).

- *FieldsMeasurements*; vmesna tabela med tabelama *Fields* in *Measurements*. Veza na območje merjenja (*Field_ID*) (`int`) je obvezna, pravtako veza na meritev (`int`), indeksa nagiba (`int`) in naklona (`int`) podajata koordinati pripadajoče osnovne meritve v pravokotnem območju.

- *Slices*; tabela meritev v rezini (`int`). Vsebuje sledeče zapise: ime rezine (`varchar[100]`), število osnovnih meritev (`int`), ki tvorijo rezino, čas pričetka merjenja (`datetime`), poročilo o zaključku merjenja (`varchar[1]`) ('Y': izmerjene vse meritve, 'N': nekatere meritve niso bile izvedene), veza na domeno (`int`), v okviru katere

je bila rezina zajeta, začetni položaj merjenja *StartYaw* (`float`), *StartTilt* (`float`) ter končni položaj *EndYaw* (`float`), *EndTilt* (`float`).

- *SlicesMeasurements*; vmesna tabela med tabelama *Slices* in *Measurements*. Veza na območje merjenja (*Slice_ID*) (`int`) je obvezna. Obvezna je tudi veza na meritev (`int`). Indeks (`int`) predstavlja zaporeno številko pripadajoče osnovne meritve v rezini.

Poglavje 4

Sklepne ugotovite

Bralec je verjetno hitro ugotovil, da diplomska naloga vključuje veliko področij iz računalništva in informatike. Programiranje mikrokrmilnikov, povezovanje elementov v vezje – elektronika. Algoritmi iz računalniškega vida, problem trgovskega potnika, uporaba razvojnih orodij, razvoj komunikacijskega protokola prek RS-232. Grafika pri prikazu kontrol. Matematika in algebra pri razvoju kontrol in pri računalniškem vidu, fizika pri simulaciji dinamičnega sistema sil in navorov ter pri razvoju gibalnega modula. Podatkovne baze za shranjevanje podatkov. Načrtovanje pri izvedbi projekta...

Dejansko diplomska naloga zajema precej znanj iz računalniških in drugih področij. Kot taka je pritegnila pozornost. Manjši seminar na temo je bil že izveden in sicer v marcu 2007 na Fakulteti za matematiko in fiziko v Ljubljani. Sledila je še predstavitev na MCC konferenci v Dubrovniku na Hrvaškem, 15.6.2007.

Rezultati diplomske naloge so dobri. Razvoj gibalnega modula je zelo dobro uspel. Manjka sicer še nekaj lepotnih popravkov, vendar glede funkcionalnosti, optimizacije in ustreznosti popolnoma zadošča potrebam. Realizacija *floating point* pospeše-vanja in sprotnega poročanja položaja ter sprejemanje ukazov istočasno je soliden rezultat pri uporabi mikrokrmilnikov. Za informacijo navedimo, da je kode za gibalni modul okoli 1.500 programskih vrstic. Drugi del naloge, programska oprema za PC omogoča precej zanimivih uporab naprave. Predvsem modul za panorame izstopa zaradi večje kompleksnosti in uporabnosti. Modul za računalniški vid pri kalibraciji kamere deluje odlično. Nekoliko slabše se (trenutno) obnese pri delu s panoramami. V sodelovanju z uporabniki naprave so bile narejene še nekatere izboljšave, ki prvotno niso bile v planu. Programska oprema je postopoma dobila obliko zaključene celote in je zasnovana za uporabo in ne zgolj za demonstracijo. Navedimo še količino kode za programsko opremo. Le-ta obsega okvirno 12.000 programskih vrstic in je s tem največji projekt, ki sem ga dotlej uspel izvesti. Pričujoči dokument, poročilo diplomske naloge naj služi kot podlaga oz. priročnik za uporabo programske opreme in gibalnega modula. Stil pisanja je pretežno usmerjen h končnemu uporabniku, da bi mu čimbolj enostavno približali način uporabe.

