Automatska kalibracija 3D vizijskog sustava laser-kamera

Mužar, Matija

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Matija Mužar

Zagreb, 2023.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentori:

Doc. dr. sc. Marko Švaco, mag. ing. mech.

Doc. dr. sc. Filip Šuligoj, mag. ing. mech.

Student:

Matija Mužar

Zagreb, 2023.

Izjavljujem da sam ovaj rad izradio samostalno koristeći znanja stečena tijekom studija te navedenu literaturu.

Zahvaljujem se mentoru dr. sc. Marku Švaci na zadavanju teme završnog rada te doc. dr. Dragi Bračunu za demonstraciju principa rada sustava i samog postupka kalibracije. Također se zahvaljujem mentoru doc. dr. sc. Filipu Šuligoju na stručnim savjetima, dobrome usmjerenju te svoj pruženoj pomoći tijekom izrade ovog rada.

Naposljetku, želim izraziti duboku zahvalnost svojoj obitelji, prijateljima i vrijednim kolegama na neizmjernoj podršci i inspiraciji koju su mi pružili tijekom studiranja.

Matija Mužar

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za završne i diplomske ispite studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment, inženjerstvo materijala i mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa: **602 – 04 / 23 – 6 / 1**	
Ur.broj: **15 - 1703 - 23 -**	

ZAVRŠNI ZADATAK

Student: **Matija Mužar** JMBAG: **0035226276**

Naslov rada na hrvatskom jeziku: **Automatska kalibracija 3D vizijskog sustava laser-kamera**

Naslov rada na engleskom jeziku: **Automatic calibration of the laser-camera 3D vision system**

Opis zadatka:

Lasersko skeniranje je proces snimanja preciznih, trodimenzionalnih informacija stvarnog objekta, grupe objekata ili okoline, korištenjem lasera kao izvora svjetlosti. Kako bi se projiciranjem laserskog svjetla na objekt i snimanjem istog s kamerom mogao stvoriti precizno izmjereni 3D oblak točaka potrebno je napraviti ekstrinzičnu kalibraciju kamere i lasera. Zadatak završnog rada je automatizirati postupak kalibracije u kojem vizijski sustav lokalizira referentnu značajku poznatog objekta dok robotska ruka postavlja vizijski sustav u veliki broj različitih pozicija. Prema tome u sklopu rješenja potrebno je:

- Uskladiti komunikaciju između robotske ruke i vizijskog sustava (na temelju postojećeg rješenja),
- primjenom algoritama strojnog vida razviti rješenje za robusnu i automatsku lokalizaciju kalibracijskog objekta te zapis podataka,
- nakon lokalizacije objekta u nizu različitih pozicija, pronaći greške (outliere) i automatski ih izbaciti iz kalibracijskog seta podataka,
- testirati uspješnost kalibracije pomoću već razvijenog programa za kalibraciju koji se temelji na neuronskim mrežama (te prebaciti postojeće rješenje iz Matlaba u Python/c++ kod).

U radu je potrebno navesti korištenu literaturu i eventualno dobivenu pomoć.

Zadatak zadan:	Datum predaje rada:	Predviđeni datumi obrane:
30. 11. 2022.	**1. rok:** 20. 2. 2023. **2. rok (izvanredni):** 10. 7. 2023. **3. rok:** 18. 9. 2023.	**1. rok:** 27. 2. – 3. 3. 2023. **2. rok (izvanredni):** 14. 7. 2023. **3. rok:** 25. 9. – 29. 9. 2023.

Zadatak zadao:

Doc. dr. sc. Marko Švaco

Dr. sc. Filip Šuligoj

Predsjednik Povjerenstva:

Prof. dr. sc. Branko Bauer

SADRŽAJ

POPIS SLIKA

SAŽETAK

U okviru ovog završnog rada, detaljno je razrađen algoritam za lokalizaciju kalibracijskog objekta koristeći podatke dobivene iz 3D vizijskog sustava. Proces prikupljanja podataka ostvaruje se korištenjem gotovog postupka koji uključuje suradnju robota Fanuc M710 i integriranog sustava (kamera-laser) gdje se u procesu kalibracije sustava robot sa vizijskim sustavom postavlja u niz poznatih pozicija u odnosu na kalibracijsku ploču.

Sama bit algoritma leži u preciznom identificiranju koordinata bridova koji pripadaju kalibracijskoj ploči koje se koriste za kalibraciju sustava laser-kamera kako bi se pozicije lasera detektirane s kamerom mogle mapirati u 3D prostoru točaka.

Prije nego se te koordinate upotrijebe za kalibraciju sustava primjenjuje se RANSAC algoritam uz pomoć kojeg se detektiraju i eliminiraju potencijalne greške i nepreciznosti u skupu podataka čime se poboljšava preciznost kalibracije. Nakon primjene RANSAC algoritma slijedi obrada podataka putem unaprijed razvijene neuronske mreže koja ima zadatak testirati uspješnost kalibracije, ocjenjujući koliko precizno sustav locira kalibracijski objekt u stvarnome svijetu. Točnost kalibracije ključna je za pouzdanost i preciznost svih kasnijih mjerenja i zadataka koje sustav obavlja. Precizna kalibracija omogućava da roboti i sustavi za obradu slika točno razumiju svoje okruženje i izvršavaju zadatke s minimalnim ili nikakvim greškama.

Ključne riječi: 3D-vizijski sustav laser kamera, kalibracija, Python, RANSAC algoritam, neuronska mreža.

SUMMARY

Within this bachelor's thesis, an algorithm for localizing the calibration object using data obtained from a 3D vision system has been elaborated in detail. The data collection process is achieved by using an established procedure that involves the collaboration of the Fanuc M710 robot and an integrated system (camera-laser). In the calibration process, the robot with a vision system is positioned at a series of known positions relative to the calibration board.

The essence of the algorithm lies in the precise identification of coordinates corresponding to the edges belonging to the calibration board, which are used for calibrating the laser-camera system to map the positions of the laser detected by the camera into 3D spatial points.

Before using these coordinates for system calibration, the RANSAC algorithm is applied to detect and eliminate potential errors and inaccuracies in the dataset, thereby improving the calibration accuracy. After applying the RANSAC algorithm, data processing is performed using a pre-developed neural network. The neural network's task is to test the success of the calibration, evaluating how accurately the system locates the calibration object in the real world. Calibration accuracy is crucial for the reliability and precision of all subsequent measurements and tasks performed by the system. Accurate calibration enables robots and image processing systems to precisely understand their environment and execute tasks with minimal or no errors.

Keywords: 3D Vision System laser camera, calibration, Python, RANSAC algorithm, neural network.

1. UVOD

Lasersko skeniranje je proces snimanja preciznih, trodimenzionalnih informacija stvarnog objekta, grupe objekta ili okoline, korištenjem lasera kao izvora svjetlosti.

Srž ovog završnog rada je ekstrinzična kalibracija odnosa između lasera i kamere. To je postupak koji se koristi kako bi se precizno odredio njihov relativni položaj i orijentacija u prostoru. Cilj ove automatske kalibracije je generiranje seta podataka za treniranje kalibracijske neuronske mreže za odnos kamera-laser. Nakon što se ti podaci generiraju koristi se RANSAC algoritam za pronalazak točaka lokaliziranih sa manjom točnošću koje se isključuju iz postupka treniranja neuronske mreže s ciljem točnijeg treniranja kalibracijske neuronske mreže.

Nakon što se neuronska mreža istrenira na ranije spomenutom setu podataka, ta istrenirana neuronska mreža omogućava da kada kamera vidi laser na nekom pikselu u slici da izračuna gdje je objekt na kojem je laser u 3D prostoru.

Postupak ekstrinzične kalibracije odnosa laser-kamera ključan je za mnoge primjene, uključujući robotski vid, sustave za autonomnu vožnju, industrijsku automatizaciju te mnoge ostale situacije gdje je potrebno točno poznavanje položaja i orijentacije uređaja u stvarnom svijetu kako bi se donosile odluke i obavljale zadatke.

U nastavku je objašnjen proces prikupljanja podataka te je objašnjen algoritam te neki njegovi dijelovi koji služe za pronalaženje lokalnog minimuma tj. bridova kalibracijske ploče. Nakon toga objašnjen je RANSAC algoritam te su prikazani rezultati dobiveni nakon što se set podataka provuče kroz neuronsku mrežu.

2. OPIS KORIŠTENE OPREME

Za dobivanje seta podataka koji je korišten u ovome završnome radu korišteni su: NVIDIA računalni modu Jetson Nano, kamera DMM 36AR0234-ML, 2D laser 45 stupnjeva, kalibracijska ploča izrađena u CRTI sa udaljenosti među bridovima od 20 mm te robot Fanuc M710.

Slika 1. Sklop lasera i kamere

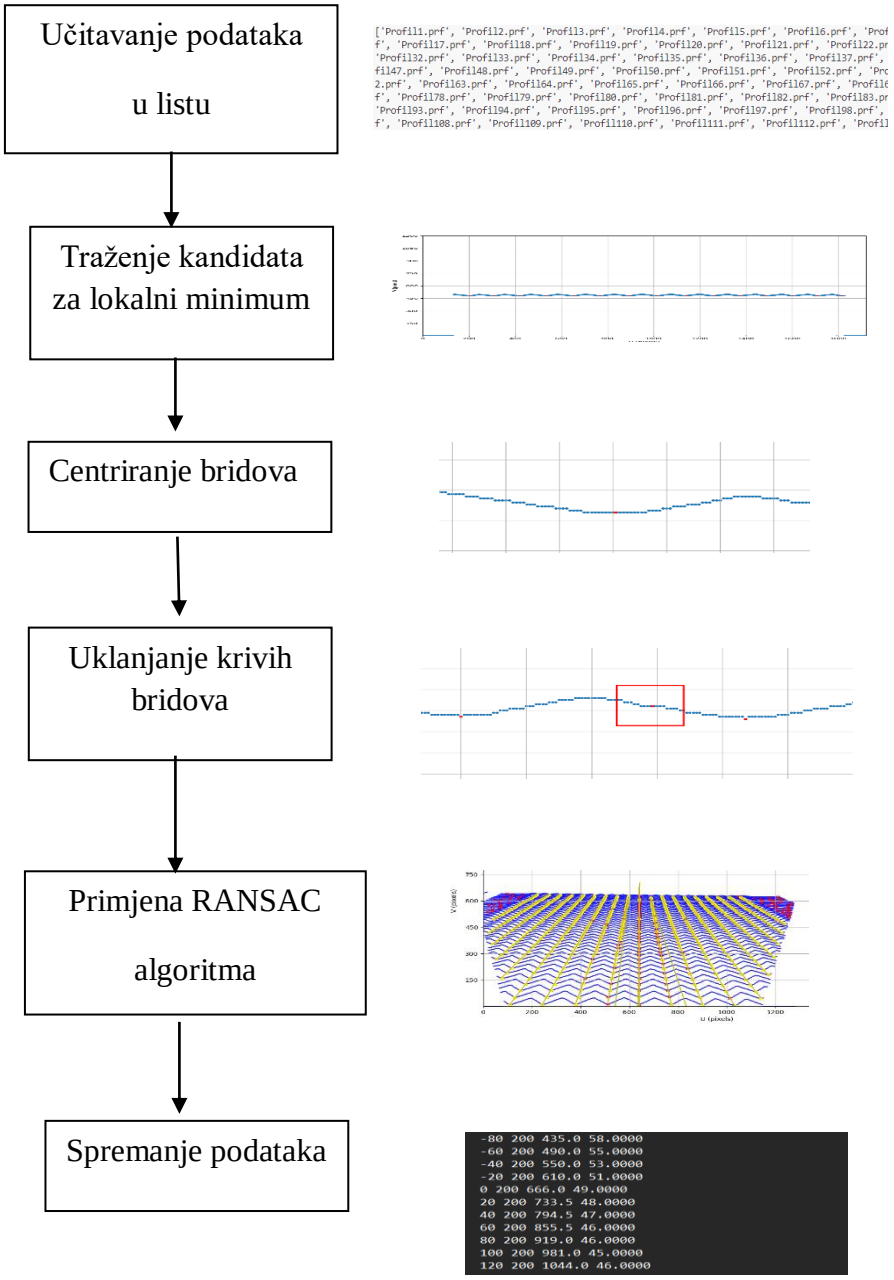
Slika 2. Sustav laser-kamera montirani na robotu Fanuc M710

Slika 3. Mjerenje podataka

Na Slika 3. možemo vidjeti način na koji su podaci bili izmjereni. Sustav laser-kamera je montiran na robota te laser projicira svoje zrake na kalibracijsku ploču. Kamera snima podatke te se ti podaci obrađuju te spremaju.

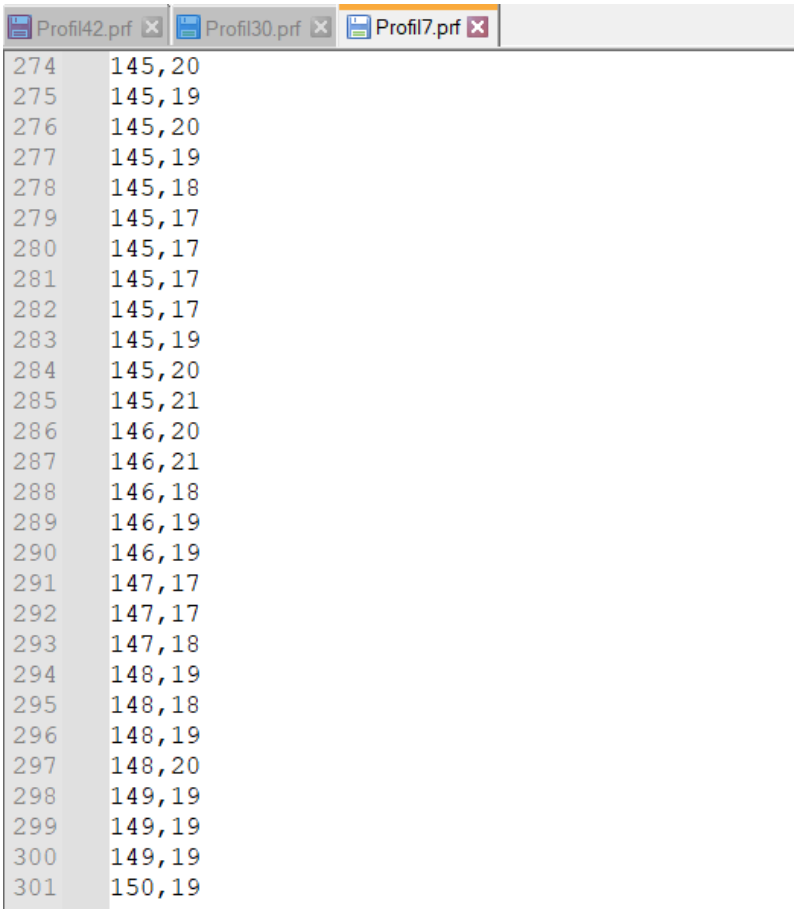
3. ALGORITAM ZA LOKALIZACIJU KALIBRACIJSKOG OBJEKTA

Algoritam za lokalizaciju bridova sastoji se od nekoliko bitnih dijelova kao što su:



Da bi se uopće krenulo u algoritam lokalizacije prvo je potrebno prikupiti podatke. Podaci se dobivaju snimanjem sustavom laser-kamera. Nakon obavljenih brojnih snimanja kalibracijske

ploče sa više različitih pozicija, podaci se spremaju u jednu mapu u obliku „prf“ datoteka te se obrađuju u programu Python.



