Mapiranje karakterističnih točaka ljudskog lica koristeći strojno učenje te Blender programsku podršku

Gregov, Petra

Undergraduate thesis / Završni rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Mechanical Engineering and Naval Architecture / Sveučilište u Zagrebu, Fakultet strojarstva i brodogradnje**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:235:038887

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2025-03-07**

Repository / Repozitorij:

Repository of Faculty of Mechanical Engineering and Naval Architecture University of Zagreb

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Petra Gregov

Zagreb, 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentor:

doc. dr. sc. Tomislav Stipančić, dipl. ing.

Student:

Petra Gregov

Zagreb, 2022.

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 / 22 – 6 / 1**	
Ur.broj: **15 - 1703 - 22 -**	

ZAVRŠNI ZADATAK

Student: **Petra Gregov** JMBAG: **0035215291**

Naslov rada na hrvatskom jeziku: **Mapiranje karakterističnih točaka ljudskog lica koristeći strojno učenje te Blender programsku podršku**

Naslov rada na engleskom jeziku: **Mapping of characteristic points of the human face using machine learning and Blender software**

Opis zadatka:

Suvremene tehnike umjetne inteligencije te vizualizacije informacija omogućuju njihov prikaz tako da kontekstualno i intuitivno budu lakše percipirane od strane ljudi. Računalna analiza ljudskog lica omogućuje učenje, identifikaciju te lokalizaciju karakterističnih točaka koje su zajedničke svim promatranim licima. Prateći te točke moguće ih je povezati s točkama na licu animiranog virtualnog agenta.

U radu je potrebno:

- proučiti metode umjetne inteligencije te identificirati prikladne Python programske biblioteke koje omogućuju učenje, identifikaciju i lokalizaciju karakterističnih točaka ljudskog lica,
- proučiti te primijeniti Blender programsku podršku,
- računalno povezati karakteristične točke na licu s videozapisa s karakterističnim točkama (armaturom) na licu virtualnog softverskog agenta.

Razvijenu aplikaciju 3D modela lica softverskog agenta eksperimentalno verificirati na humanoidnoj glavi afektivnog robota PLEA koja je dostupna u sklopu Laboratorija za projektiranje izradbenih i montažnih sustava.

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

Zadatak zadan:
30. 11. 2021.

Zadatak zadao:
Doc. dr. sc. Tomislav Stipančić

Datum predaje rada:
1. rok: 24. 2. 2022.
2. rok (izvanredni): 6. 7. 2022.
3. rok: 22. 9. 2022.

Predviđeni datumi obrane:
1. rok: 28. 2. – 4. 3. 2022.
2. rok (izvanredni): 8. 7. 2022.
3. rok: 26. 9. – 30. 9. 2022.

Predsjednik Povjerenstva:
Prof. dr. sc. Branko Bauer

Izjavljujem da sam ovaj rad izradila samostalno koristeći znanja stečena tijekom studija i navedenu literaturu.

Petra Gregov

SADRŽAJ

POPIS SLIKA

SAŽETAK

Tema ovog rada je istražiti metode umjetne inteligencije za detekciju i mapiranje karakterističnih crta ljudskog lica te implementirati iste u python-u i Blender-u radi animacije 3D modela. Najefikasnija takva metoda su konvolucijske neuronske mreže koje precizno razlučuju obrasce na datim slikama. U prvom dijelu rada se razradio njihov princip rada te prikladne python biblioteke. Dok se u drugom dijelu rada opisao proces dodavanja i namještanja 3D modela, te način animiranja istih unutar Blender grafičkog softvera. Animiranje se izvelo u python kodu pomoću OpenPose biblioteke.

Ključne riječi: Python, Blender, 3D model, karakteristične crte lica, OpenPose, CNN

SUMMARY

The topic of this paper is to investigate artificial intelligence methods for the detection and mapping of characteristic features of the human face and their implementation in Python and Blender for the animation of 3D models. The most efficient such a method are convolutional neural networks that accurately distinguish patterns in given images. Their working principle and suitable python libraries are explained in the first part of the work. While the second part of the paper shows how to add and adjust 3D models and how to achieve their animation within the Blender graphics software. The animation was performed in python code using the OpenPose library.

Keywords: Python, Blender, 3D model, characteristic facial features, OpenPose, CNN

1. UVOD

Računalna analiza ljudskog lica spada u široki spektar primjene – od animacije do sigurnosnih sustava. Trenutno postoje mnogi načini praćenja i analize karakterističnih crta lica, no razvojem metoda umjetne inteligencije, ti načini se vrlo brzo optimiziraju, mijenjaju i poboljšavaju.

U ovom radu će se prvo opisati razvoj modernih metoda umjetne inteligencije koje omogućavaju efikasno praćenje karakterističnih crta lica, a zatim će se opisati njihov princip rada i primjena. Osim toga, opisat će se programski jezik python i pripadajuće biblioteke te njihova korisnost i način primjene za spoj umjetne inteligencije i analize ljudskog lica.