Po prvotnih planih smo imeli namen vključiti še nekatere algoritme za analizo lidarskega signala, vendar sta že prva dva dela praktično presegla obseg povprečne diplomske naloge. Tega dela zato nismo vključili. Zasnova programske opreme sicer omogoča preprosto integracijo dodatnih modulov, ki bodo v bližnji prihodnosti verjetno dodani.

Uporaba programske opreme bo pokazala, kako dobra v resnici je. Kot avtor menim,

da bo vsaj približno zadovoljila uporabnika. Pri razvoju smo namreč izhajali tako iz stališča uporabnika, kot iz stališča enostavne implementacije.

4.1 Možnosti za nadaljni razvoj in plani

Preden govorimo o možnostih za nadaljni razvoj moramo omeniti, da obstoječa programska koda verjetno ni povsem pravilna. Za obseg pričujoče programske opreme bi bilo potreno narediti več testiranj, kot nam jih je čas dopuščal. V prvi meri zato v to točko spada odpravljanje napak in manjše posodobitve, ki jih bo potrebno narediti ob pojavitvi napak med uporabo. Nato sledijo izboljšave in večje dodelave.

Možnosti za nadaljni razvoj in izboljšave je v programski opremi ter v gibalnem modulu precej:

- Pri kalibraciji kamere je priročno dodati parameter rotacije. Njegova vpeljava je, upoštevajoč trenutni postopek kalibracije, dokaj enostavna. Vsekakor bi se s tem ognili precej mukotrpni nastavitvi kamere v vodoraven položaj. Trenutno je nastavljanje ročno.

- Optimizacija modula za računalniškli vid oz. za detekcijo značilk na slikah bi bila vsekakor dobrodošla. Implementacija tega modula trenutno ne vključuje optimizacij. Ne časovne, ne prostorske. Za hitrejše delovanje bi morali optimizirati npr. izračun konvolucije, ki se uporablja pri filtriranju slik z Gaussovim filtrom. Pri matričnem računanju pravtako ostaja časovna vrzel, ki jo je možmo precej enostavo premostiti. Pri optimizaciji bi bilo smiselno uporabiti kodo, namenjeno točno tarčnemu procesorju. .NET okolje, v katerem so trenutno implementirani algoritmi, ni ravno najhitrejša možnost.

- Pri panoramskih meritvah bi lahko že vnaprej izračunavali opisnike panoramske slike. Pri samodejni prilagoditvi položaja naprave namreč pri vsakem primerjanju panoramske slike s sliko na kameri pride do ponovnega izračunavanja opisnikov panoramske slike. Če bi imeli namesto panoramske slike v bazi shranjene le njene opisnike in njihove položaje, bi pri primerjanju prihranili na času, pri shranjevanju pa na prostoru. Na pomanjkljivost izboljšave bi naleteli, če bi se parametri algoritmov računalniškega vida spremenili, vendar so ti zaenkrat za uporabnika nespremenljivi.

- Če bi želeli izdelek ponuditi na trgu, je vsekakor potrebno izdelati načrt za tiskano vezje krmilnega modula, ki je zaenkrat na prototipni ploščici. Poleg tega bi bil verjetno potreben še razmislek v zvezi z vezjem samim, saj ne vemo, kakšni motorčki se nahajajo v trenutno razpoložljivih teleskopih, ki bi jih uporabili za predelavo v končni izdelek.

- V zvezi z uporabnikom ostaja še eno vprašanje. Katere možnosti pravzaprav ohraniti oz. dodati na prikazovalniku in tipkovnici pri gibalnem modulu? Morda bodo izkušnje pokazale, da sploh nista potrebna. Morda bo potrebno dodati kako novo funkcijo tipkovnice ali spremeniti njeno obliko.

- Nedvomno dobra izboljšava je dodatek tipala za izhodiščni položaj pri nagibu naprave. Trenutno ob vklopu gibalni modul nima nobene informacije o nagibu ali odklonu. Za odklon to pravzaprav ni pomembno, saj skrajna lega zanj ne obstaja. Za nagib tega ne moremo trditi. Pri uporabi koračnih motorjev je navada, da se v takem primeru doda tipalo. Ob vklopu se naprava samodejno premakne v skrajno lego, kjer se tipalo aktivira. S tem položaj postane znan.