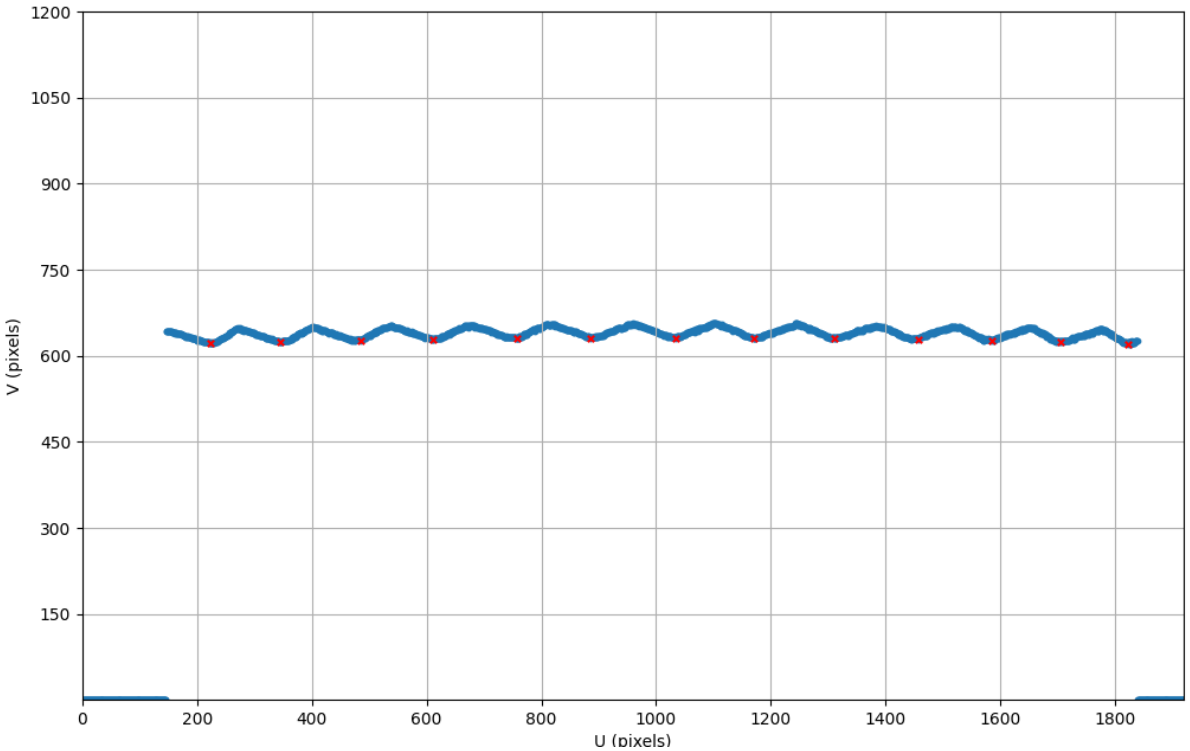
Slika 4. Prikaz podataka otvaranjem "prf" datoteke u programu Notepad++

Radi lakšeg snalaženja i boljeg razumijevanja koda koji slijedi u nastavku na Slika 4. prikazani su podaci mjerenja. Broj reda govori nam koliko smo udaljeni od ishodišta po x osi a vrijednost do zareza nam govori kolika udaljenost točke po y osi od ishodišta. Sve mjere u pikselima. Slika je rezolucije 1920x1200 stoga nam vrijednosti po x-osi idu od 0 do 1920 a po y osi od 0 do 1200 piksela.

Kako bi dodatno olakšali percepciju točaka, točke su nacrtane na grafu korištenjem programa Python. Dio algoritma zaslužan za vizualni prikaz podataka svakog profila ima sljedeći oblik :

```
### Drawing graphs ###
all_profiles1 = []
all_bridges = []
br = 1
def drawing(input_data, input_dict):
    global br
    x_cord = list(range(1, len(input_data) + 1))
    y_cord = input_data
    fig, ax = plt.subplots(
        figsize=(12, 9)
    )  # Set the figure size to maintain the aspect ratio
    ax.scatter(x_cord, y_cord, s=10)  # Blue line for profile data
    # Draw 'x' symbols using dots
    x_edges = list(input_dict.keys())
    y_edges = list(input_dict.values())
    ax.scatter(
        x_edges, y_edges, marker="x", color="red", s=15
    )  # Draw 'x' symbols using dots
    ax.set_xlim(0, 1920)  # Set x-axis limits
    ax.set_ylim(1, 1200)  # Set y-axis limits
    ax.set_xlabel("U (pixels)")
    ax.set_ylabel("V (pixels)")
    ax.grid(True)
    ax.set_aspect("equal")  # Set 1:1 aspect ratio
    ax.xaxis.set_major_locator(plt.MaxNLocator(integer=True))
    ax.yaxis.set_major_locator(plt.MaxNLocator(integer=True))
    all_profiles1.append(input_data)
    all_bridges.append(input_dict)
    filename = f"slika{br}.png"
    full_path = os.path.join(output_path1, filename)
    plt.savefig(full_path)
    #plt.close()
    plt.show()
    br += 1
```

Funkcija „drawing“ svojim pozivanjem prikazuje svaki od profila te se slike tih profila spremaju u mapu. To spremanje u mapu omogućuje nam pregled radi li naš algoritam dobro te jesu li odabrani lokalni minimumi zapravo lokalni minimumi ili je došlo do greške. Slika 5. prikazuje kako izgleda jedan takav prikaz nekog od profila sa ucrtanim lokalnim minimuma koji su dobiveni korištenjem algoritama koji je kasnije opisan u radu.



Slika 5. Vizualni prikaz jednog od profila sa pronađenim lokalnim minimumima

Sljedeći dio koda je dio koji je zaslužan za stvaranje liste sa svim datotekama koje se nalaze u mapi kako bi se kasnije lakše moglo prolaziti kroz svaki profil i učitati podatke sa njega. Kod ima sljedeći oblik:

```
def list_with_profiles(location):
    profile_list = []
    file_extension = ".prf"
    for file_name in os.listdir(folder_path):
        if file_name.endswith(file_extension):
            profile_list.append(file_name)

    def extract_number(
        filename,
    ):  # function to get profile number by which we sort them
        return int("".join(filter(str.isdigit, filename)))

    sorted_files = sorted(profile_list, key=extract_number)
    return sorted_files
```

Funkcija „list_with_profiles“ uzima lokaciju mape sa podacima kao ulaznu vrijednost a vraća listu koja ima sve profile sortirane na način tako da je profil1 prvi u listi.

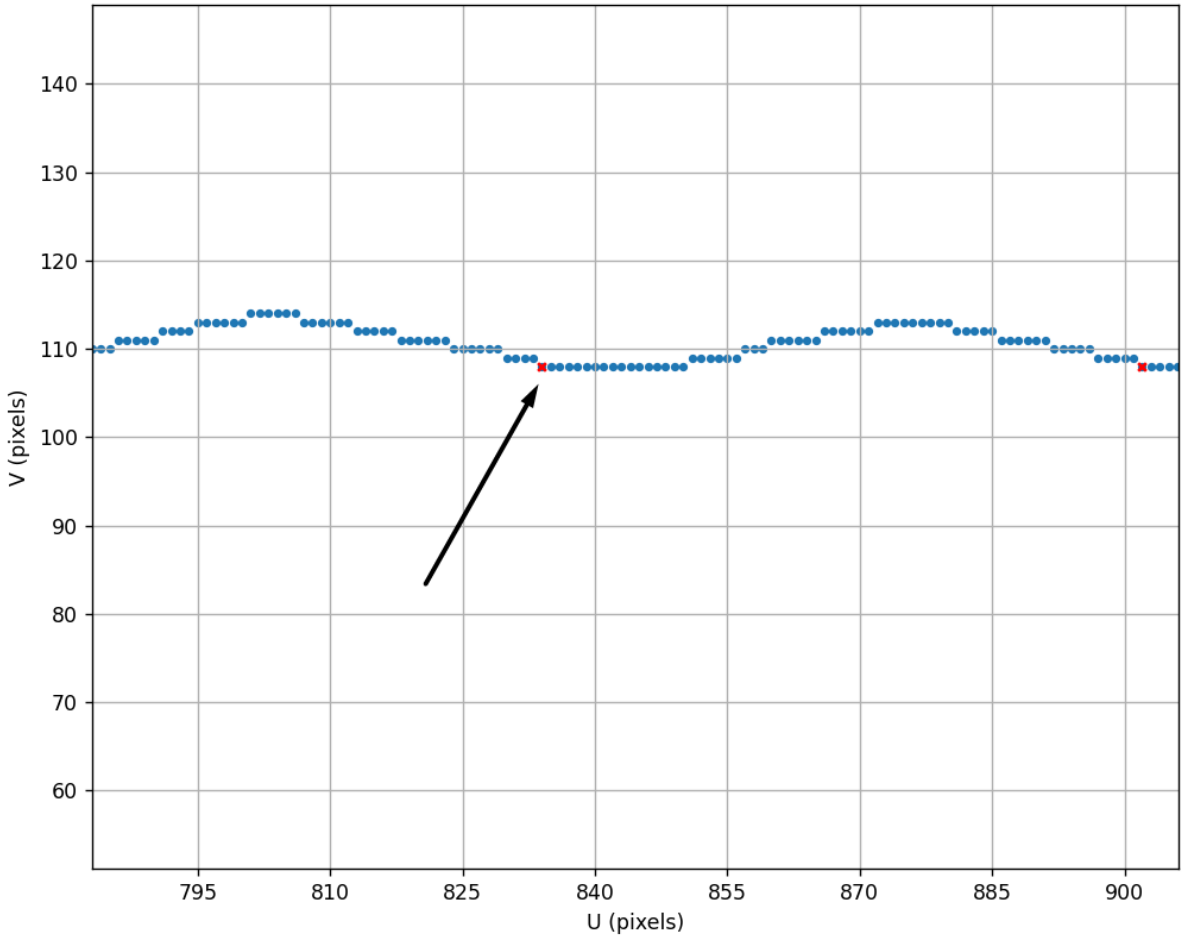
Nakon što se dobije lista sa profilima, sljedeći korak je obrada podataka svakog profila posebno. Kreira se nova funkcija „local_minimum“ koja prolazi kroz svaki red profila u potrazi za kandidatima koji bi mogli biti lokalni minimumi tj. bridovi naše kalibracijske ploče. Zbog greške mjerenja koja se pojavljuje zbog brojnih faktora kao što su distorzija svijetla, distanca, kutna preciznost, šum i interferencija, temperaturne promjene, šum i vibracija. Upravo zbog tih faktora pronalazak idealnog kandidata koji bi mogao biti lokalni minimum je podosta težak te je potrebno puno uvjeta da bi neka točka postala kandidat za lokalni minimum. Iako ima puno uvjeta koji se moraju zadovoljiti točke koje postanu kandidati ne moraju zapravo biti lokalni minimumi upravo zbog ranije spomenutih faktora koji utječu na točnost mjerenja. Uvjeti koji su korišteni za traženje lokalnih minimuma nalaze se u kodu u nastavku.

```
def local_minimum(profil):
    local_minimums = {}
    dict_with_all_profiles = {}
    for i in range(len(profil)):
        dict_with_all_profiles[i + 1] = profil[
            i
        ]  # creating a dict with all profiles
        if (
            profil[i] != 0
            and profil[i] < profil[i - 1]
            and profil[i] < profil[i - 2]
            and profil[i] < profil[i - 3]
            and profil[i] < profil[i - 4]
            and profil[i] < profil[i - 17]
            and profil[i] <= profil[i + 1]
            and profil[i] <= profil[i + 2]
            and profil[i] <= profil[i + 3]
            and profil[i - 1] > 1
            and profil[i + 1] > 1
            and profil[i] <= profil[i + 17]
        ):
            local_minimums[i + 1] = profil[i]
```

Da bi neka točka postala kandidat za lokalni minimum ona mora biti veća od prethodne 4 točke te mora biti manja ili jednaka od 3 točke nakon nje. Zašto može biti jednaka objašnjeno je u sljedećem poglavlju . Još jedan od uvjeta da točka postane kandidat da postane lokalni minimum je taj da ona također mora biti manja od 17-e točke prije nje te 17-e točke nakon nje. Taj uvjet je dodan zbog točaka koje su se pojavljivale kao kandidati za lokalne minimume a zapravo se radilo o greškama mjerenja. Još jedan od uvjeta da točka postane kandidat je ta da vrijednost nakon nje i prije nje budu veće od nule. Razlog dodavanja toga uvjeta je to da se ponovo izbjegnu točke koje imaju vrijednost manju od 1 a zadovoljavaju ranije spomenute uvjete. Točke koje zadovoljavaju spomenute uvjete dodaju se u rječnik „local_minimums“. Oblik rječnik odabran je iz razloga što nam ključ rječnika predstavlja poziciju(u pikselima) gdje se kandidat za minimum nalazi, a vrijednost nam predstavlja kolika je ta vrijednost našeg minimuma koja je također izražena u pikselima. Iz priloženog koda se vidi da se također stvara jedan rječnik sa sa svim vrijednostima. Cilj stvaranja te liste objašnjen je dalje u radu.

3.1 Centralizacija lokalnog minimuma

Jedan od problema koji se pojavljuje prilikom traženja lokalnog minimuma tj. točne vrijednosti i točne pozicije brida je netočnost podataka. U idealnome slučaju jedna točka trebala bi predstavljati brid i u takvome slučaju uvjet za traženje minimuma bi bio da su sljedeće tri vrijednosti nakon te točke samo veće no pošto je takav rezultat skoro pa nemoguće dobiti zbog nepreciznosti mjerenja, u setu podataka stvori se više točaka sa istom vrijednosti (jedan od idealnijih slučajeva) koje predstavljaju lokalni minimum tj. brid kalibracijske ploče te je iz tog razloga uvjet za traženje kandidata za lokalni minimum taj da 3 vrijednosti nakon ne moraju biti nužno samo veće već mogu biti veće ili jednake.



Slika 6. Grafički prikaz jednog od kandidata za lokalni minimum

Na Slika 6. možemo vidjeti lokalni minimum koji je zadovoljio uvjete funkcije „local_minimum“. Plavom bojom obojene točke su sve ostale točke koje nisu zadovoljile uvjete da postanu kandidat za lokalni minimum.