U sklopu praktičnog dijela ovog rada, proučena je potrebna Blender podrška, odnosno načini animiranja 3D modela koristeći python i umjetnu inteligenciju.

2. METODE STROJNOG UČENJA ZA RAČUNALNU EKSTRAKCIJU KARAKTERISTIČNIH TOČAKA LJUDSKOG LICA

Umjetna inteligencija (eng. *artificial intelligence*) je, u svom najužem značenju, sposobnost neživih sustava da se snalaze u novim i nepoznatim situacijama što je ujedno i pojednostavljena definicija inteligencije [1]. Unatoč spominjanja sličnih koncepata još od antičke Grčke, umjetna inteligencija kao znanstvena disciplina se utemeljuje tek 1956. godine u Sjedinjenim Američkim Državama na konferenciji „Artificial Intelligence“ u Dartmouth-u. Tamo umjetna inteligencija dobiva svoju prvu službenu definiciju: *Istraživačka disciplina koja se temelji na pretpostavci da se svaki aspekt učenja ili bilo koje drugo obilježje inteligencije može načelno precizno opisati tako da se može izraditi stroj koji to simulira* [2].

Ipak, definiranje svrhe razvoja umjetne inteligencije će se s vremenom pokazati kao predmet rasprave što će rezultirati različitim ciljanim ishodima primjene umjetne inteligencije i time granama umjetne inteligencije. Tako 1970-tih nastaje računalni vid (eng. *computer vision*), grana umjetne inteligencije koja se bavi pronalaženjem metoda za dohvaćanje, obradu, analizu i razumijevanje višedimenzionalnih podataka (npr. slika) iz realnog svijeta. Cilj računalnog vida je naći način da računalo imitira ljudsku sposobnost vida, odnosno da brzo i efikasno razlučuje značenje vizualnih podataka analogno ljudskom razumijevanju. Detekcija i analiza ljudskog lica, što je tema ovog rada, je jedan od mnogih zadataka (problema) računalnog vida [3].

Algoritamsko rješenje za analizu digitalnih slika se pokazalo kao kompleksan i težak zadatak. Naime, slika kao ulazni podatak može iznimno varirati u kompleksnosti dok točnost interpretacije ovisi o količini definiranih atributa i parametara. Što znači da se broj potrebnih definiranih informacija povećava sa svakom promjenom osvjetljenja, boje, rotacije i translacije elemenata slike. S manjkom potrebnih podataka se gubi preciznost analize, dok s druge strane, dodavanje velikog broja definiranih podataka za analizu je zahtjevan i spor proces s obzirom da kvaliteta algoritma ovisi o njihovoj reprezentativnosti [4].

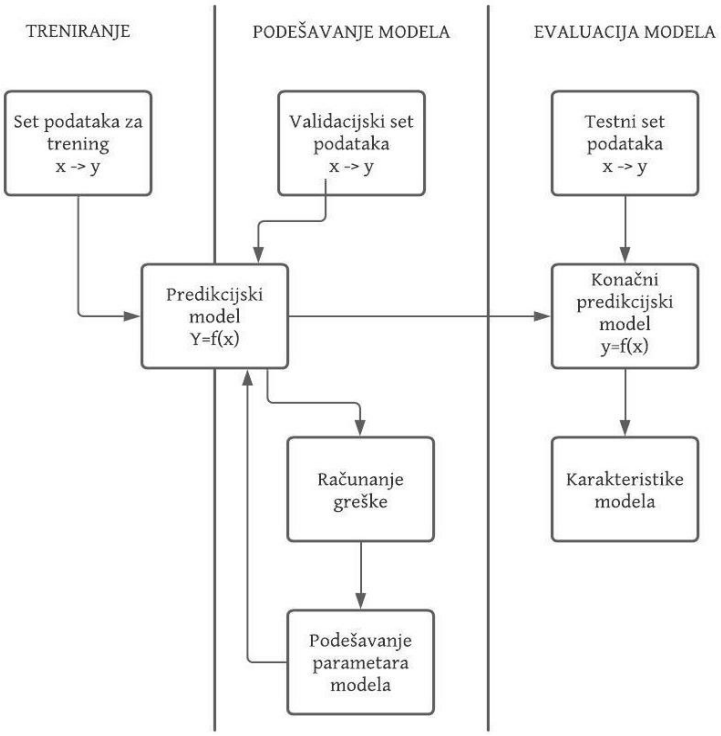
Uz računalni vid nastaje i strojno učenje (eng. *machine learning*), grana umjetne inteligencije s ciljem stvaranja računalnih algoritama koji su sposobni učiti, odnosno detektirati značajne obrasce iz datih podataka. Značajno u tom kontekstu znači ono što se želi detektirati [5]. Algoritmi koji samostalno definiraju parametre obrade vizualnih podataka su direktno rješenje

prethodno navedenog problema analize slika te metode strojnog učenja vrlo brzo postaju temelj mnogih zadataka računalnog vida.