- Znatno pohitritev bi dosegli z zamenjavo koračnih motorjev s hitrejšimi oz. močnejšimi. Trenutna hitrost premikanja naprave je razmeroma nizka. Pri zamenjavi velja omeniti, da bi pri gibalnem modulu morali zamenjati regulator napetosti, ker je sedanji prešibek.

- Pri programski opremi trenutno manjka grafični prikaz meritev v pravokotnem območju. Gotovo bi jo uporabnik z zadovoljstvom sprejel.

- Panoramske meritve se trenutno shranjujejo neposredno v tabelo enostavnih meritev. Dobro bi bilo dodati novo tabelo panoramskih meritev in vmesno tabelo. Za zgled je moč uporabiti primer shranjevanja meritev v rezini.

- Diagrami v programski opremi so solidno oblikovani, a jih žal še ni moč tiskati. Podpora tiskanja bi dodala veliko mero funkcionalnosti.

- Z zaključitvijo pričujočega diplomskega dela projekt še ne bo zaključen. Trenutno se delo že nadaljuje v okviru druge diplomske naloge. Gre za razvoj metod analize lidarskega signala in za opravljanje konkretnih meritev ter opredelitev rezultatov. Prvotno je bilo mišljeno, da bi omenjena področja pokrili v okviru tega diplomskega dela, a se je izkazalo, da bi s tem precej presegli obseg. Vsekakor je smiselno rezultate vključiti v programsko opremo.

- Lidarski sistem in podobna merilna oprema bo nameščena na prikolico ali v avto. S tem bo lidarski sistem postal del premične merilne postaje za določanje onesnaženosti zraka.