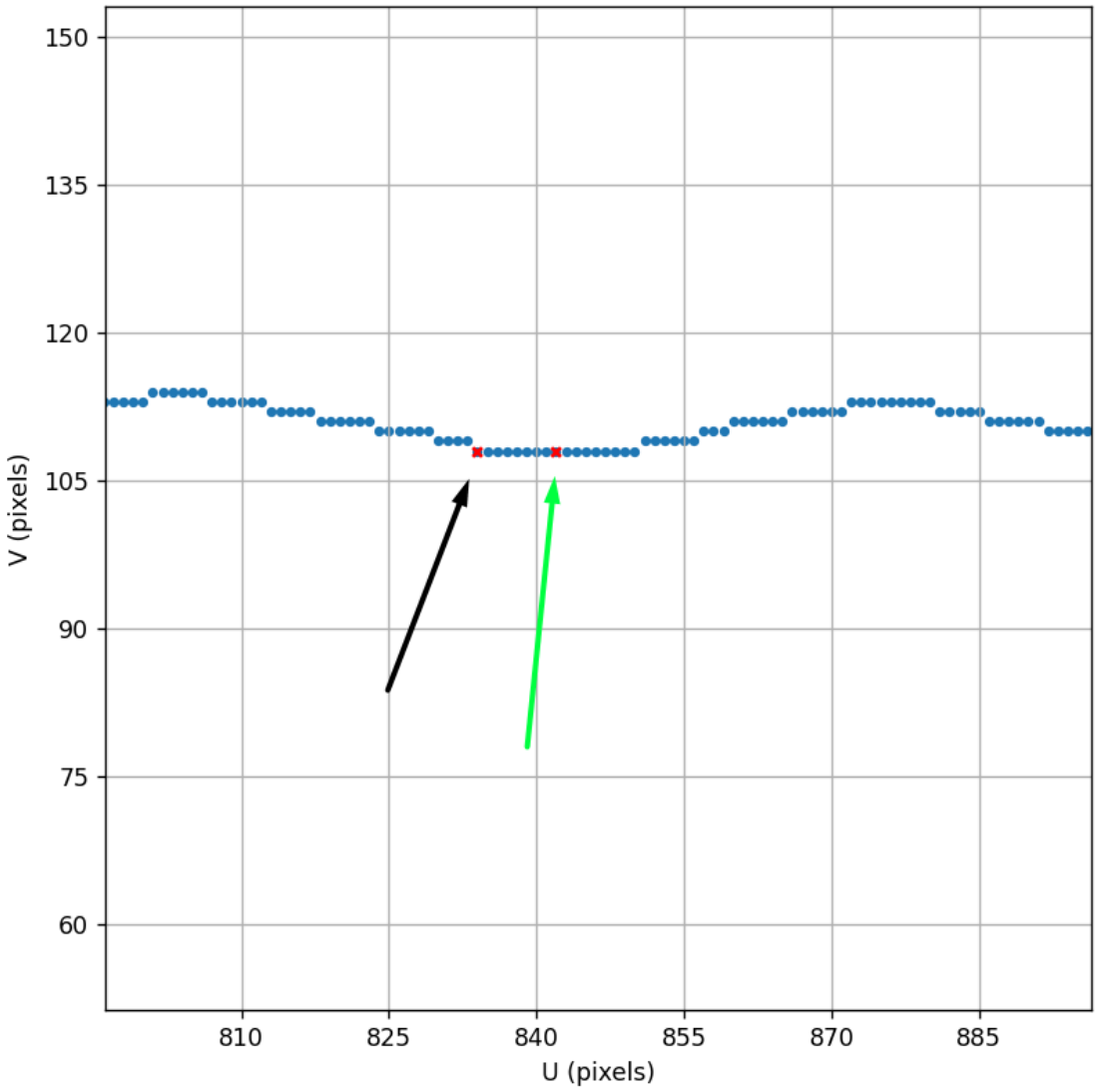
Promatrajući Slika 6. možemo uočiti da postoji još puno točaka koje imaju istu vrijednost kao i naš kandidat te da je zapravo stvarni položaj našeg lokalnog minimuma sredina tih točaka koje imaju istu vrijednost. Upravo iz tog razloga je potrebno definirati funkciju koja će centralizirati našeg kandidata za lokalni minimum. Ta funkcija naziva se „centering“ te prvi dio koda ima sljedeći oblik:

```
def centering(dict_all_points, dict_minimums):
    global br
    keys_to_delete = []
    sum = 0
    counter = 0
    keys_to_add = []
    last_added_key = 1
    last_value = None
    for key1, value1 in dict_minimums.items():
        counter += 1
        sum += key1
        if key1 in dict_all_points:
            key2 = key1
            count = 0
            while key2 in dict_all_points and (
                dict_all_points[key2] == value1
                or (
                    key2 + 1 in dict_all_points
                    and dict_all_points[key2 + 1] == value1
                )
                or (
                    key2 + 2 in dict_all_points
                    and dict_all_points[key2 + 2] == value1
                )
                or (
                    key2 + 3 in dict_all_points
                    and dict_all_points[key2 + 3] == value1
                )
                or (
                    key2 + 4 in dict_all_points
                    and dict_all_points[key2 + 4] == value1
                )
            ):
                count += 1
                key2 += 1
            if count > 0:
                interval_middle = (key1 + key2 - 1) / 2.0
                if interval_middle - last_added_key <= 10:
                    last_added_key = interval_middle
                    last_value = value1
                keys_to_add.append((interval_middle, value1))
                keys_to_delete.append(key1)
```

Ulazne vrijednosti u funkciji „centering“ su rječnik sa svim vrijednostima koji smo stvorili na početku te rječnik sa kandidatima za lokalni minimum. For petljom prolazimo kroz ključeve i vrijednosti tih ključeva koristeći .items() metodu.

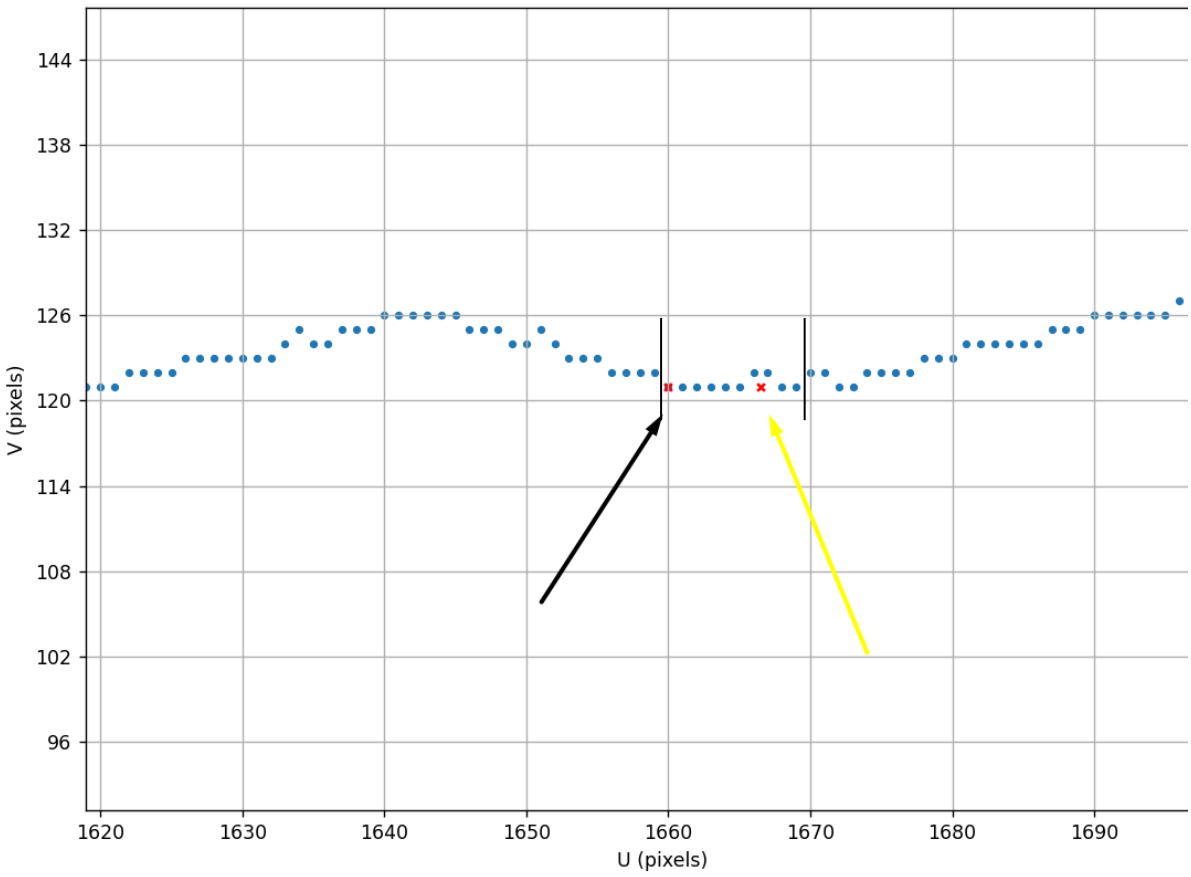
Zatim provjeravamo nalazi li se naš ključ u rječniku sa svim vrijednostima što bi se morao nalaziti te nakon toga provjeravamo vrijednosti ključeva nakon njega koristeći while petlju.

While petlja se vrti sve dok jedan od 4 ključeva unaprijed nema istu vrijednost. Razlog zašto se ne provjerava samo jedan od ključeva unaprijed je taj da može doći do pojavljivanja greške ako jedna točka odstupa, a ostale nakon nje imaju istu vrijednost.



Slika 7. Prikaz starog kandidata te novog kandidata za lokalni minimum

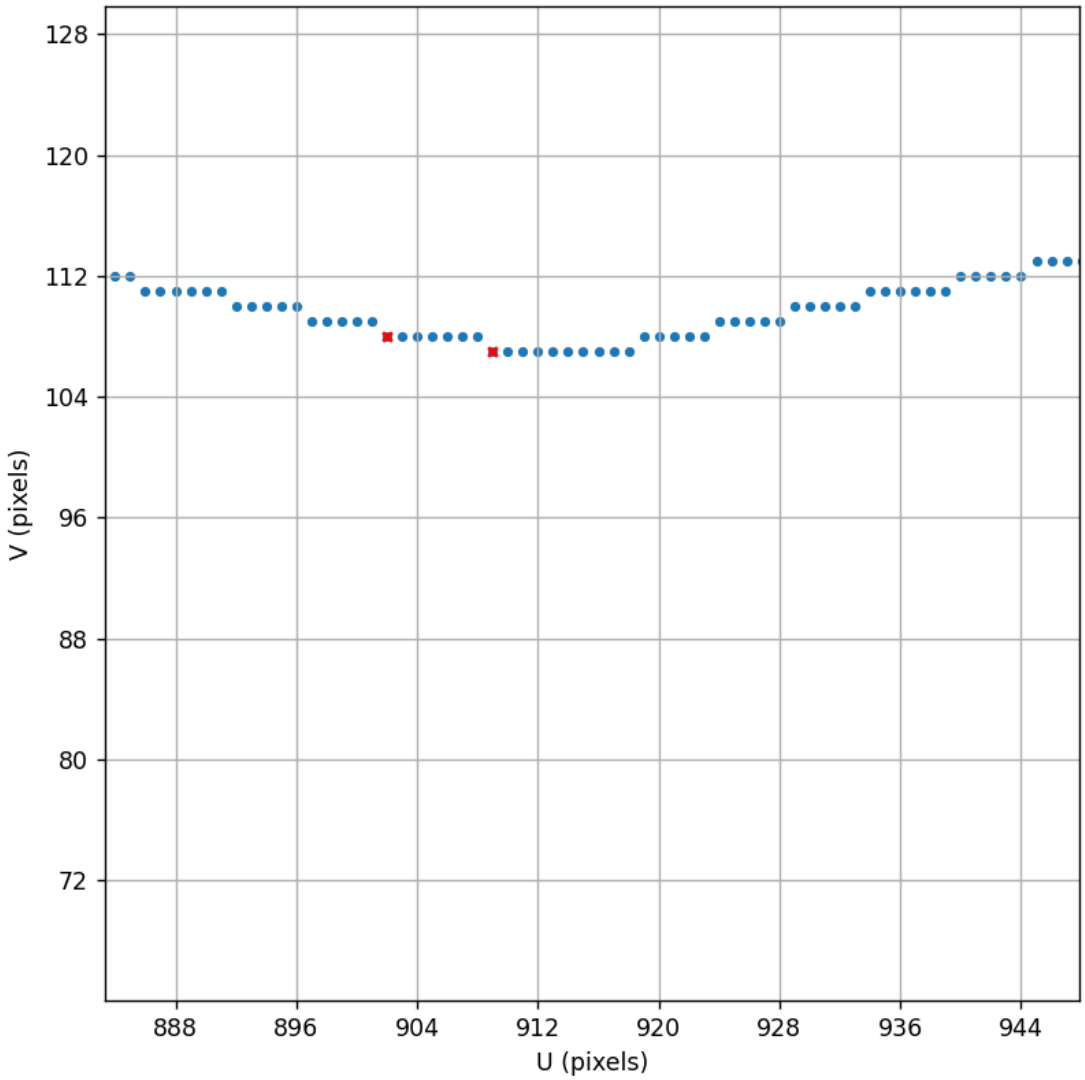
Slika 7. prikazuje prvotnog kandidata za lokalni minimum (označeno crnom strelicom) te novog kandidata za lokalni minimum koji zapravo i je lokalni minimum. Funkcija „centering“ broji koliko točaka sa istom vrijednosti uz dozvoljena odstupanja ima nakon prvotnog kandidata ako ih uopće ima. Na Slika 7. prikazan je jedan od idealnijih slučajeva gdje nije došlo do nijedne greške tj. sve točke su iste vrijednosti. Nova točka zadržava vrijednost prvotnog kandidata, ali se nalazi na poziciji koja puno bolje odgovara stvarnome položaju brida. Nova točka dodaje se u listu keys_to_add u obliku kao tupple. Taj oblik je kasnije puno lakši za obradu od rječnika. Prvotni ključ koji je bio na krivoj poziciji dodaje se u listi keys_to_delete te se kasnije briše. Razlog zašto se direktno ne brišu je taj što što se u Pythonu ne smiju brisati ključevi rječnika kada se vrti for petlja kroz rječnik jer se mijenja veličina rječnika dok se petlja vrti što može dovesti do neočekivanog ponašanja ili grešaka. Stoga se vrijednosti spremaju u liste te se kasnije iterira kroz te liste te se dodaju ili brišu parovi ključ, vrijednost.



Slika 8. Traženje lokalnih minimuma u nesavršenim uvjetima

Na Slika 8. može se vidjeti da algoritam dobro pronalazi novi položaj lokalnog minimuma(označeno žutom strelicom) iako nisu sve vrijednosti jednake već imamo dva odstupanja po dvije točke.

Novi problem koji se pojavljuje u nekim profilima na nekim mjestima je taj kad imamo dva kandidata za lokalni minimum jedan kraj drugoga. Taj slučaj vidljiv je na Slika 9.