Strojno učenje razlikuje tri tipa učenja s obzirom na stanje izlaznih podataka:

I. nadzirano učenje (eng. *supervised learning*) gdje su izlazni podaci zadani,
II. nenadzirano učenje gdje izlazni podaci nisu zadani (eng. *unsupervised learning*) i
III. polu-nadzirano učenje (eng. *semi-supervised learning*) gdje je dio izlaznih podataka zadan, a drugi dio nije [6].

Kako su tema ovog rada lice i karakteristične crte lica, koje su po svojoj prirodiveć definirani izlazni podaci u ovom radu će sve daljnje metode analize strojnog učenja biti prvog tipa, odnosno nadzirano učenje. Pojednostavljen proces nadziranog učenja započinje prikupljanjem klasificiranih podataka koji se koriste za učenje algoritma (eng. *training data set*); algoritam analizira dobivene podatke kreirajući model predikcije koji samostalno donosi odluke s novim podacima, zatim se pokušava minimizirati greška predikcije na način da se provjeri uspješnost modela na novom, validacijskom setu podataka (eng. *validation data set*). U skladu s novim informacijama, algoritam „uči“ i prilagođava parametre modela. Efikasnost, brzina izvođenja i drugi bitni parametri modela se ispituju na konačnom modelu i novom setu podataka (eng. *test data set*) [7] [8]. Proces je grafički prikazan na slici 1.

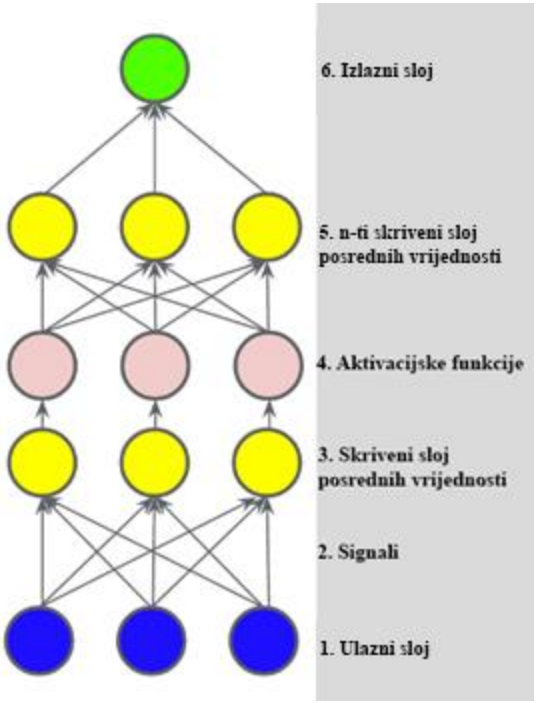


Slika 1 Dijagram nadziranog učenja, Izvor: Autor

Razvojem tehnologije i hardvera[1] (eng. *hardware*), otvaraju se vrata puno kompleksnijim metodama strojnog učenja zajedničkog naziva duboko učenje (eng. *deep learning*).

Još 1958. predstavljena je ideja algoritma koji funkcionira kao neuroni, živčane stanice mozga. Imitiranjem neurona i njihovog rada nastaje pojam umjetnih neuronskih mreža (eng. *artificial neural networks*, skraćeno ANN), točnije nelinearna obrada informacija hijerarhijski od jednostavnih do složenih.

[1] Računalna sklopovska podrška, sklopovlje, strojna oprema, strojevina



Slika 2 Pojednostavljen prikaz umjetne neuronske mreže, Izvor: Autor

Pojednostavljen prikaz ANN je vidljiv na slici 2. Plavi, žuti i zeleni krugovi su analogni neuronima, odnosno stanice koje sadržavaju neki električki potencijal. Taj potencijal se mijenja u ovisnosti o okolnim neuronima, točnije potencijal stanice se povećava ili smanjuje s obzirom na nove informacije. Donji sloj plavih krugova predstavlja ulazne podatke ili prepoznate elemente ulaznih podataka. Linije koje povezuju krugove označavaju signale koji su opisani numeričkim vrijednostima. Signali su imitacija sinapse, spojnog sredstva dvaju neurona. Sloj žutih krugova poznat je kao skriveni sloj (eng. *hidden layer*) jer se njegove vrijednosti i stanja ne mogu promatrati za razliku od ulaznog i izlaznog sloja. Prije ulaska u skriveni sloj, vrijednosti na signalima se množe težinskim faktorima. Dobivene težinske vrijednosti neurona se zbrajaju unutar skrivenih slojeva te pod uvjetom zadovoljavaju li zadani prag, neuron dalje odašilje signal. Odnosno, svi ucrtani signali na slici se ne moraju dogoditi (ako sumirane težinske informacije ne dosižu zadani prag), na taj način se neke manje bitne i nebitne informacije ne prenose, s čime se smanjuje njihov utjecaj na krajnji rezultat. Ukratko, skriveni slojevi služe kao filter informacija koje naposljetku konvergiraju prema izlazu. Proces se ponavlja s obzirom na broj skrivenih slojeva dok ANN ne izbaci krajnji rezultat obrade podataka – izlazni podatak. Izlazni podatak se zatim uspoređuje sa zadanim, prethodno definiranim izlaznim podacima, te se postupak iterira na način da se nasumično mijenjaju težinske funkcije dok izlazni podatak modela ne postane zadovoljavajući [9], [10].