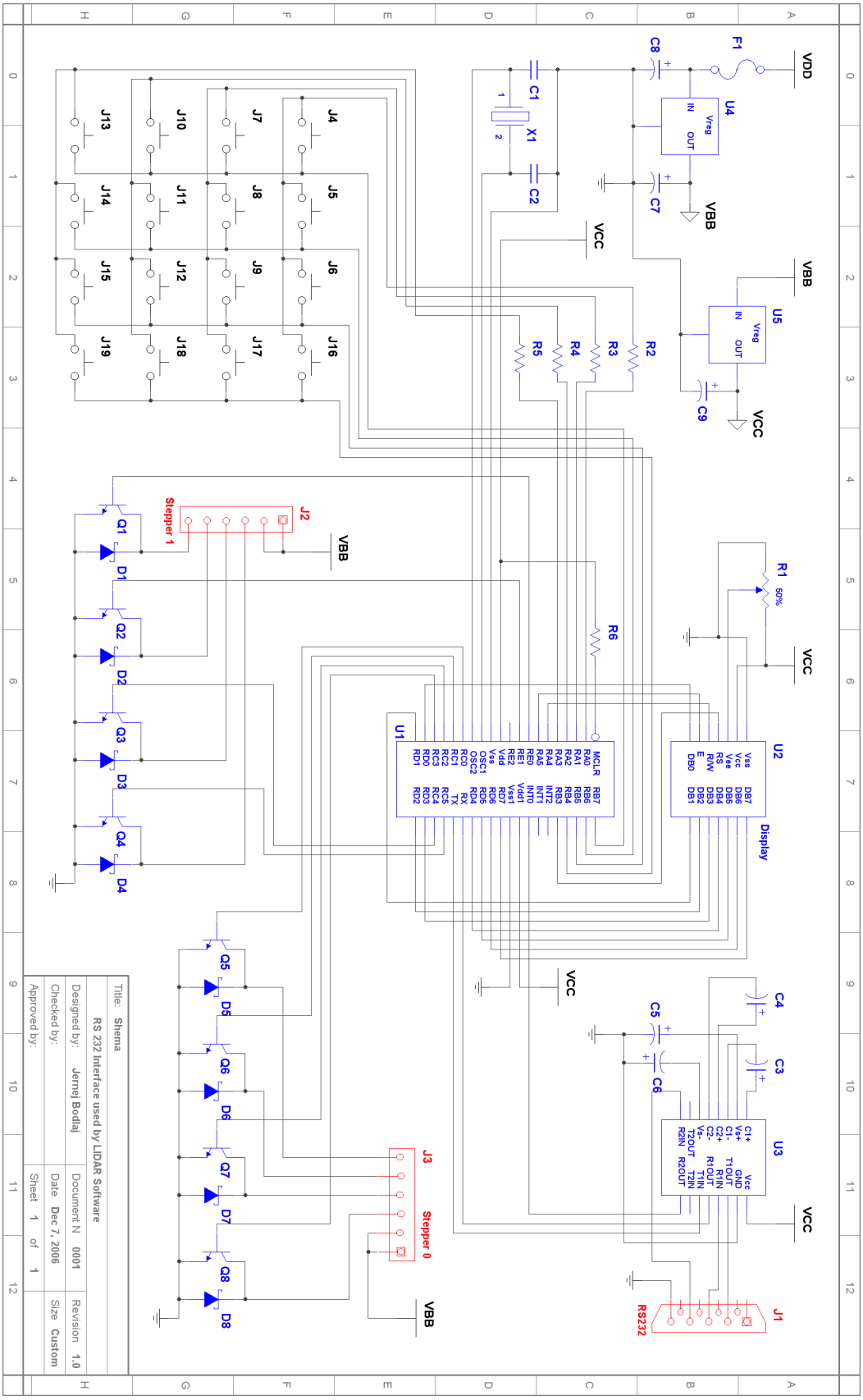
Dodatek A

Priloge

A.1 Shematični prikaz vezja gibalnega modula

A.1.1 Opombe k shemi (A.1)

1. VDD – napetost od 12 – 25V (Lahko vežemo direktno na akumulator v avtomobilu.)
2. VBB – napetost 10V – za napajanje koračnih motorjev.
3. VCC – napetost 5V – za napajanje logičnih komponent.
4. Uporniki R2 – R5 načeloma niso nujno potrebni. Igrajo le zaščitno vlogo v primeru, da uporabnik pritisne dve tipki v različnih vrsticah (na shemi) naenkrat.
5. Kristal X1 v sodelovanju s kondenzatorjema C1 in C2 tvori natančno vezje za generiranje urinega signala. Zaradi šibkega signala naj bo vezje neposredno ob nožicah U1.
6. Potenciometer R1 služi nastavitvi kontrasta prikazovalnika. Nastavi se ga pred prvo uporabo.
7. Pri konektorjih J2 in J3 je možna drugačna permutacija priključkov, kar zavisi od uporabljenih koračnih motorjev. VBB nožice morajo biti izbrane pravilno, neposrečeno izbiro kombinacije ostalih priključkov pa lahko popravimo programsko.
8. Diode, [8], D1 – D8 ščitijo tranzistorje Q1 – Q8 pred povratnimi napetostnimi sunki, ki se inducirajo v navitjih motorja ob njihovih odklopih.
9. Periferni kondenzatorji v okolici U3: C3 – C6 so vezani po specifikaciji vezja U3.
10. Upornik R6 služi kot *pull-up* upornik. Realizira preprosto ponastavitveno *reset* vezje za inicializacijo U1.
11. Poraba vezja skupaj s prikazovalnikom in brez motorjev je $\approx$ 1.75 W pri 12 V napajanju.



Slika A.1: Shema vezja gibalnega modula.