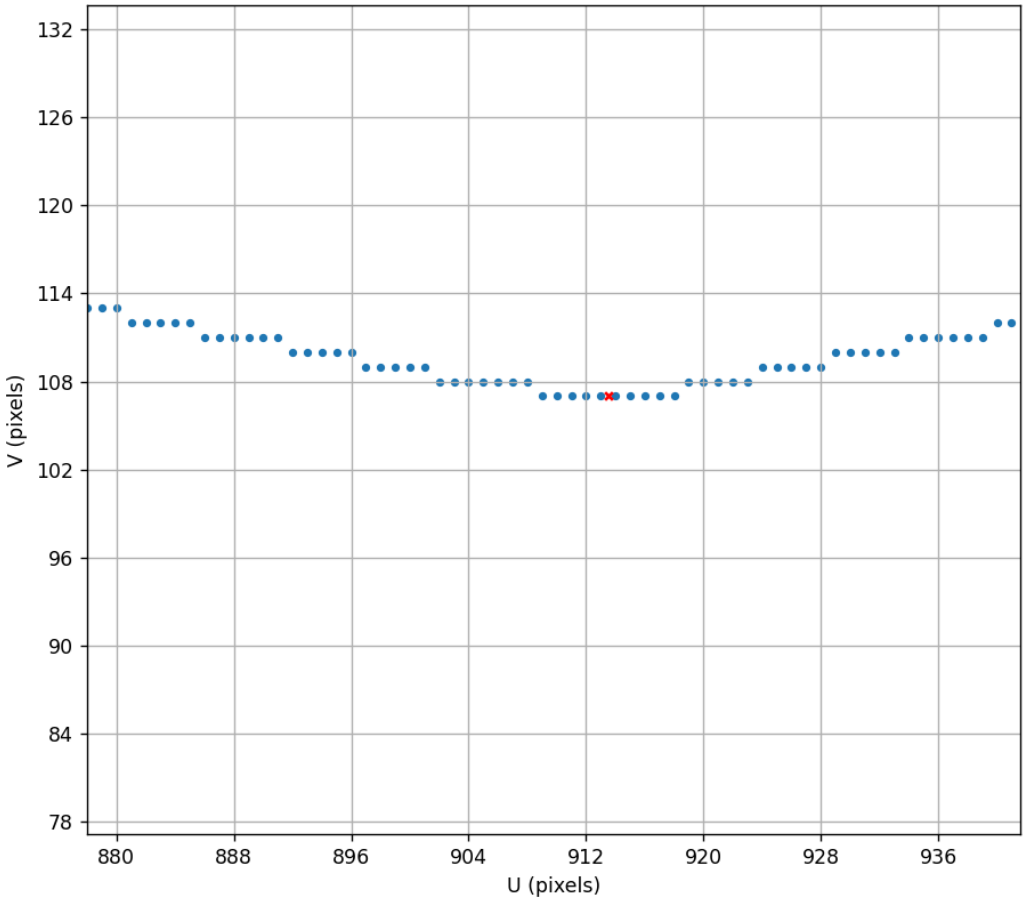


Slika 9. Kandidati za lokalni minimum na bliskoj udaljenosti

Na Slika 9. možemo primijetiti da se kandidati za lokalni minimum nalaze na pozicijama 901 i 909. Prvi dio funkcije „centering“ prolazi kroz oba ključa te ih oboje briše te stvara nove ključeve sa točnijim pozicijama no i dalje su prisutna dva kandidata. Rješenje tog problema nalazi se u drugome dijelu funkcije centering koja glasi:

```
        bad_keys = []
        space = keys_to_add[1][0] - keys_to_add[0][0]
        for i in range(len(keys_to_add) - 1):
            temporary_key, temporary_value = keys_to_add[i]
            next_key, next_value = keys_to_add[i + 1]
            if next_key - temporary_key < 20 and temporary_key != next_key:
                if next_value > temporary_value:
                    bad_keys.append(next_key)
                else:
                    bad_keys.append(temporary_key)
        for keyz in keys_to_delete:
            if keyz in dict_minimums:
                del dict_minimums[keyz]
        for keey, valuee in keys_to_add:
            if keey not in bad_keys:
                dict_minimums[keey] = valuee
        return dict_minimums
    output_dict = centering(dict_with_all_profiles, local_minimums)
```

Problem je riješen na taj način tako što se kreira lista bad_keys. Iterirajući kroz ključeve koje trebamo dodati promatramo trenutni ključ pod imenom temporary_key te ključ koji slijedi nakon njega pod imenom next_key. U slučaju kada je razlika ključeva manja od 20 to znači da su ključevi jedan kraj drugoga. Gledajući slučaj sa Slika 9. oduzmemo li 909 i 901 dobivamo 8 što je manje od 20 što znači da su naši ključevi zadovoljili taj uvjet. Ako par ključeva zadovolji taj uvjet u listu bad_keys se dodaje ključ koji ima veću vrijednost što rezultira time da uvijek ostane pa ključ sa manjom vrijednosti što je uistinu minimum što je vidljivo na Slika 10.



Slika 10. Lokalni minimum koji ostaje nakon funkcije "centering"

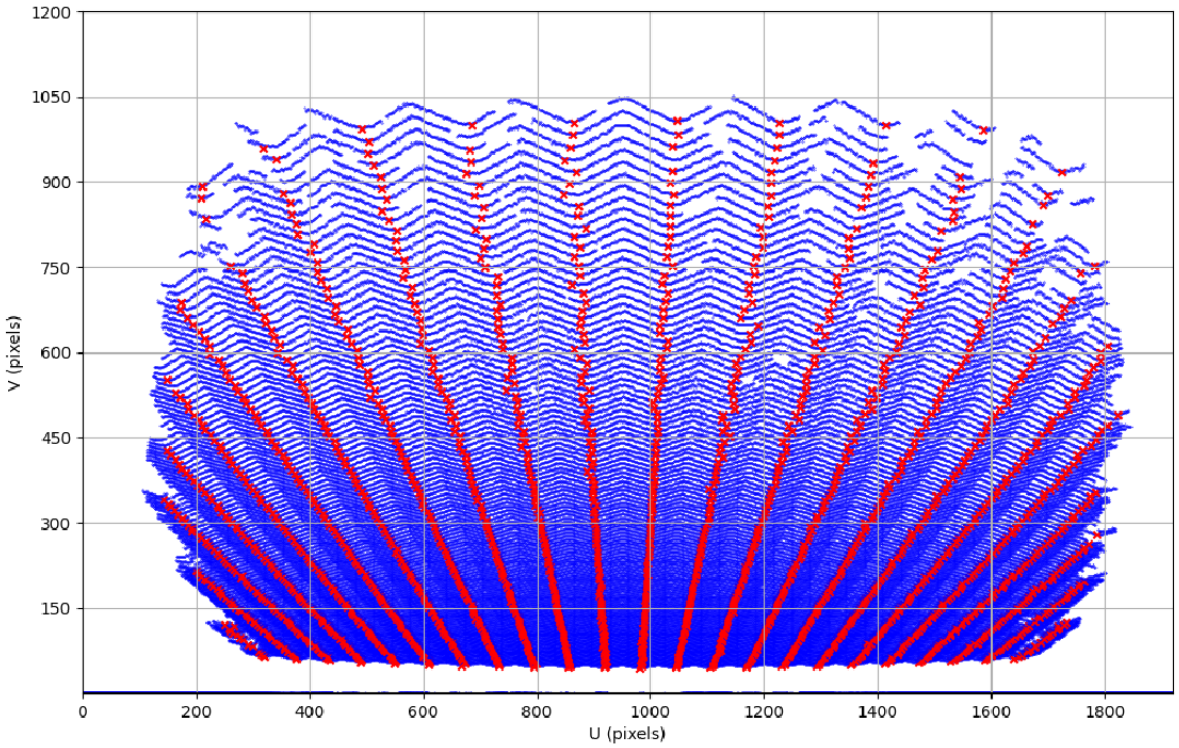
Propust koji se može dogoditi kod ovakvog načina rješavanja problema je taj da jedan od ključeva ostaje na svome mjestu tj. da ne ulazi u funkciju „centering“ te se ne može detektirati da imamo dva ključa koja su blizu jer se taj jedna ključ ne nalazi u listi keys_to_add. Rješenje tog problema je funkcija „removing_wrong_edges“ koja je opisana u sljedećem poglavlju.

3.2 Funkcija za uklanjanje krivih bridova

```
def removing_wrong_edges(dct):
    global a
    keys = list(dct.keys())
    keys_to_delete = []
    for i in range(1, len(keys)):
        last_key = keys[i - 1]
        current_key = keys[i]

        if current_key - last_key < a:
            if dct.get(last_key, 0) > dct.get(current_key, 0):
                keys_to_delete.append((last_key, dct[last_key]))
            else:
                keys_to_delete.append((current_key, dct[current_key]))
    for key_x, key_value in keys_to_delete:
        if key_x in dct:
            del dct[key_x]
        elif key_value in dct.values():
            keys_to_delete = [
                key for key, value in dct.items() if value == key_value
            ]
            for key in keys_to_delete:
                del dct[key]
    return dct

output_dict = removing_wrong_edges(output_dict)
```

Funkcija iterira kroz ključeve rječnika koji smo dobili kao output funkcije „centering“, ti ključevi zapravo predstavljaju sve pozicije bridova koje smo za sad dobili. Uvjet koji provjeravamo je kolika je razlika između ključeva te taj broj uspoređujemo sa globalnom varijablom a. Ako je ta razlika manja nego što bi trebala biti tada se ključ sa većom vrijednosti proširuju u listu kroz koju se na kraju iterira kako bi se obrisali parovi iz rječnika bridova.

Nakon što su pronađeni svi lokalni minimumi njihov grafički prikaz možemo vidjeti na Slika 11.



Slika 11. Grafički prikaz lokalnih minimuma na svim profilima

Crvenom bojom te sa x simbolom su nacrtani svi lokalni minimumi koji su pronađeni dok su plavom bojom označene sve točke koje nisu zadovoljile uvjete da postanu lokalni minimum. Promatrajući Slika 11. možemo primijetiti da algoritam dosta dobro pronalazi lokalne minimume te da se oni relativno dobro pozicioniraju te imaju dobre vrijednosti no kako bi dodatno poboljšali točnost te uklonili neke od grešaka(outliere) koje je algoritam izbacio koristi se RANSAC algoritam koji je detaljno objašnjen u sljedećem poglavlju.

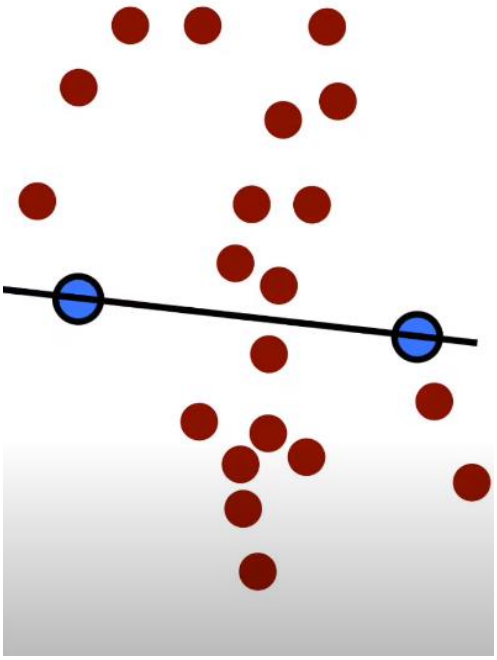
4. RANSAC ALGORITAM

RANSAC (Random Sample Cnsensus) je iterativni statistički algoritam koji se koristi za procjenu parametara matematičkog modela na temelju skupa podataka koji sadrži određeni broj odstupajućih (outlier) ili pogrešnih mjerenja. Glavna zadaća algoritma je identificirati i izolirati ispravne podatke (inliers) između nepouzdanih ili kontaminiranih podataka(outliers) te procijeniti parametre modela koristeći samo ispravne podatke što omogućava preciznu kalibraciju kamere i senzora za određivanje položaja i orijentacije objekta u prostoru.

RANSAC algoritam ima široku primjenu u području robotike i strojnog vida zbog svoje sposobnosti za robusno i pouzdano rješavanje problema odabira modela iz podataka s prisutnošću outliera ili šuma. Neke od konkretnih primjena RANSAC algoritma su sljedeće:

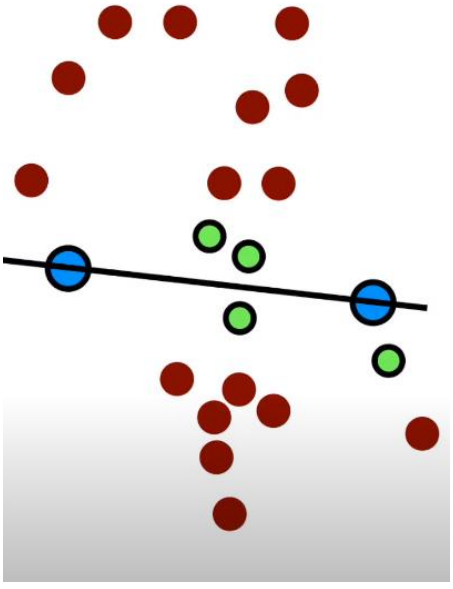
1. Stereo vizija: U stereoskopskoj viziji, RANSAC se može koristiti za određivanje odgovarajućih parova točaka u dvije stereoskopske slike što pomaže u određivanju dubine objekata i stvaranje trodimenzionalnih mapa okoline
2. Registracija oblaka točaka: U području robotske percepcije, RANSAC se često koristi za registraciju oblaka točaka dobivenih iz različitih senzora, poput 3D skenera i LIDAR-a gdje pomaže u poravnanju i registraciji oblaka točaka kako bi se stvorila cjelovita slika okoline.
3. Detekcija objekata: RANSAC može biti primijenjen za detekciju i praćenje objekata u realnom vremenu. Na primjer, u autonomnim vozilima, ovaj algoritam može pomoći u prepoznavanju drugih vozila ili prepreka na cesti.
4. Segmentacija slika: U analizi slika, RANSAC može biti korišten za segmentaciju slika, posebno kada su objekti u slici različitih oblika i veličina gdje algoritam pomaže u izdvajanju relativnih dijelova slike, čime se olakšava daljnja obrada.

Način na koji RANSAC algoritam funkcionira je sljedeći. Zamislimo da imamo skup točaka te kroz taj skup točaka je potrebno provući pravac. Neke od tih točaka su ispravni podaci dok su neki drugi greške. Prvi korak RANSAC algoritma je nasumičan odabir dvije od tih točaka u cilju da provučemo pravac kroz njih što je vidljivo na Slika 12.



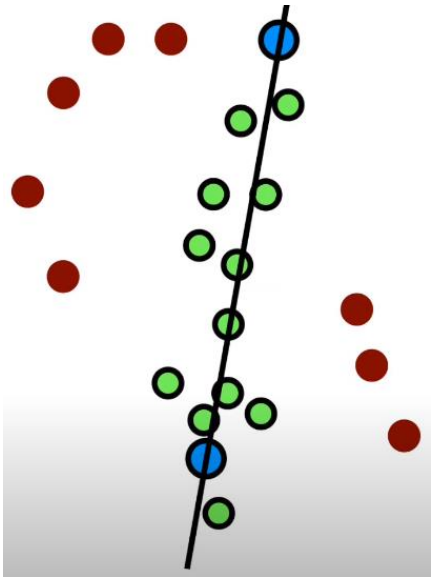
Slika 12. Skup točaka i nasumično provučen pravac

Sljedeći korak je brojanje koliko točaka leži na pravcu ili blizu pravca ovisno o željenoj greški tj odstupanju koje se postave te točke proglašavaju se kao inliers, a ostale točke proglašavaju se kao outliers što je vidljivo na Slika 13.