Težinske vrijednosti se nasumično mijenjaju dok ANN ne prepozna uzorak, dok su pragovi ugrađene aktivacijske funkcije (eng. *activation function*) sa svrhom unošenja nelinearnosti. Aktivacijska funkcija može imati bilo kakav oblik, od kojih je jedna sigmoidalna funkcija:

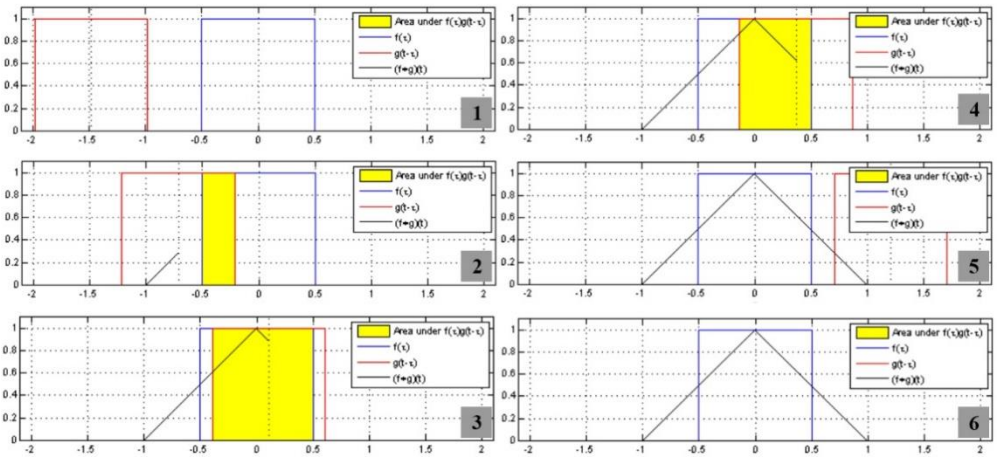
$$F(x) = \frac{1}{1 + e^{-x}}.$$

Nizanjem sličnih nelinearnih pretvorbi, stvara se mogućnost izrazito kompleksnih analiza i puno preciznijih izlaznih podataka. Ružičastim krugovima na slici 2 prikazan je aktivacijski sloj koji se inače ne ucrtava na strukturu ANN [10].

Ovisno o primjeni, postoje različite konfiguracije ANN koje se razlikuju po vrsti neurona, broju slojeva i strukturi, odnosno topologiji. Svaka od konfiguracija odgovara različitim zadacima. Trenutno, za ekstrakciju karakterističnih crta lica, najefikasnija vrsta ANN su konvolucijske neuronske mreže [11].

2.1. Konvolucija

Konvolucija, u svom najužem značenju, je matematička operacija nad dvije funkcije koja rezultira trećom funkcijom koja opisuje njihovu međuovisnost. Novo dobivena funkcija definirana je kao integral produkta dviju funkcija, s time da se od jedne funkcije uzima inverz koji se translatira po x-osi, i kao takva vrijedi za sve pomake [12].

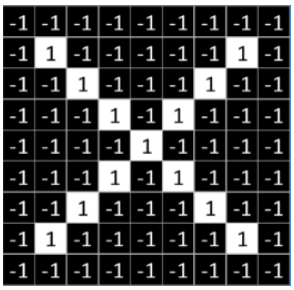


Slika 3 Proces dobivanja funkcije konvolucije [12]

Radi lakše vizualizacije konvolucije, na slici 3 dat je postupak dobivanja funkcije konvolucije. Prikazane su dvije funkcije označene plavom i crvenom bojom. S obzirom da su obje funkcije parne, inverz crvene funkcije je jednak originalnoj funkciji. Crvena funkcija je klizna (pomična), te će prelaziti preko plave funkcije. Žutom bojom označena je promjena površine, a crnom bojom označena je funkcija konvolucije koja opisuje datu površinu.

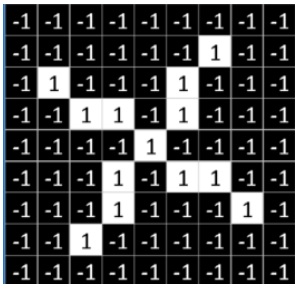
2.2. Konvolucijske neuronske mreže

Slikovne podatke računalo prepoznaje kao dvodimenzionalne mape vrijednosti kako je prikazano na slici 4.