A.1.2 Seznam komponent vezja

Količina	Opis predmeta	Oznaka	Ohišje
4	RESISTOR, 1.5K	R5, R4, R3, R2	RES0.5
2	CONNECTOR	J3, J2	
2	CAP ELECTROLIT, 1000uF	C8, C7	ELKO10R5
1	VOLTAGE REGULATOR, LM7810CT	U4	TO-220
1	VOLTAGE REGULATOR, LM7805CT	U5	TO-220
1	FUSE, 1.5A - FAST	F1	
4	CAP ELECTROLIT, 1uF	C6, C5, C4, C3	ELKO5R5
1	IC, MAX232	U3	DIP-16
1	CONNECTOR	J1	
1	POTENTIOMETER, 20K LIN	R1	LIN POT
8	Transistor NPN, BD437	Q1 - Q8	TO-225AA
8	SCHOTTKY DIODE, 1N5818	D1 - D8	SMB
1	LCD 2x16, Hitachi 44780 comp.	U2	
2	CAPACITOR, Ceramic, 22pF	C2, C1	CAP1
1	CRYSTAL, HC-49/US 10MHz	X1	HC-49US
1	MICROCONTROLER, PIC18F458	U1	DIP-40vcx
16	MICROKEY	J4 - J19	

Tabela A.1: Komponente za vezje gibalnega modula.

Seznam komponent vezja gibalnega modula se nahaja v tebeli (A.1).

Opomba:

Zavedati se moramo, da smo izdelali prototip vezja. Nekatere komponente (žice, podnožja, vtičinice in vtikači, letvice ipd.) zato v tabeli niso navedene.

A.2 Primeri izvozov meritev v datoteke

A.2.1 Izvoz *gain* profila v datoteko **gn00000003.txt**

```
Name: Gain nizko bližnje
Profile
0,50790
0,45743
0,44479
   .
   .
0,78551
```

A.2.2 Izvoz enostavne meritve v datoteko **msm00000070.txt**

```
Measurement taken: 10.05.2007 17:05:06
```

```
Position: HorizontDesno
Yaw: 18,5384998321533
Tilt: 0
Panorama:
Type: Data - Background
Gain: gain 1,0.txt
Pulses: 3
Energy: 20
Distance Intensity
       0    0,282654
      20    0,142639
      41    0,102295
          .
          .
    5177    0,001099
```

A.2.3 Izvoz meritve območja v datoteko fld0000000.txt

- Način izvoza brez pripadajočih osnovnih meritev:

```
Name: Horizont
Yaw resolution: 40
Tilt resolution: 2
Field taken: 10.05.2007 17:49:01
Completed: Yes
Yaw  Tilt Measurement
   0    0 00000311
   1    0 00000313
   2    0 00000315
      .
      .
  39    0 00000389
   0    1 00000312
   1    1 00000314
   2    1 00000316
      .
      .
  39    1 00000390
```

- Način izvoza z ločenimi datotekami pripadajočih osnovnih meritev:

```
Name: Horizont
Yaw resolution: 40
Tilt resolution: 2
Field taken: 10.05.2007 17:49:01
```

```
Completed: Yes
Yaw  Tilt Measurement
   0    0 E:\Jernej Bodlaj\Desktop\test\msm00000311.txt
   1    0 E:\Jernej Bodlaj\Desktop\test\msm00000313.txt
     .
     .
  39    0 E:\Jernej Bodlaj\Desktop\test\msm00000389.txt
   0    1 E:\Jernej Bodlaj\Desktop\test\msm00000312.txt
     .
     .
  39    1 E:\Jernej Bodlaj\Desktop\test\msm00000390.txt
```

Datoteke msm00000311.txt, msm00000312.txt... so datoteke pripadajočih enostavnih meritev 311, 312... in so enake tistim, ki bi jih dobili pri ročnem izvozu.