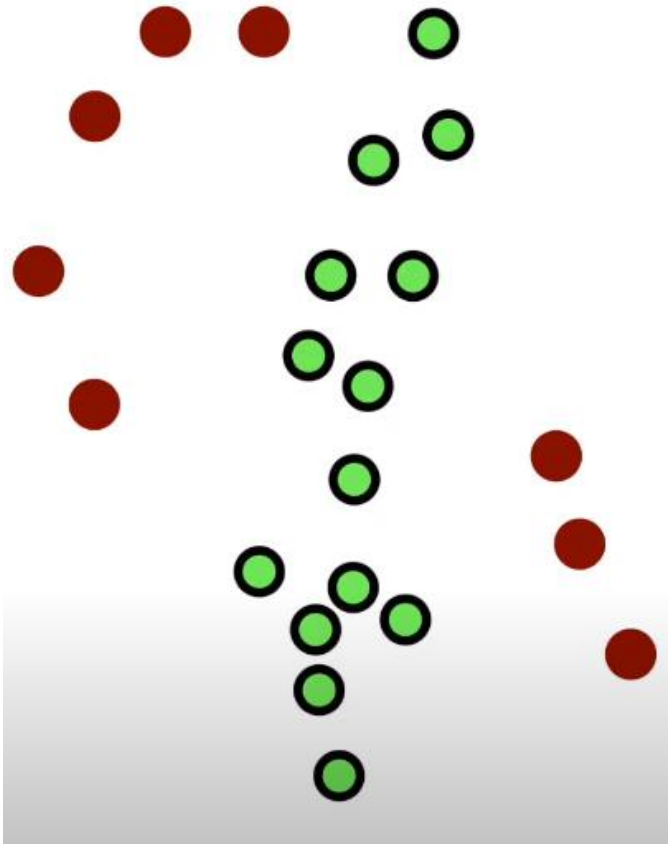
Slika 13. Proglašavanje inliera i outliera

Na Slika 13.. zelenom bojom su obojane točke koje smo proglasili kao inliere a crvenom točke su sve ostale točke koje su outlieri. Ovaj nasumičan odabir dvije točke(koje su označene) plavom bojom ostvario je rezultat od 4 pronađena inliera. Sljedeći korak je ponovni odabir dvije točke te brojanje koliko inliera je pronađeno. Na Slika 14. mogu se vidjeti nove dvije točke koje su generirane nasumično (označeno plavom bojom) te svi inlieri koji su proglašeni(označeno zelenom bojom). Može se primijetiti de je broj inliera ovaj put 12 komada.



Slika 14. Nove dvije točke i novi inlieri

Ovaj postupak se ponavlja deset, sto, tisuću puta sve dok se ne dobije najviši rezultat koji je diktiran koliko je inliera pronađeno. Kada je pronađen najveći rezultat sa najviše pronađenih inliera početne dvije točke se također zapamte kao inlieri dok sve ostale proglasimo outlierima što je vidljivo na Slika 15.



Slika 15. Konačni rezultat

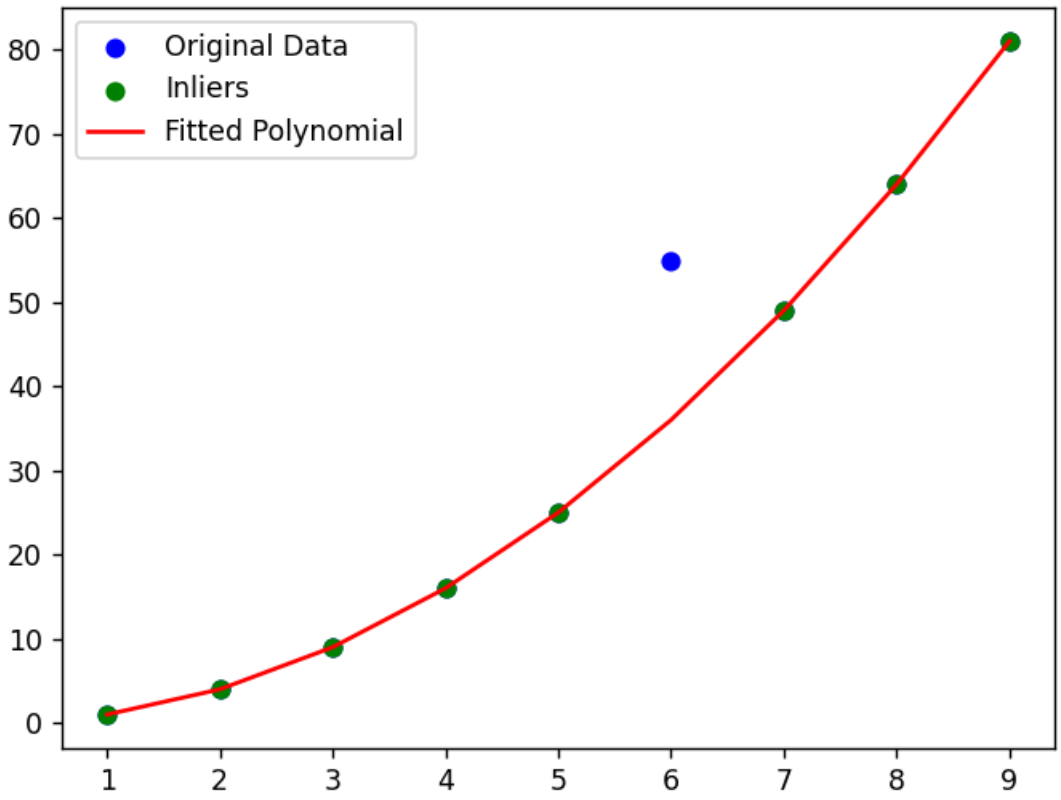
Koristeći ovu logiku, RANSAC algoritam ima sljedeći oblik u Pythonu:

```
def ransac_polyfit(x, y, degree=2, threshold=1.0, iterations=100):
    """
    Use RANSAC algorithm to robustly fit a polynomial of a given degree to 2D
data points.

    Parameters:
    - x, y: Data points
    - degree: Degree of polynomial to fit
    - threshold: Maximum distance for a point to be considered an inlier
    - iterations: Number of iterations for the RANSAC algorithm
    Returns:
    - best_coeffs: Coefficients of the best-fitting polynomial
    - best_inliers: Inliers for the best-fitting polynomial
    """
    best_inliers = []
    best_coeffs = None
    for _ in range(iterations):
        # Randomly select 'degree + 1' points
        random_indices = np.random.choice(len(x), degree + 1, replace=False)
        x_sample = x[random_indices]
        y_sample = y[random_indices]
        # Fit polynomial to random points
        coeffs = np.polyfit(x_sample, y_sample, degree)
        poly = np.poly1d(coeffs)
        # Compute errors
        y_pred = poly(x)
        errors = np.abs(y - y_pred)
        # Identify inliers based on the error threshold
        inliers = np.where(errors < threshold)[0]
        # Update best model if the number of inliers increased
        if len(inliers) > len(best_inliers):
            best_inliers = inliers
            best_coeffs = coeffs

    return best_coeffs, best_inliers
```

Kao ulaz u funkciju koriste se x koordinate točaka, y koordinate točaka, stupanj polinoma u koji želimo spojiti točke, tolerirano odstupanje te broj iteracija. Kao izlaz funkcija vraća najbolje koeficijente te najbolje inliere. Najbolje inliere vraća kao listu sa pozicijama na kojima se nalaze najbolji inlieri .

```
# Sample data
x = np.array([ 1, 2, 3, 4, 5, 6, 7, 8, 9])
y = np.array([ 1, 4, 9, 16, 25, 55, 49, 64, 81])
# Apply RANSAC
best_coeffs, best_inliers = ransac_polyfit(x, y, degree=2, threshold=5.0,
iterations=100)
# Visualize results
plt.scatter(x, y, label='Original Data', color='b')
plt.scatter(x[best_inliers], y[best_inliers], label='Inliers', color='g')
plt.plot(x, np.poly1d(best_coeffs)(x), label='Fitted Polynomial', color='r')
plt.legend()
plt.show()
print(best_inliers)
print("Best-fitting coefficients:", best_coeffs)
```

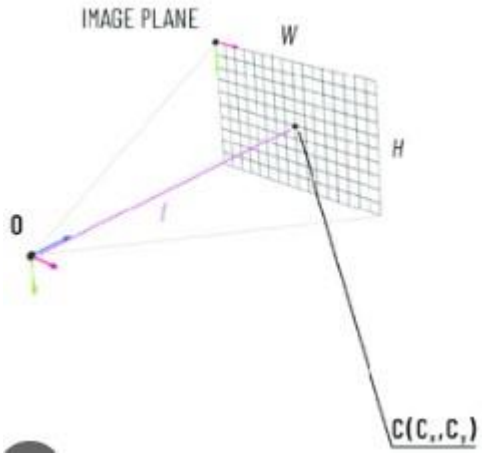
Za odabrani testni skup točaka vidljiv u kodu iznad kao izlaz funkcija vraća sljedeću listu inliera: [0 1 2 3 4 6 7 8]. Ova lista nam govori da se inlieri nalaze na svim pozicijama osim na poziciji 5 što odgovara poziciji plave točke vidljive na Slika 16.



Slika 16. RANSAC algoritam u Pythonu

5. POHRANA I ZAPIS PODATAKA

Posljednja zadaća algoritma je zapisati dobiveni podatke u txt datoteku te provesti testiranje točnosti nad dobivenim podacima. Podaci se zapisuju na sljedeći način:



Slika 17. Koordinatni sustav kamere i bridova

-60 -300 794.5 47.0000

-40 -300 855.5 46.0000

-20 -300 919.0 46.0000

0 -300 981.0 45.0000

20 -300 1044.0 46.0000

40 -300 1105.5 46.0000

60 -300 1169.0 47.0000

…

0 -295 985.5 48.0000

Vrijednost prvoga stupca predstavlja udaljenost brida od središta koordinatnog sustava O gledajući u horizontalnom smjeru(smjer osi W)

Na primjeru ovih podataka može se uočiti da je 4.red točno u ishodištu jer mu je vrijednost nula te se radi i o središnjem bridu. Da je to uistinu središnji brid može se uočiti prema trećoj vrijednosti. Naime slika je rezolucije 1920x1200 te kada podijelimo 1920 sa 2 dobije se 960, a broj najbliži 960 iz ovoga skupa podatak je 981. Svim ostalim vrijednostima koje su veće od one od središnjeg brida se vrijednost u prvome stupcu povećava za 20 jer je udaljenost od brida do brida na kalibracijskoj ploči točno 20 milimetara.

Druga brojka u svakome redu predstavlja udaljenost kamere od početne lokacije robota. Početna lokacija robota označena je sa koordinatnim sustavom koji ima središte O te je taj koordinatni sustav nepomičan. Na ovome skupu podatak vidimo da ta udaljenost iznosi -300 milimetara za prvih 7 vrijednosti što znači da je robot na udaljenosti 300 milimetra u negativnome smjeru gledajući plavu os sa Slika 17. To je ujedno i najdalja pozicija od kud robot pravi slike. Sljedeća vrijednost koja dolazi je -295 što znači da se robot pomaknuo za 5 milimetra u pozitivnome smjeru gledajući plavu os. Vrijednost -300 ujedno nam i govori da se radi o prvome profilu jer robot slika prvo sa najdalje pozicije te se kasnije približava kalibracijskoj ploči.

Treća i četvrta predstavljaju vrijednosti dobivene algoritmom. Treća vrijednost predstavlja poziciju brida mjereći u pikselima (koordinatna os W) dok četvrta vrijednost predstavlja visinu(H) na kojoj se brid nalazi.

Sve navedene vrijednosti zapisane su u praznu txt datoteku te na njih nije primijenjen RANSAC-OV algoritam. U nastavku slijedi kod koji pojašnjava kako se primjenjuje RANSAC-ov algoritam na dobivene podatke.

```
all_offsets = list(range(-220, 221, 20))
good_edges = {}
offset_count_list = []
for offset_x in all_offsets:
    extracted_numbers_x = []
    extracted_numbers_y = []
    offset_count = 0
    with open(
        all_edges_output_filename, "w"
    ) as all_bridovi_file:  # , open(all_bridovi_output_filename_novi, 'w') as
all_bridovi_file_novi:
        target_value=960

        for idx, (file_name, bridovi_dict) in enumerate(all_edges.items()):
            closest_key = find_closest_key(bridovi_dict, target_value)
            target_value=closest_key
            keys = list(bridovi_dict.keys())
            second_number = (-300) + idx * offset_increment1
            for i, key in enumerate(keys):
                offset = offset_increment * (keys.index(closest_key) - i)
                position = i * 10
                if offset == offset_x:
                    extracted_numbers_x.append(key)
                    extracted_numbers_y.append(bridovi_dict[key])
                    offset_count += 1
                offset_pair = f"{offset:<5}{second_number:<5}"  # Increase
spacing width
                formatted_line = (
                    f"{offset_pair}{key:>{spacing_width}}   {bridovi_dict[key]
:.4f}\n"
                )
                formatted_line = " ".join(
                    formatted_line.split()
                )  # Ensure consistent spacing
                all_bridovi_file.write(formatted_line + "\n")
    x = np.array(extracted_numbers_x)
    y = np.array(extracted_numbers_y)
    best_coeffs, best_inliers = ransac_polyfit(
        x, y, degree=1, threshold=7.00, iterations=100
    )
    all_extracted_numbers_x.append(extracted_numbers_x)
    all_extracted_numbers_y.append(extracted_numbers_y)
    all_x.append(x)
    all_y.append(y)
    all_inliers.append(best_inliers)
    all_coeffs.append(best_coeffs)
    good_edges[offset_x] = list(best_inliers)
    offset_count_list.append(offset_count)
```

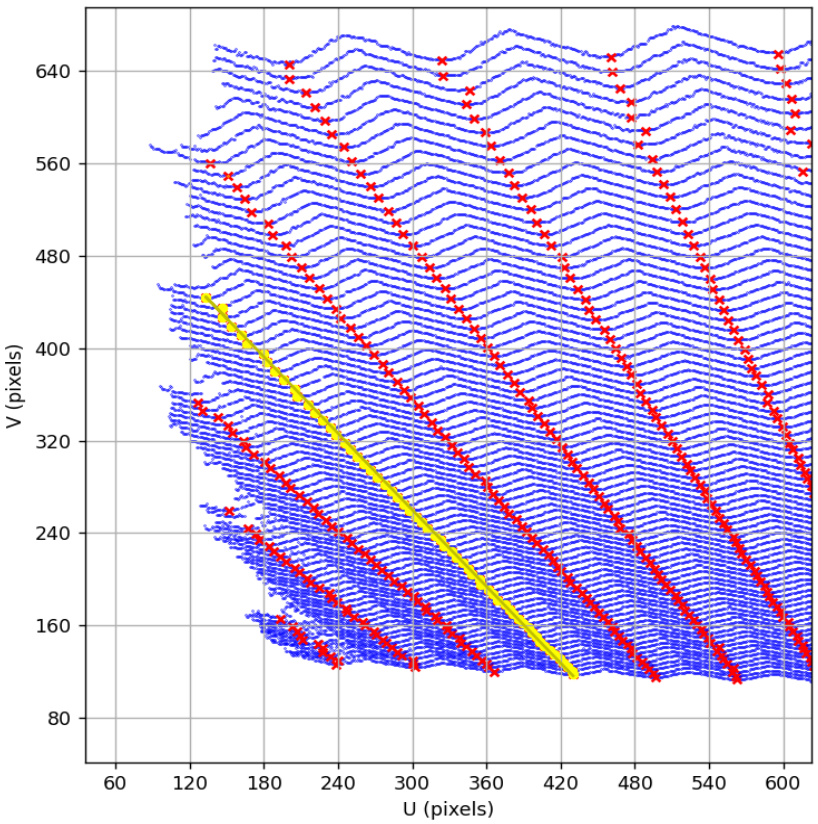
Prije iteriranja kroz sve profile moraju se kreirati liste i rječnici koji će nam dalje koristiti u kasnijim koracima. Jedna od tih lista je lista all_offsets koja predstavlja listu svih pozicija na kojima se bridovi mogu pojavljivati. To ujedno i predstavlja vrijednosti iz prvoga stupca iz ranije

spomenutog primjera skupa podataka. Good edges predstavlja rječnik u koji ćemo spremati najbolje inliere iz Ransacovog algoritma. Koristeći for petlju iteriramo kroz sve offsete koje smo ranije definirali. Kod svakog profila središnji brid imat će uvijek offset nula dok će brid desno od njega imati offset 20 i tako do 220. Koristeći drugu for petlju iteriramo kroz rječnik all-edges što predstavlja rječnik sa svim bridovima svakog profila.