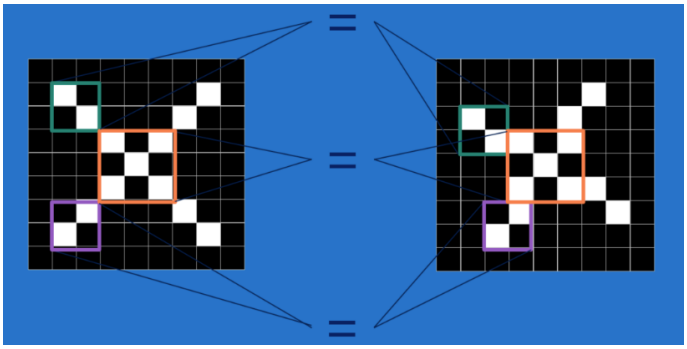
Slika 4 Znak X prikazan kao mreža vrijednosti od -1 do 1 [13]

Kod klasifikacije slika, kao što je već spomenuto, problem se javlja u trenutku kad se za jednak objekt mogu naći različite reprezentacije. Na primjer, svaka osoba ima različiti rukopis, pa bi prethodni znak X mogao izgledati kao na slici 5.



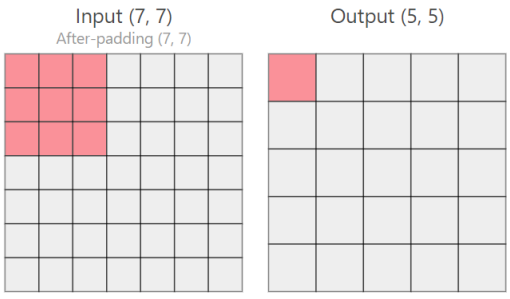
Slika 5 Drukčiji oblik znaka X kao mreža vrijednosti od -1 do 1 [13]

Konvolucijske neuronske mreže (eng. *Convolutional Neural Networks – CNN*) rade na principu prepoznavanja bitnih atributa. U slučaju znaka X, to bi bili krakovi te njihov presjek u sredini (slika 6).



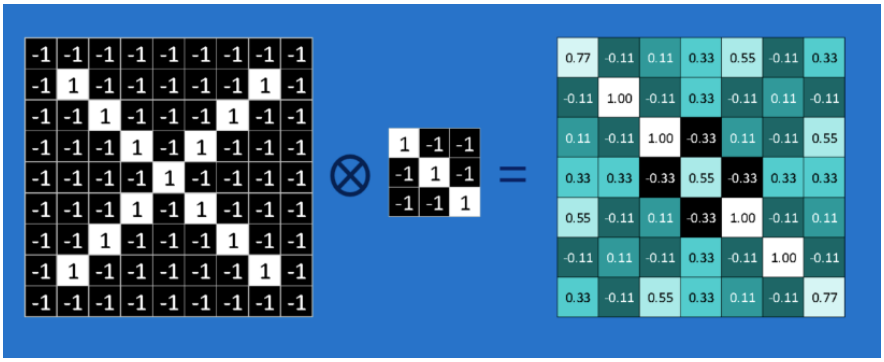
Slika 6 Bitni atributi znaka X [13]

Nakon što CNN prepozna bitne atribute objekta koje analizira, na novim slikama ih mora pronaći. Naime, izdvojenim atributom CNN prelazi preko svakog dijela nove slike te primjenjuje matematičke operacije konvolucije koja se prethodno opisala. Vrijednosti atributa i vrijednosti cijele slike nalik su matricama, s time da je cijela slika većih dimenzija. Matrica atributa klizi po svim pozicijama na slici te se korelacijske vrijednosti množe i zbrajaju. Dobivene vrijednosti će skupa tvoriti novu matricu koja je umanjena i „filtrirana“ verzija originalne slike. Na slici 7 ružičasta 3x3 matrica na lijevoj strani slike je „filter“ matrica koja redom prelazi preko 7x7 sive matrice koja predstavlja sliku. S desne strane ružičasta ćelija predstavlja rezultat konvolucije nakon što se vrijednosti na lijevoj strani pomnože i zbroje. Novo dobivena filtrirana slika biti će dimenzija 5x5. Dimenzije dobivene matrice ovise o dimenzijama originalne slike i dimenzijama atributa [13].



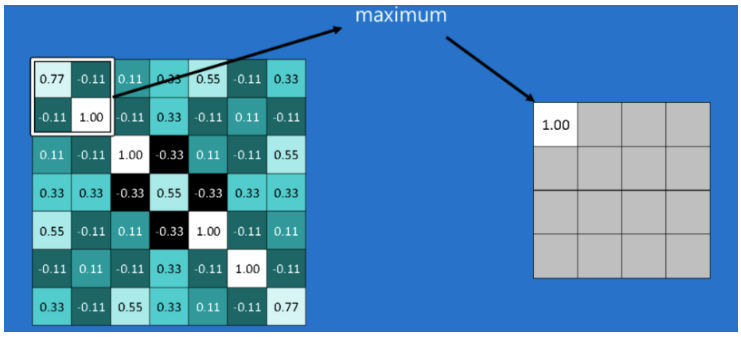
Slika 7 Veličine matrica prije i poslije konvolucije [14]

Na slici 8 su prikazane vrijednosti za prethodan primjer sa znakom X gdje je atribut jedan od krakova. „Pozitivnost“ i „negativnost“ ćelija nakon konvolucije označavaju koliko je prepoznat atribut, odnosno ako je vrijednost bliža 1 došlo je do poklapanja, a ako je bliža -1 znači da atribut na tom dijelu nije prepoznat [13].