- Način izvoza z integriranimi pripadajočimi osnovnimi meritvami v eno samo datoteko:

```
Name: Horizont
Yaw resolution: 40
Tilt resolution: 2
Field taken: 10.05.2007 17:49:01
Completed: Yes
Yaw  Tilt Measurement
   0    0
#begin
Measurement taken: 10.05.2007 17:49:03
Position: FLD: Horizont
Yaw: 0
Tilt: 0,52700001001358
Panorama:
Type: Data - Background
Gain: gain 1,0.txt
Pulses: 3
Energy: 20
Distance Intensity
       0    0,977844
         .
         .
    5177    -0,000305
#end
   1    0
#begin
Measurement taken: 10.05.2007 17:49:12
Position: FLD: Horizont
Yaw: 0,512820541858673
```

```
Tilt: 0,52700001001358
Panorama:
Type: Data - Background
Gain: gain 1,0.txt
Pulses: 3
Energy: 20
Distance Intensity
       0    0,767944
         .
         .
    5177    0,000244
#end
   2    0
#begin
...
#end
     .
     .
     .
   39    0
#begin
...
#end
    0    1
#begin
...
#end
     .
     .
     .
   39    1
#begin
...
#end
```

A.2.4 Izvoz meritve v rezini v datoteko **slc00000000.txt**

- Način izvoza brez pripadajočih osnovnih meritev:

```
Name: TestSlice
Resolution: 2
Slice taken: 11.05.2007 16:07:54
Completed: Yes
Idx. Measurement
   0 00000853
```

```
   1 00000854
```

- Način izvoza z ločenimi datotekami pripadajočih osnovnih meritev:

```
Name: TestSlice
Resolution: 2
Slice taken: 11.05.2007 16:07:54
Completed: Yes
Idx. Measurement
   0 E:\Jernej Bodlaj\Desktop\test\msm00000853.txt
   1 E:\Jernej Bodlaj\Desktop\test\msm00000854.txt
```

  Datoteki msm00000853.txt in msm00000854.txt sta datoteki pripadajočih enostavnih meritev 853 in 854 in sta enaki tistima, ki bi ju dobili z ročnim izvozom.

- Način izvoza z integriranimi pripadajočimi osnovnimi meritvami v eno samo datoteko:

```
Name: TestSlice
Resolution: 2
Slice taken: 11.05.2007 16:07:54
Completed: Yes
Idx. Measurement
   0
#begin
Measurement taken: 11.05.2007 16:08:08
Position: Oblak
Yaw: 0
Tilt: 0
Panorama:
Type: Data - Background
Gain: gain 1,0.txt
Pulses: 3
Energy: 20
Distance Intensity
       0    0,135132
         .
         .
    5177    0,001160
#end
   1
#begin
Measurement taken: 11.05.2007 16:08:21
Position: SLC: TestSlice
Yaw: 1,19879996776581
```

```
    Tilt: 0
    Panorama:
    Type: Data - Background
    Gain: gain 1,0.txt
    Pulses: 3
    Energy: 20
    Distance Intensity
           0    0,107910
             .
             .
        5177    0,000427
    #end
```

A.3 RANSAC algoritem, [23]

```
vhod:
  domnevne korespondence
  model, ki se lahko prilagodi korespondencam
  n - minimalno število pravilnih korespondenc za določitev parametrov
      modela
  k - maksimalno število iteracij algoritma
  t - prag napake, ki odloči ali dana korespondenca ustreza modelu
  d - število dodatnih kompatibilnih korespondenc, ki potrjujejo
      ustreznost modela

izhod:
  bestfit - parametri modela, ki najbolje opiše korespondence (ali null,
            če dobrega modela ni moč najti)

iterations := 0
bestfit := nil
besterr := infinity
while iterations < k
  maybeinliers := n naključno izbranih korespondenc
  maybemodel := parametri modela, prilagojenega maybeinliers
  alsoinliers := prazna množica

  za vsako korespondenco, ki ni v maybeinliers
    če korespondenca ustreza maybemodel z napako manjšo od t
      dodaj korespondenco v alsoinliers

    če je število korespondenc v alsoinliers > d
      (to pomeni, da smo našli dober model, ocenimo mero njegove
       ustreznosi)
```

```
      bettermodel := parametri modela, prilagojenega korespondencam v
                     maybeinliers in alsoinliers
      thiserr := mera ustreznosti modela tem korespondencam
      če thiserr < besterr
        bestfit := bettermodel
        besterr := thiserr

  povečaj iterations

vrni bestfit
```

Slike

Tabele

Literatura

[1] BIG SKY LASER Technologies, Inc., "*ULTRA CFR Nd:YAG Laser System: User's Manual,*" 1999.