U svakome profilu kada će se pojaviti offset brid kojemu pridružujemo vrijednost -220 što ujedno i predstavlja brid koji je najviše lijevo njegova pozicija će se dodati u listu extracted_numbers_x, a njegova vrijednost će se pamtiti u extracted_numbers_y.

Te liste će se proširivati za svaki profil svaki put kad vrijednost offseta bude -220 te će se nakon toga resetirati te će se proširivati sa offsetima -200 i tako sve do +220.

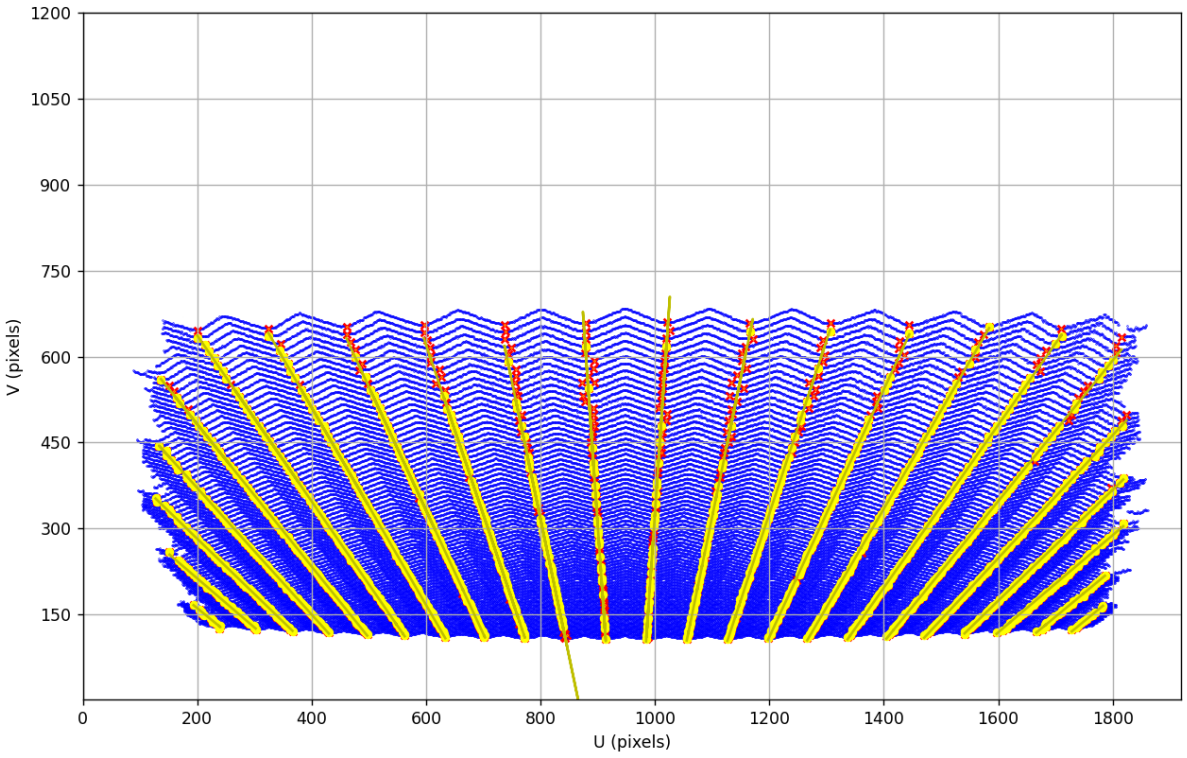
Te dvije liste koristimo kao ulazne vrijednosti za RANSAC algoritam koji u svakoj iteraciji računa nove inliere za zadani offset.



Slika 18. Primjena RANSAC algoritma za offset -160

RANSAC algoritam za generirani skup točaka za zadani offset računa sve inliere te izbacuje njihove pozicije. Na Slika 18. su inlieri označeni žutom bojom te je algoritam primijenjen samo za bridove kojima je pridružen offset -160.

Kada se RANSAC algoritam primijeni za sve bridove uz dopušteni grešku od 5 dobiju se rezultati vidljivi na Slika 19.:



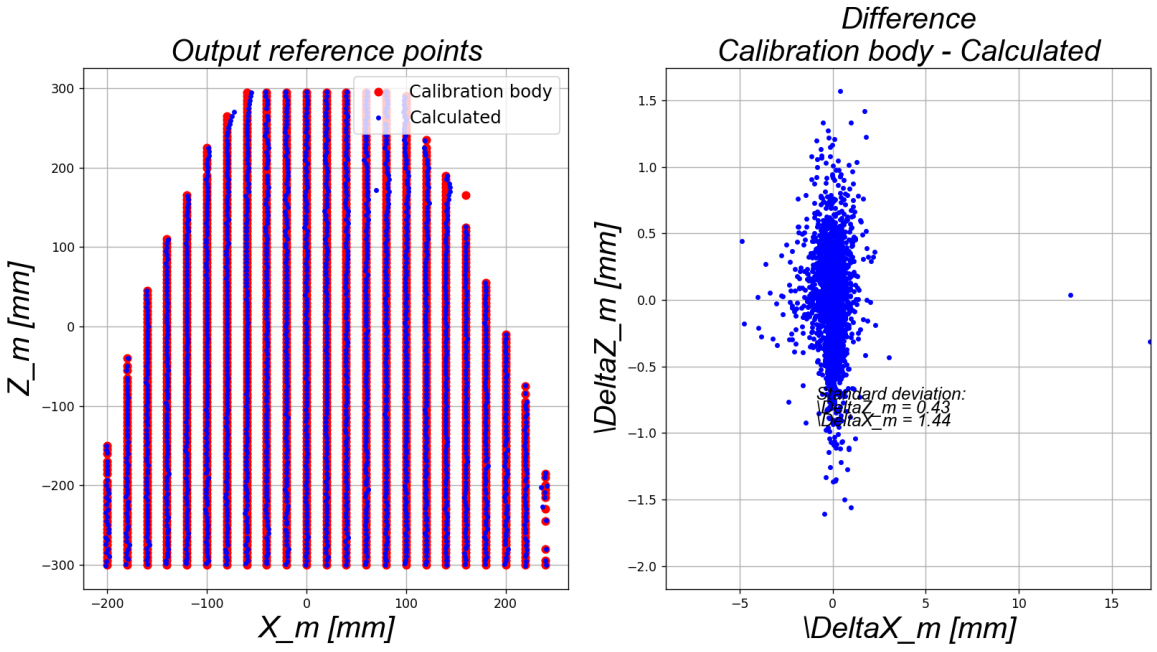
Slika 19. Primjena RANSAC algoritma za sve offsete

Svi dobiveni podaci nakon primjene RANSAC algoritma spremaju se u novi txt file koji je dobiven uspoređivanjem sa starom txt datotekom te izbacivanjem na temelju dobiveni outliera.

6. EVALUACIJA REZULTATA TEMELJENA NA TRENIRANJU NEURONSKE MREŽE

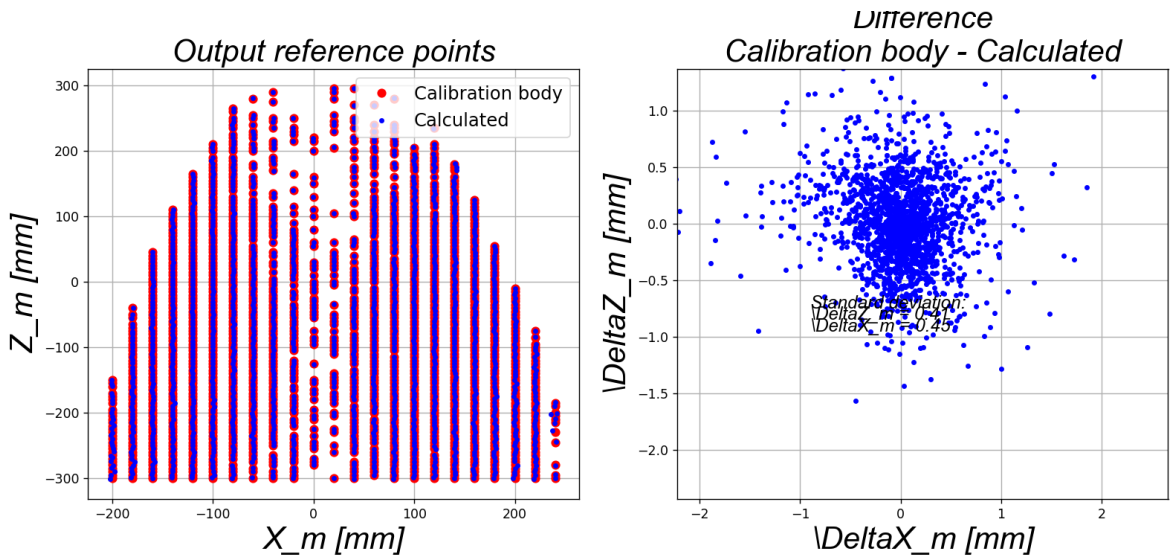
Posljednji korak ovog rada je testiranje uspješnosti kalibracije uz pomoć neuronske mreže u Pythonu. Neuronska mreža sadrži jedan skriveni sloj sa 17 neurona te se trenira pomoću Levenberg-Marquardt algoritma s maksimalno 1000 iteracija. Glavni cilj neuronske mreže je istrenirati sebe da bi na temelju poznatih 3D udaljenosti kalibracijskog tijela od lasera uspješno mapirala laser i kameru. U konačnici se ishod trenirane mreže uspoređuje s onim koje smo pronašli sa lokalizacijskim algoritmom. Veća devijacija greške istrenirane mreže pojavljuje se u slučajevima kada je veći šum kod snimanja kalibracijske ploče i posljedično greške lokalizacije vrhova na slici. Evaluacija rezultata rađena je na tri seta podataka. Prvi set ujedno je i najlošiji set te sadrži 120 mjerenja sa relativno puno šumova. Drugi set podataka također ima 120 mjerenja ali ima manje šumova te je točniji od prvoga. Prva dva seta podataka snima su u CRTI dok je treći set podataka sniman u Ljubljani a snimio ga je profesor dr.sc.Drago Bračun te taj set ujedno predstavlja i najtočniji set podataka uz najmanje šumova

Rezultati dobiveni bez primjene RANSAC algoritma na prvome setu podatak vidljivi su na Slika 21.



Slika 20.Rezultati bez RANSAC algoritma

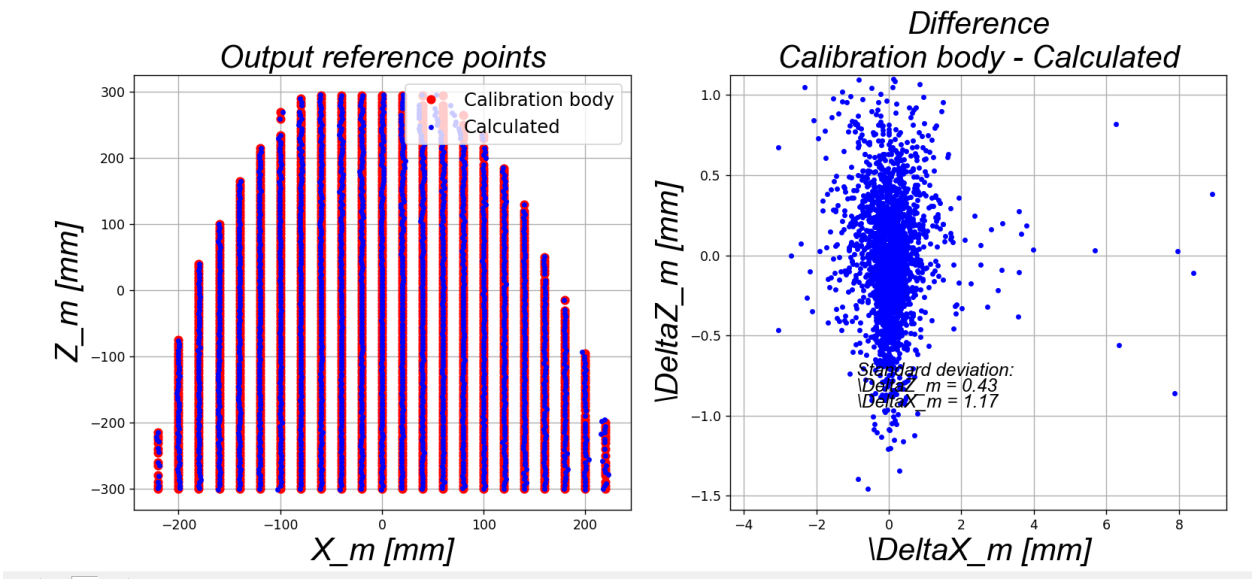
Na Slika 20. je vidljivo da su devijacije u smjeru vertikalne osi 0.43mm te smjeru horizontalne osi 1.44 mm što je relativno velika devijacija ako uzmemo u obzir da je razmak između bridova 20 mm. Do tako velike devijacije dolazi zbog ranije spomenutih grešaka u mjerenju.



Slika 21. Rezultati dobiveni primjenom RANSAC algoritma

Primjenom RANSAC algoritma koji izbacuje odstupanja koja su se pojavila prilikom mjerenje dobiva se rezultat vidljiv na Slika 21. Odstupanja u smjeru vertikalne osi dz su manja te ona sad iznose 0.41 mm dok odstupanja u smjeru horizontalne osi dx su također manja te ona sad iznose 0.45 mm.