Slika 8 Primjer filtrirane slike znaka X nakon konvolucijskih operacija [13]

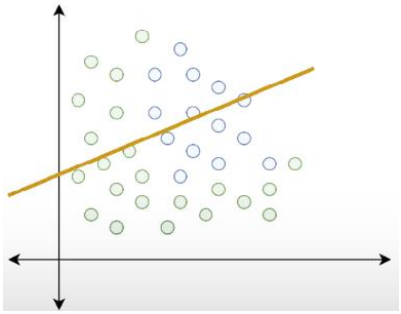
Proces konvolucije se zatim ponavlja za sve značajne atribute te sve dobivene obrađene slike skupa čine jedan konvolucijski sloj (eng. *convolution layer*). Osim kovolucijskog sloja, važno je spomenuti sloj sažimanja (eng. *pooling layer*). Naime, ovisno o kompleksnosti slike i broju atributa, unatoč tome što su u pitanju jednostavne matematičke operacije, izvedba svakog pojedinog sloja se odvija u isto vrijeme te je potrebna velika procesna snaga računala. Kako bi se proces i pojednostavnio i vremenski skratio uveden je sloj sažimanja. Funkcija sloja sažimanja je smanjiti rezoluciju filtriranih slika s time da se i dalje prepoznaju relevantni atributi. S obzirom na željenu novu rezolucije slike, na određenom broju ćelija najčešće se izvodi izračun aritmetičke sredine ili se prepisuje samo maksimalna vrijednost. Na slici 9 prikazana je izvedba funkcije sloja sažimanja s maksimalnom vrijednošću [13].



Slika 9 Maksimalna vrijednost kao funkcija sloja sažimanja [13]

Osim smanjenja procesne snage, CNN nakon sloja sažimanja „zna“ postoji li određeni atribut na slici bez obzira na njegovu poziciju na slici [13].

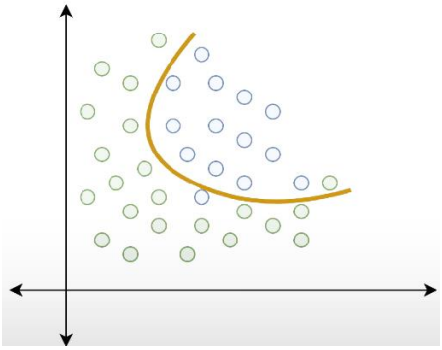
Veliki značaj CNN leži u odgovarajućoj primjeni aktivacijskih funkcija. Aktivacijske funkcije omogućavaju nelinearne granice odluke. Prije aktivacijske funkcije, CNN bi rezultirala linearnom granicom odluke kao na slici 10.



Slika 10 Linearna granica odluke za dati set podataka, Izvor:Autor

Granicom odluke se definiralo u koju vrstu krugova (zeleni/plavi) spada neki podatak $k(x, y)$. Sa slike je vidljivo da set podataka nije pravilno odvojen te su šanse za točnu klasifikaciju malene. Granica odluke je linearna zbog toga što je i CNN linearna, točnije svaki neuron je jednako aktivan. Primjerice na slici 10 s lijeve strane ima puno više zelenih krugova pa su neuroni koji su namijenjeni za prepoznavanje plavih krugova negativni i umanjuju pozitivnost neurona namijenjenih za zelene krugove. Aktivacijske funkcije smanjuju utjecaj pojedinih neurona ili ih potpuno gase. Pa ako se u istu CNN nakon konvolucije uvede prethodno

spomenuta sigmoidalna funkcija, jako negativne vrijednosti će se približavati nuli, dok će se pozitivne vrijednosti približavati jedinici. Time se smanjuje utjecaj neurona koji nisu aktivni, odnosno koji ne prepoznaju svoje atribute, pa granica odluke poprima novi oblik sa slike 11 [14], [13].