[2] Fairchild Semicondustor Corporation, "*KA78XX / KA78XXA 3-Terminal 1A Positive Voltage Regulator,*" 2001.

[3] Hitachi, Ltd., "*HD44780U (LCD-II) (Dot Matrix Liquid Crystal Display Controller/Driver),*" 1998.

[4] Microchip Technologies Inc., "*MPLAB® C18 C COMPILER LIBRARIES,*" 2005.

[5] Microchip Technologies Inc., "*MPLAB® C18 C COMPILER USER'S GUIDE,*" 2005.

[6] Microchip Technologies Inc., "*PIC18FXX8 Data Sheet: 28/40-Pin High-Performance, Enhanced Flash Microcontrollers with CAN Module,*" 2004.

[7] Motorola Inc., "*BD437 BD441 Plastic Medium Power NPN Transistor,*" 1995.

[8] Phillips Semiconductors, "*1N5817; 1N5818; 1N5819 Schottky barrier diodes, Data Sheet,*" 1996.

[9] Texas Instruments Inc., "*MAX232, MAX232I Dual EIA-232 Drivers/Receivers,*" 2005.

[10] T. Tuma, "*Mikrokrmilniški sistemi: programiranje za družino HC11,*" Ljubljana: Fakulteta za elektrotehniko, 2000, str. 13–15.

[11] E. Trucco, A. Verri, "*Introductory Tehniques for 3-D Computer Vision,*" Prentice Hall, Upper Saddle River, New Jersy 07458, 1998, str. 22–26.

[12] B. Prihavec, F. Solina, "*User interface for video observation over the internet,*" Journal of Network and Computer Applications, št. 21, 1998, str. 219–237.

[13] M. Brown, R. Szeliski, S. Winder, "*Multi-Image Matching using Multi-Scale Oriented Patches,*" International Conference on Computer Vision and Pattern Recognition, 2005, str. 510–517.

[14] Konstantinos G. Derpanis, "*The Harris Corner Detection,*" 2004.

[15] P. H. S. Torr, "*A Structure and Motion Toolkit in Matlab,*" Technical report, 2002.

[16] (1995) D. W. Jones, "*Control of Stepping Motors (A Tutorial).*" Dostopno na: http://www.cs.uiowa.edu/∼jones/step/

[17] (2002) P. Bourke, "*Lens Correction and Distortion.*" Dostopno na: http://local.wasp.uwa.edu.au/∼pbourke/projection/lenscorrection/

[18] (2006) Olimex Ltd., "*PIC-PG2 – Serial port programmer.*" Dostopno na: http://www.olimex.com/dev/pic-pg2.html

[19] (2007) G. Bonny, "*IC-Prog Prototype Programmer.*" Dostopno na: http://www.ic-prog.com/index1.htm

[20] (2007) Microsoft Corporation, "*MSDN. (Windows multimedia, video capture.)*" Dostopno na: http://msdn.microsoft.com/library/en-us/multimed/htm/_win32_video_capture.asp

[21] (2007) Microsoft Corporation, "*Microsoft SQL Server 2005 Express.*" Dostopno na: http://www.microsoft.com/sql/editions/express/default.mspx

[22] (2007) Get Wired!, "*Handbook of hardware pinouts, cables schemes and connectors layouts.*" Dostopno na: http://pinouts.ru/SerialPorts/Serial9_pinout.shtml

[23] (2007) "*RANSAC algorithm.*" Dostopno na: http://en.wikipedia.org/wiki/RANSAC

Izjava

Izjavljam, da sem diplomsko delo izdelal samostojno pod vodstvom mentorja prof. dr. Franca Soline in somentorja prof. dr. Tomaža Pisanskega. Izkazano pomoč drugih sodelavcev sem v celoti navedel v zahvali.

Ljubljana, 1. julij, 2007 Jernej Bodlaj