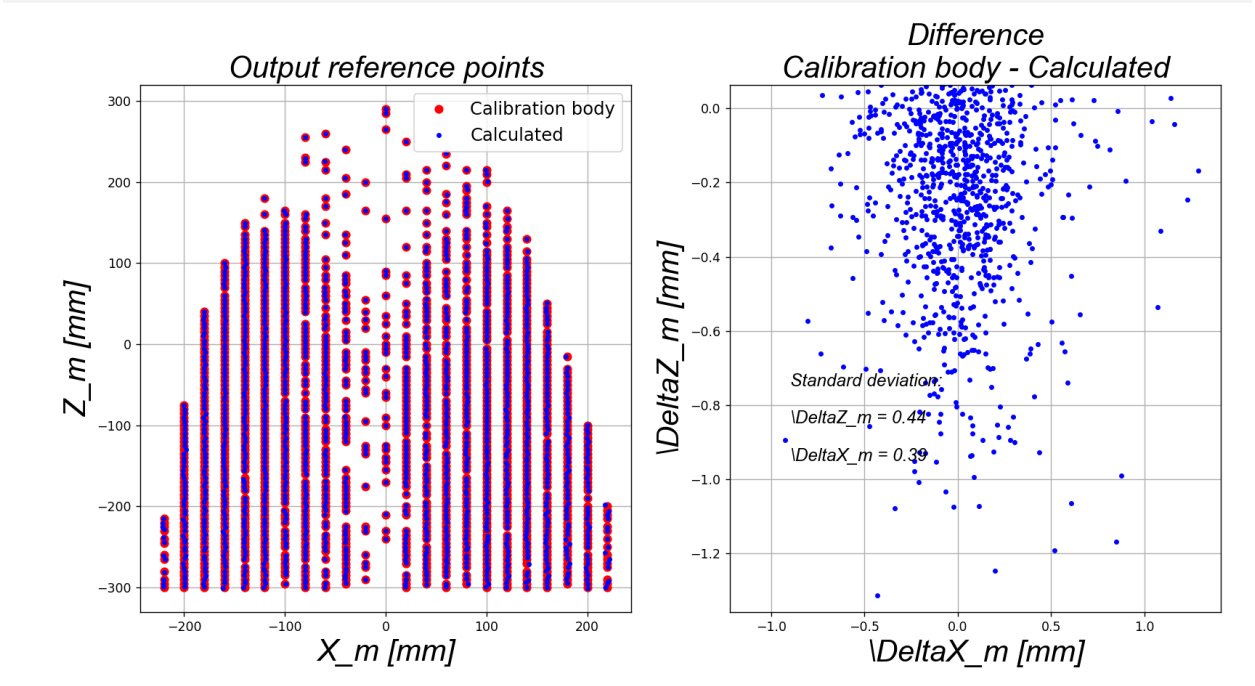
Rezultati dobiveni na drugome setu podataka bez primjene RANSAC algoritma vidljivi su na Slika 22.



Slika 22. Drugi set podataka, bez RANSAC algoritma

Na drugome setu podataka može se primijetiti i bez primjene RANSAC algoritma da su rezultati bolji tj devijacije manje. Devijacija u smjeru vertikalne osi sada iznosi 0.43 mm dok u smjeru horizontalne osi sada iznosi 1.17mm.

Primijenimo li RANSAC algoritam na drugi set podataka dobiju se rezultati vidljivi na Slika 23.

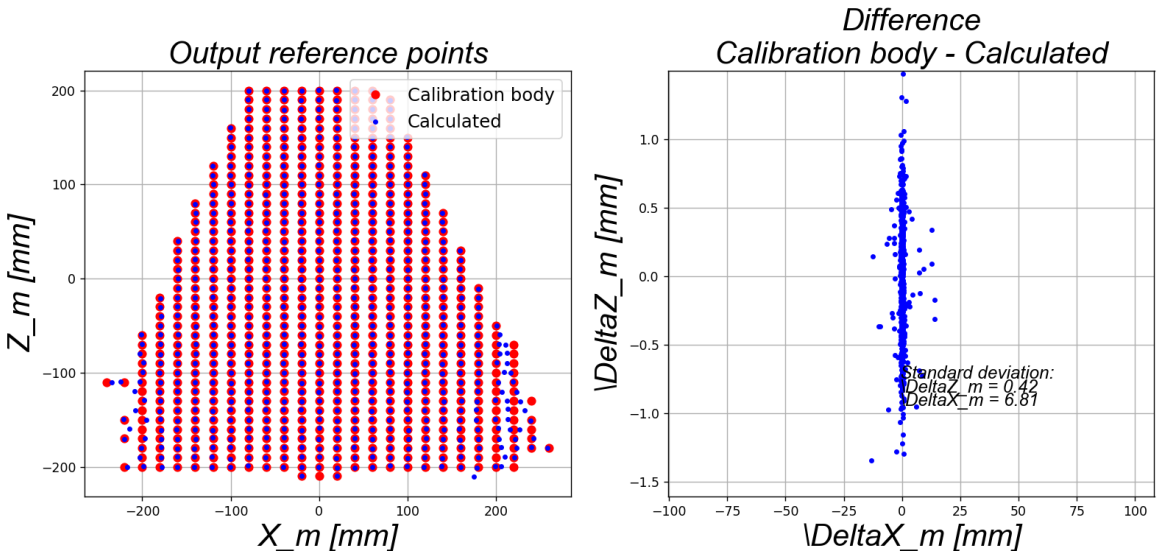


Slika 23. Drugi set podataka, uz primijenjen RANSAC algoritam

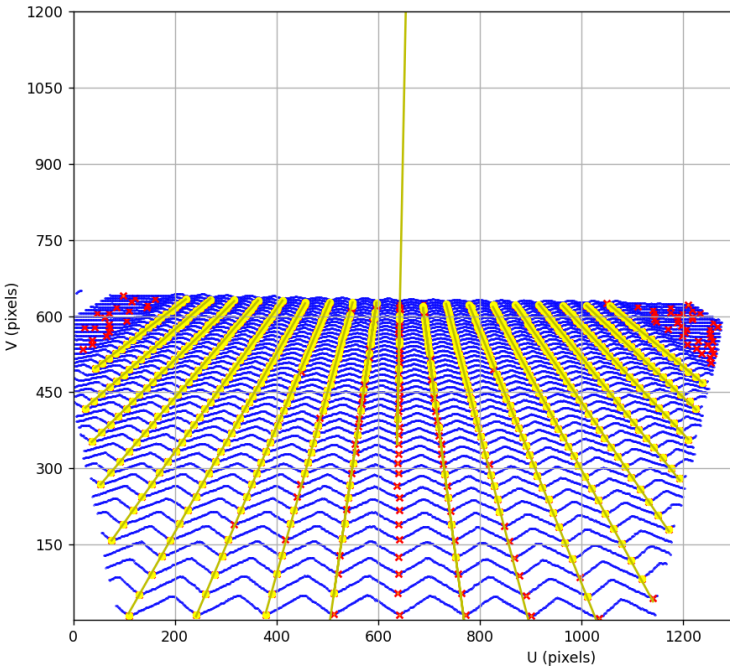
Rezultati dobiveni na trećem setu podataka snimljenih u Ljubljani mogu se vidjeti na Slika 24.

I bez primjene RANSAC algoritma devijacija u smjeru vertikalne osi iznosi 0.42 dok devijacija u smjeru horizontalne osi iznosi 6.81 mm. Razlog tako velike devijacije u smjeru horizontalne osi su točke na rubovima koje zadovoljavaju uvjete za lokalne minimume ali to zapravo nisu.

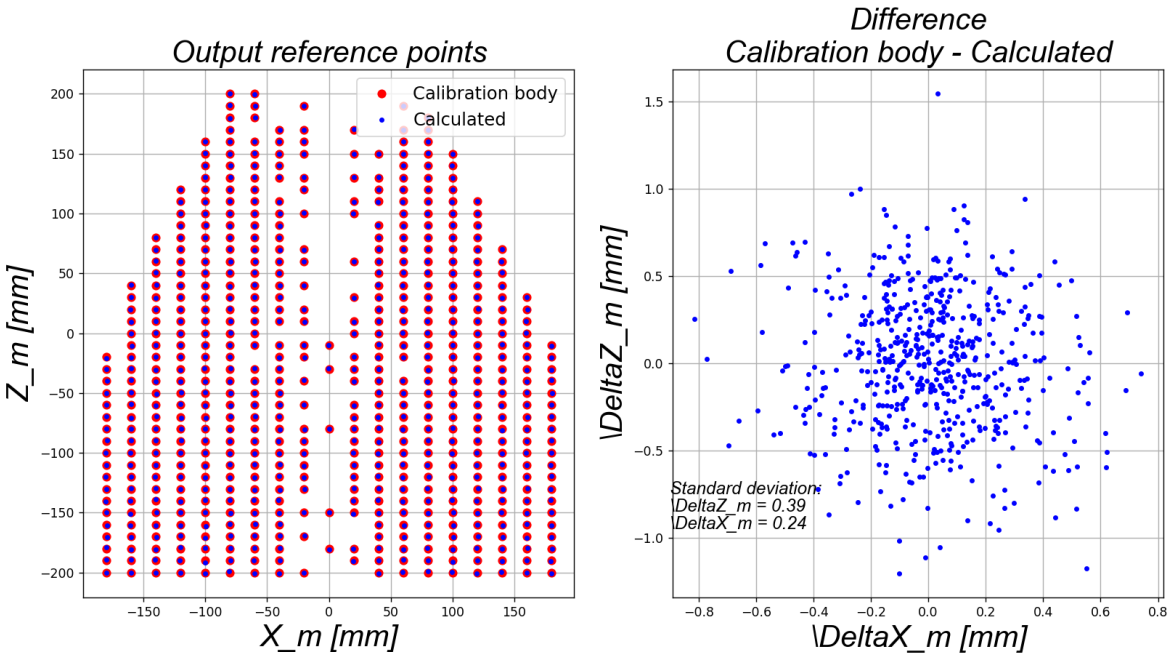
Te točke vidljive su na Slika 25. gdje su označene crvenom bojom kao i outlieri koji su pronađeni.



Slika 24. Treći set podataka bez RANSAC algoritma



Slika 25. Prikaz podataka iz trećeg seta



Slika 26. Treći set podataka uz RANSAC algoritam

Primjenom RANSAC algoritma na treći set podataka dobivaju se devijacija u smjeru horizontalne osi od 0.39 mm te devijacija u smjeru horizontalne osi od 0.24 mm. Na trećem setu podataka može se uočiti i jedna od mana RANSAC algoritma a to je da on ne radi kad treba pronaći jednadžbu pravca koji je okomit na horizontalnu os. Razlog tome je taj što RANSAC algoritam ne može pronaći točnu jednadžbu pravca koji ima oblik x=n gdje n predstavlja neki broj.

7. ZAKLJUČAK

U ovom završnom radu obrađen je postupak kalibracije 3D-vizijskog sustav laser kamera. Opisan je način na koji se dobivaju podatku te kako se ti podatci obrađuju .Detaljno je opisan algoritam koji lokalizira bridove te su istaknute neke njegove ključne funkcije koje čine njegovu učinkovitost. Objašnjen je način na koji funkcionira RANSAC algoritam te je prikazana njegova praktična primjena na setu stvarnih izmjerenih podataka. Također je prikazan način na koji algoritam ispisuje podatke. Na kraju analizirali smo rezultate koje smo postigli korištenjem neuronske mreže koja koristi ranije spomenute ispisane podatke .

Ovaj završni rad uspješno je razradio postupak kalibracije 3D-vizijskog sustava laser kamera istražujući njegove ključne komponente i funkcionalnosti. RANSAC algoritam koji je temeljno objašnjen pokazao se kao vrijedan alat za identifikaciju i uklanjanje potencijalnih grešaka u podacima, poboljšavajući preciznost kalibracije. Ispis i prikaz podataka iz algoritma pružio je važan uvid u njegov način funkcioniranja te prikaz izlaznih rezultata.

Rezultati temeljeni na treniranju neuronske mreže bili su jako blizu rezultatima dobivenim algoritmom za kalibraciju što pokazuje njegovu funkcionalnost i točnost.

Jedan od mogućih načina unapređenja postupka kalibracije u budućnosti bio bi optimizacija algoritma za traženje lokalnih minimuma uz traženje lokalnih minimuma nekom drugom metodom. Osim RANSAC algoritma postoje i neki drugi algoritmi za identifikaciju inliera koji nisu spomenuti u ovome završnome radu te bi neki od njih mogli još dodatno poboljšati postupak kalibracije, a neki od njih su MSAC, LMedS, MSER i PROSAC.

LITERATURA

[1] Python for begginers-Full course- freeCodeCamp.org-

https://www.youtube.com/watch?v=eWRfhZUzrAc

[2] Harvard CS50's Introduction to Programming with Python – Full University Course https://www.youtube.com/watch?v=nLRL_NcnK-4