Slika 11 Nelinearna granica odluke za dati set podataka, Izvor:Autor

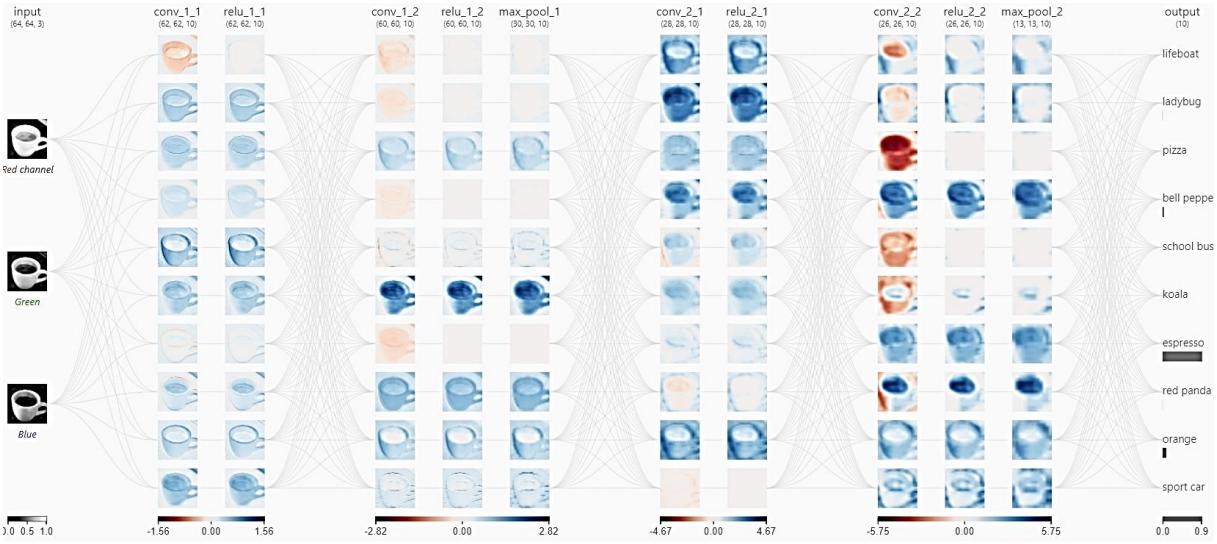
U praksi se najčešće upotrjebljuje „Rectified Linear Unit“ prijenosna funkcija koja se još skraćeno naziva ReLU funkcija:

$$f(x) = max(0, x)\ .$$

Njena svrha je preslikavanje datih vrijednosti s time da sve negativne vrijednosti postaju nula. Dosta je slična sigmoidalnoj funkciji, no zbog svoje jednostavnosti zahtjeva manju procesnu snagu. Postoje različite aktivacijske funkcije koje odgovaraju različitim zadacima CNN, stoga je bitno odabrati točnu funkciju s najboljim rezultatom procesa učenja CNN-a [14]. Na slici 12 dan je primjer izvedbe ReLU funkcije prilikom prepoznavanja naranče gdje je gornji sloj konvolucijski sloj, a donji rezultat primijenjene funkcije. Plave boje su pozitivne, dok su narančaste negativne vrijednosti.

Slika 12 Primjena ReLU funkcije [14]

Naposljetku, struktura CNN se sastoji od ulaznog i izlaznog sloja, n slojeva konvolucije, aktivacijskih funkcija i slojeva sažimanja. Rezultat izlaznog sloja i dodatne funkcije ovise o svrsi modela CNN. Na slici 13 vidljiva je cijela struktura gotovog modela CNN koja klasificira slike u definirane kategorije.



Slika 13 Analiza slike šalice kave u modelu CNN specijaliziranom za klasifikaciju [14]

Proces učenja jednog ovakvog modela započinje ljudskim kategoriziranjem velikog broja slika. Slike se analiziraju s nasumičnim vrijednostima težinskih faktora i nasumičnim atributima. Nakon svake analize, rezultat CNN jest šansa da slika spada u neku od kategorija. S obzirom da je slika već kategorizirana, CNN može izračunati veličinu greške, nakon čega izmjenjuje neke faktore i neke atribute. Ovisno o tome je li se greška nakon podešavanja smanjila, odnosno povećala, CNN nove vrijednosti zadržava, odnosno odbacuje. Na taj način se prepoznaju uzorci, dok se neki elementi specifični za pojedinu sliku zanemaruju. Taj postupak se naziva učenje s povratnim postupkom (eng. *backpropagation learning*) [13].

3. IMPLEMENTACIJA STROJNOG UČENJA U PYTHON-U

Python je besplatan programski jezik koji spada u 10 najkorištenijih programskih jezika u svijetu. Njegova pojednostavljena sintaksa i široka primjena, opravdavaju sve veću popularnost u svijetu programiranja. Podržava različite stilove programiranja poput objektnog, strukturnog i aspektno orijentiranog programiranja te se može koristiti na različitim platformama [15]. No, najveći razlog uporabe python-a za zadatke strojnog učenja je postojanje unaprijed definiranih biblioteka koje su otvorenog izvora [2] (eng. *open source*). Naime, algoritmi strojnog učenja su sami po sebi kompleksni te dostupnost gotovih alata olakšava cijeli proces programiranja jednog takvog koda. Odnosno, programer se može manje koncentrirati na programiranje, a više na sami algoritam i njegovu primjenu. U nastavku su navedene i opisane neke od ključnih biblioteka koje se mogu koristiti u python-u te povezuju strojno učenje i računalni vid za analizu ljudskog lica.

3.1. NumPy

NumPy je jedna od značajnijih python biblioteka za manipulaciju velikim bazama podataka. Naime, osnovna funkcija NumPy biblioteke je strukturiranost podataka u više dimenzionalnim nizovima i matricama te mogućnost upravljanja istim s kompleksnim matematičkim funkcijama. Primjenjuje se u širokom spektru računarstva s naglaskom na suvremene analističke aplikacije među kojima je i obrada vizualnih podataka. Trenutnom vektorizacijom velikog broja podataka zaobilaze se individualne skalarne operacije te je kod u isto vrijeme brži i jednostavniji [16].