[3] Camera Instrinsics and Extrinsics- Cyrill Stachniss

https://www.youtube.com/watch?v=ND2fa08vxkY

[4] Random Sample Consensus Explained, https://www.baeldung.com/cs/ransac

[5] RANSAC regression Explained with Python Examples- Ajtesh Kumar-

https://vitalflux.com/ransac-regression-explained-with-python-examples/

[6] RANSAC- Cyrill Stachniss - https://www.youtube.com/watch?v=9D5rrtCC_E0

PRILOZI

Kompletan Python kod:

```
import os
import numpy as np
import matplotlib.pyplot as plt
from collections import OrderedDict


# Folder from where we take data,  watch out for \\
folder_path = (
    "C:\\Users\\HN\\Documents\\Calibration_data\\draser_calib_2022_05_20\\Data"
)
# Folder where we print data
output_path =
"C:\\Users\\HN\Documents\\Calibration_data\\draser_calib_2022_05_20"
# Folders where pictures are saved
output_path1 =
"C:\\Users\\HN\Documents\\Calibration_data\\draser_calib_2022_05_20\\Output_slike
-bolji"
output_path2 =
"C:\\Users\\HN\Documents\\Calibration_data\\draser_calib_2022_05_20\\Output_slike
-bolji-ukupno"


### creating profile list ###
def list_with_profiles(location):
    profile_list = []
    file_extension = ".prf"
    for file_name in os.listdir(folder_path):
        if file_name.endswith(file_extension):
            profile_list.append(file_name)

    def extract_number(
        filename,
    ):  # function to get profile number by which we sort them
        return int("".join(filter(str.isdigit, filename)))

    sorted_files = sorted(profile_list, key=extract_number)
    return sorted_files
a = 20
br2 = 0

### From list of folders we only take number before semicolon###
def before_semicolon(list):
    list_with_data = [int(broj.split(",")[0]) for broj in list]
    return list_with_data
```

```
### Edge searching ###
def local_minimum(profil):
    local_minimums = {}
    dict_with_all_profiles = {}
    for i in range(len(profil)):
        dict_with_all_profiles[i + 1] = profil[
            i
        ]  # creating a dict with all profiles
        if (
            profil[i] != 0
            and profil[i] < profil[i - 1]
            and profil[i] < profil[i - 2]
            and profil[i] < profil[i - 3]
            and profil[i] < profil[i - 4]
            and profil[i] < profil[i - 10]
            and profil[i] <= profil[i + 1]
            and profil[i] <= profil[i + 2]
            and profil[i] <= profil[i + 3]
            and profil[i - 1] > 1
            and profil[i + 1] > 1
            and profil[i] <= profil[i + 10]
        ):
            local_minimums[i + 1] = profil[i]

    ### centering if we have multiple points with same value ###
    def centering(dict_all_points, dict_minimums):
        global br
        keys_to_delete = []
        sum = 0
        counter = 0
        keys_to_add = []
        last_added_key = 1
        last_value = None
        for key1, value1 in dict_minimums.items():
            counter += 1
            sum += key1
            if key1 in dict_all_points:
                key2 = key1
                count = 0
                while key2 in dict_all_points and (
                    dict_all_points[key2] == value1
                    or (
                        key2 + 1 in dict_all_points
                        and dict_all_points[key2 + 1] == value1
                    )
                    or (
                        key2 + 2 in dict_all_points
                        and dict_all_points[key2 + 2] == value1
                    )
                    or (
                        key2 + 3 in dict_all_points
```

```
                        and dict_all_points[key2 + 3] == value1
                    )
                    or (
                        key2 + 4 in dict_all_points
                        and dict_all_points[key2 + 4] == value1
                    )
                ):
                    count += 1
                    key2 += 1
                if count > 0:
                    interval_middle = (key1 + key2 - 1) / 2.0
                    if interval_middle - last_added_key <= 10:
                        last_added_key = interval_middle
                        last_value = value1
                    keys_to_add.append((interval_middle, value1))
                    keys_to_delete.append(key1)
        bad_keys = []
        space = keys_to_add[1][0] - keys_to_add[0][0]
        for i in range(len(keys_to_add) - 1):
            temporary_key, temporary_value = keys_to_add[i]
            next_key, next_value = keys_to_add[i + 1]
            if next_key - temporary_key < 20 and temporary_key != next_key:
                if next_value > temporary_value:
                    bad_keys.append(next_key)
                else:
                    bad_keys.append(temporary_key)
        for keyz in keys_to_delete:
            if keyz in dict_minimums:
                del dict_minimums[keyz]
        for keey, valuee in keys_to_add:
            if keey not in bad_keys:
                dict_minimums[keey] = valuee
        return dict_minimums
    output_dict = centering(dict_with_all_profiles, local_minimums)

    def removing_wrong_edges(dct):
        global a
        keys = list(dct.keys())
        keys_to_delete = []
        for i in range(1, len(keys)):
            last_key = keys[i - 1]
            current_key = keys[i]

            if current_key - last_key < a:
                if dct.get(last_key, 0) > dct.get(current_key, 0):
                    keys_to_delete.append((last_key, dct[last_key]))
                else:
                    keys_to_delete.append((current_key, dct[current_key]))
        for key_x, key_value in keys_to_delete:
            if key_x in dct:
                del dct[key_x]
```

```
            elif key_value in dct.values():
                keys_to_delete = [
                    key for key, value in dct.items() if value == key_value
                ]
                for key in keys_to_delete:
                    del dct[key]
        return dct
    output_dict = removing_wrong_edges(output_dict)
    utput_dict = removing_wrong_edges(output_dict)



    return output_dict



### Drawing graphs ###
all_profiles1 = []
all_bridges = []
br = 1
def drawing(input_data, input_dict):
    global br
    x_cord = list(range(1, len(input_data) + 1))
    y_cord = input_data
    fig, ax = plt.subplots(
        figsize=(12, 9)
    )  # Set the figure size to maintain the aspect ratio
    ax.scatter(x_cord, y_cord, s=10)  # Blue line for profile data
    # Draw 'x' symbols using dots
    x_edges = list(input_dict.keys())
    y_edges = list(input_dict.values())
    ax.scatter(
        x_edges, y_edges, marker="x", color="red", s=15
    )  # Draw 'x' symbols using dots
    ax.set_xlim(0, 1920)  # Set x-axis limits
    ax.set_ylim(1, 1200)  # Set y-axis limits
    ax.set_xlabel("U (pixels)")
    ax.set_ylabel("V (pixels)")
    ax.grid(True)
    ax.set_aspect("equal")  # Set 1:1 aspect ratio
    ax.xaxis.set_major_locator(plt.MaxNLocator(integer=True))
    ax.yaxis.set_major_locator(plt.MaxNLocator(integer=True))
    all_profiles1.append(input_data)
    all_bridges.append(input_dict)
    filename = f"slika{br}.png"
    full_path = os.path.join(output_path1, filename)
    plt.savefig(full_path)
    plt.close()
    #plt.show()
    br += 1
```

```
### OUTPUT ###
def write_dict_keys_to_txt(dictionary, filename):
    with open(filename, "w") as file:
        for key in dictionary.keys():
            file.write(str(key) + "\n")
all_edges = {}
def find_closest_key(dict, target_value):
    closest_key = min(dict, key=lambda x: abs(x - target_value))

    return closest_key

### Caliing all functions and drawings ###
offset_increment = -20
offset_increment1 = -10
spacing_width = 14
all_profiles = []

for file_name in list_with_profiles(folder_path):
    file_path = os.path.join(folder_path, file_name)
    with open(file_path, "r") as f:
        content = f.readlines()
        content = before_semicolon(content)
        all_profiles.append(content)
    br2 += 1
    edges = local_minimum(content)

    drawing(content, edges)
    if br2 > 40:
        a = 30
    if br2 > 40:
        a = 60
    if br2 > 100:
        a = 100
        ##### resolution/2 #############



    all_edges[file_name] = edges
output_filename = os.path.join(output_path, "ispis_MM.txt")
write_dict_keys_to_txt(all_edges, output_filename)
lista_za_usporedbu = []


def ransac_polyfit(x, y, degree=1, threshold=2.0, iterations=100):
    best_inliers = []
    best_coeffs = None
    for _ in range(iterations):
        random_indices = np.random.choice(len(x), degree + 1, replace=False)
        x_sample = x[random_indices]
```

```
        y_sample = y[random_indices]
        coeffs = np.polyfit(x_sample, y_sample, degree)
        poly = np.poly1d(coeffs)
        y_pred = poly(x)
        errors = np.abs(y - y_pred)
        inliers = np.where(errors < threshold)[0]
        if len(inliers) > len(best_inliers):
            best_inliers = inliers
            best_coeffs = coeffs
    return best_coeffs, best_inliers


def create_combined_plot(
    profile_data_list,
    bridovi_list,
    all_x1,
    all_y1,
    all_inliers1,
    all_coeffs1,
    output_path,
):
    fig, ax = plt.subplots(
        figsize=(12, 9)
    )  # Set the figure size to maintain the aspect ratio
    for ulazni_podaci_profila, rjecnik in zip(profile_data_list, bridovi_list):
        x_kord = list(range(1, len(ulazni_podaci_profila) + 1))
        y_kord = ulazni_podaci_profila
        ax.scatter(x_kord, y_kord, color="blue", s=0.1)  # Blue line for profile
data
        # Draw 'x' symbols using dots
        x_bridova = list(rjecnik.keys())
        y_bridova = list(rjecnik.values())
        ax.scatter(
            x_bridova, y_bridova, marker="x", color="red", s=20
        )  # Draw 'x' symbols using dots

    for i, offset_x in enumerate(all_offsets):
        x = all_x1[i]
        y = all_y1[i]
        inliers = all_inliers1[i]
        coeffs = all_coeffs1[i]

        plt.scatter(x, y, s=0.1, label=f"Offset {offset_x} Data", color="yellow")
        plt.scatter(x[inliers], y[inliers], label=f"Offset {offset_x} Inliers",
s=20, color="yellow")
        plt.plot(
            x,
            np.poly1d(coeffs)(x),
            label=f"Offset {offset_x} Fitted Polynomial",
            color="y",
        )
```

```
    ax.set_xlim(0, 1920)  # Set x-axis limits
    ax.set_ylim(1, 1200)  # Set y-axis limits
    ax.set_aspect("equal")  # Set 1:1 aspect ratio
    ax.set_xlabel("U (pixels)")
    ax.set_ylabel("V (pixels)")
    ax.grid(True)
    ax.set_aspect("equal")  # Set 1:1 aspect ratio
    ax.xaxis.set_major_locator(plt.MaxNLocator(integer=True))
    ax.yaxis.set_major_locator(plt.MaxNLocator(integer=True))
    full_path = os.path.join(output_path2, "combined_plot.png")
    ax.set_xlim(0, 1920)  # Set x-axis limits
    ax.set_ylim(1, 1200)  # Set y-axis limits
    ax.set_aspect("equal")  # Set 1:1 aspect ratio
    ax.set_xlabel("U (pixels)")
    ax.set_ylabel("V (pixels)")
    ax.grid(True)
    ax.set_aspect("equal")  # Set 1:1 aspect ratio
    ax.xaxis.set_major_locator(plt.MaxNLocator(integer=True))
    ax.yaxis.set_major_locator(plt.MaxNLocator(integer=True))

    # plt.legend()
    plt.savefig(full_path)
    plt.show()


all_edges_output_filename = os.path.join(output_path, "ispis_MM.txt")
all_extracted_numbers_x = []
all_extracted_numbers_y = []
all_x = []
all_y = []
all_inliers = []
all_coeffs = []
all_offsets =list(range(-180,200,20))
good_edges = {}
offset_count_list = []

for offset_x in all_offsets:
    extracted_numbers_x = []
    extracted_numbers_y = []
    offset_count = 0
    with open(
        all_edges_output_filename, "w"
    ) as all_bridovi_file:
        target_value=640
        for idx, (file_name, bridovi_dict) in enumerate(all_edges.items()):
            closest_key = find_closest_key(bridovi_dict, target_value)
            target_value=closest_key

            keys = list(bridovi_dict.keys())
            second_number = (200) + idx * offset_increment1
```

```
            for i, key in enumerate(keys):
                offset = offset_increment * (keys.index(closest_key) - i)
                position = i * 10
                if offset == offset_x:
                    extracted_numbers_x.append(key)
                    extracted_numbers_y.append(bridovi_dict[key])
                    offset_count += 1
                offset_pair = f"{offset:<5}{second_number:<5}"  # Increase
spacing width
                formatted_line = (
                    f"{offset_pair}{key:>{spacing_width}}
{bridovi_dict[key]:.4f}\n"
                )
                formatted_line = " ".join(
                    formatted_line.split()
                )  # Ensure consistent spacing
                all_bridovi_file.write(formatted_line + "\n")
    x = np.array(extracted_numbers_x)
    y = np.array(extracted_numbers_y)
    best_coeffs, best_inliers = ransac_polyfit(
        x, y, degree=2, threshold=5.00, iterations=100
    )
    all_extracted_numbers_x.append(extracted_numbers_x)
    all_extracted_numbers_y.append(extracted_numbers_y)
    all_x.append(x)
    all_y.append(y)
    all_inliers.append(best_inliers)
    all_coeffs.append(best_coeffs)
    good_edges[offset_x] = list(best_inliers)
    offset_count_list.append(offset_count)
create_combined_plot(
    all_profiles1, all_bridges, all_x, all_y, all_inliers, all_coeffs,
output_path
)


# Define the input and output file names
input_file22 = os.path.join(output_path, "ispis_MM.txt")
output_file22 = os.path.join(output_path, "ispis_MM_novi.txt")


# Load the treba_maknuti dictionary


total_sum = 0  # Initialize the total sum
for (key, value), offset_count1 in zip(good_edges.items(), offset_count_list):
    # Add the total sum to each number in the list

    good_edges[key] = [x + total_sum for x in value]

    # Update the total sum with the length of the current list
```

```
    total_sum += offset_count1
# print("nakon", treba_maknuti)

for value12 in good_edges.values():
    lista_za_usporedbu.extend(value12)



list123 = []


def write_matching_rows(input_file, output_file, dictionary):
    with open(input_file, "r") as input_file, open(output_file, "w") as
new_output_file:
        for key in dictionary.keys():
            input_file.seek(
                0
            )  # Reset the input file pointer to the beginning for each key
            for line in input_file:
                first_number = int(
                    line.split()[0]
                )  # Extract the first number from the line
                if first_number == key:
                    list123.append(line)

        for index in lista_za_usporedbu:
            if 0 <= index < len(list123):
                new_output_file.write(list123[index])



# Use the write_matching_rows function to append additional matching rows
write_matching_rows(input_file22, output_file22, good_edges)
```

Neuronska mreža kod u Pythonu:

```
import numpy as np
import datetime
import matplotlib.pyplot as plt
import pyrenn as prn
from sklearn.model_selection import train_test_split
import pickle

# Load the data
data =
np.genfromtxt(r'C:\Users\HN\Documents\Calibration_data\draser_calib_2022_05_20\is
pis_MM_novi.txt', delimiter=' ', dtype=float)
#data = np.genfromtxt(r'C:\Users\HN\Documents\FAKULTET\7.SEMESTAR\ZAVRŠNI
RAD\pyren_LM_train\tockem.txt', delimiter=' ', dtype=float)
X = data[:, -2:]  # All rows, last two columns
```

```
Y = data[:, :2]  # All rows, first two columns

#print(X)
#print(Y)

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.1,
random_state=42)

# Create the model with 1 hidden layer with 17 neurons
net = prn.CreateNN([2, 17, 2])

# Train the model using Levenberg-Marquardt algorithm
net = prn.train_LM(X_train.T, y_train.T, net, verbose=True, k_max=1000,
E_stop=1e-5)

# Test the model
y_test_est = prn.NNOut(X_test.T, net)
y_train_est = prn.NNOut(X_train.T, net)

# Calculate MSE for testing and training data
mse_test = np.mean((y_test.T - y_test_est)**2)
mse_train = np.mean((y_train.T - y_train_est)**2)

# Calculate MAE for testing and training data
mae_test = np.mean(np.abs(y_test.T - y_test_est))
mae_train = np.mean(np.abs(y_train.T - y_train_est))

print(f"Mean Square Error on test set: {mse_test}")
print(f"Mean Square Error on train set: {mse_train}")
print(f"Mean Average Error on test set: {mae_test}")
print(f"Mean Average Error on train set: {mae_train}")

timestamp = datetime.datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
filename = f"{timestamp}.txt"

with open(filename, "w") as file:
    file.write(f"Mean Square Error on test set: {mse_test}\n")
    file.write(f"Mean Square Error on train set: {mse_train}\n")
    file.write(f"Mean Average Error on test set: {mae_test}\n")
    file.write(f"Mean Average Error on train set: {mae_train}\n")

print(f"Text data saved to {filename}")

# Save the trained network
with open(f"network_{timestamp}.pkl", "wb") as f:
    pickle.dump(net, f)


y_est = prn.NNOut(X.T, net)
# Assuming that 'xd' and 'yd' are X_train and 'nx' and 'ny' are the estimated
outputs 'y_train_est'
```

```
xd = Y[:, 0]
yd = Y[:, 1]
nx = y_est[0, :]
ny = y_est[1, :]

# Assuming dx and dy are the differences (errors) between actual and estimated
outputs
dx = Y[:, 0] - y_est[0, :]
dy = Y[:, 1] - y_est[1, :]

# Standard deviations
stddifx = np.std(dx)
stddify = np.std(dy)

fig, axs = plt.subplots(1, 2, figsize=(15, 6))

# Subplot 1
axs[0].plot(xd, yd, 'ro', label='Calibration body')
axs[0].plot(nx, ny, 'b.', label='Calculated')
axs[0].set_title('Output reference points', fontname='Arial', fontsize=24,
fontstyle='italic')
axs[0].set_xlabel('X_m [mm]', fontname='Arial', fontsize=24, fontstyle='italic')
axs[0].set_ylabel('Z_m [mm]', fontname='Arial', fontsize=24, fontstyle='italic')
axs[0].legend(fontsize=14, loc='upper right')
axs[0].grid(True)

# Subplot 2
axs[1].plot(dx, dy, 'b.')
axs[1].set_xlabel('\DeltaX_m [mm]', fontname='Arial', fontsize=24,
fontstyle='italic')
axs[1].set_ylabel('\DeltaZ_m [mm]', fontname='Arial', fontsize=24,
fontstyle='italic')
axs[1].set_title('Difference\nCalibration body - Calculated', fontname='Arial',
fontsize=24, fontstyle='italic')
axs[1].text(-0.9, -0.95, f'\DeltaX_m = {stddifx:.2f}', fontname='Arial',
fontsize=14, fontstyle='italic')
axs[1].text(-0.9, -0.85, f'\DeltaZ_m = {stddify:.2f}', fontname='Arial',
fontsize=14, fontstyle='italic')
axs[1].text(-0.9, -0.75, 'Standard deviation:', fontname='Arial', fontsize=14,
fontstyle='italic')
axs[1].grid(True)

plt.show()
```