3.2. Dlib

Dlib je biblioteka napisana u programskom jeziku C++ koja nudi mogućnost rada u python-u. Sadrži velik broj algoritama strojnog učenja s fokusom na duboko učenje i analizu ljudskog lica. Ističe se s detaljnom dokumentacijom i visokom točnošću rezultata [17].

[2] Informacije dostupne javnosti

3.3. OpenCV

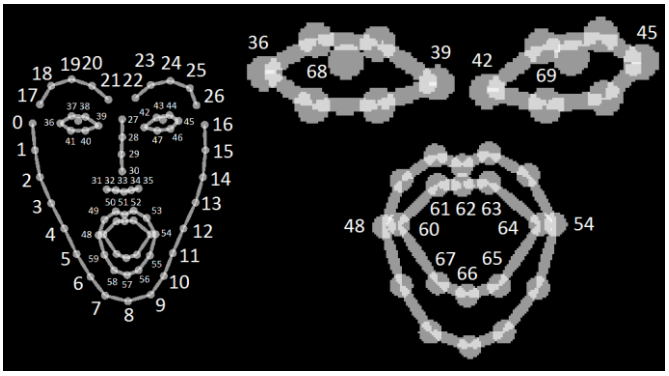
OpenCV (eng. *Open Source Computer Vision Library*) je biblioteka otvorenog izvora koja sadrži nekoliko stotina različitih algoritama računalnog vida za korištenje u stvarnom vremenu. Osim neuronskih mreža, OpenCV nudi velik broj različitih algoritamskih rješenja strojnog učenja za različite zadatke računalnog vida kao što su:

- Algoritam podizanja (eng. *boosting*),
- Učenje pomoću stabla odluke (eng. *decision tree learning*),
- Algoritam gradijentnog podizanja (eng. *Gradient boosting trees*),
- Algoritam maksimizacije očekivanja (eng. *Expectation-maximization algorithm*),
- Algoritam k-najbližeg susjeda (eng. *k-nearest neighbor algorithm*),
- Algoritam naivnog Bayes-a (eng. *Naive Bayes classifier*) i
- Algoritam nasumične šume (eng. *Random forest*) [18].

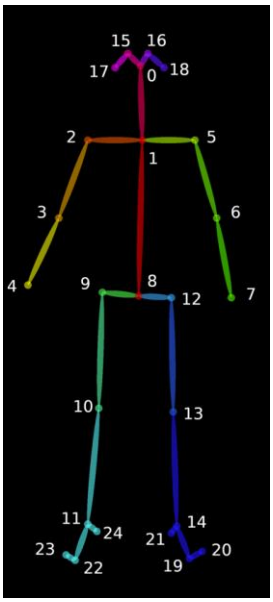
3.4. OpenPose

U radu se primijenila OpenPose[3] biblioteka otvorenog izvora specijalizirana za praćenje ljudskog tijela, lica i ruku u stvarnom vremenu koja se temelji na kombinaciji OpenCV i CNN algoritama [19]. Biblioteka se po danim uputama kreatora instalira te će se koristiti samo modeli za lice „face“ i poziciju „pose“ gdje su definirane točke lica (slika 14) te pozicija očiju u ovisnosti o nosu (slika 15).

[3] https://github.com/CMU-Perceptual-Computing-Lab/openpose



Slika 14 Oznake karakterističnih crta lica [19]



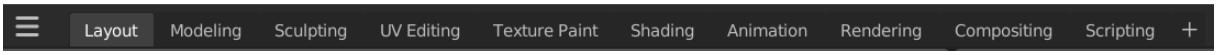
Slika 15 Oznake točaka tijela, te ovisnost očiju (15 i 16) o nosu (0)

Rezultat primjene algoritma na video je niz JSON datoteka. JSON je skraćen naziv za „*JavaScript Object Notation*“ te je standardiziran format datoteka koji služi za prijenos podataka u obliku teksta bez dodatnog programskog rješenja za njihovo očitavanje. U slučaju praćenja točaka lica, JSON sadržava njihove koordinate u jednom trenutku.

4. BLENDER PODRŠKA

Praktični dio ovog rada uključuje primjenu prethodno opisanih metoda strojnog učenja u konkretnoj animaciji 3D modela pomoću python-a i biblioteke OpenPose. Animacija će se izvoditi u „Blender“ softveru koji to omogućava.

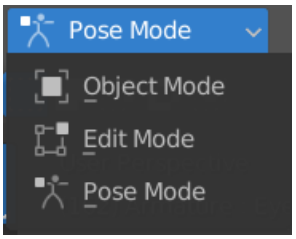
Blender je besplatan grafički softver otvorenog izvora koji nudi velik broj značajki: 3D modeliranje, UV mapiranje, teksturiranje, digitalno crtanje, animiranje, simuliranje, skulpturiranje, uređivanje videa i slične. Na gornjoj alatnoj traci prikazana su svi radni prostori u kojima je moguće raditi, te svaki ima svoje odvojene elemente (slika 16).



Slika 16 Osnovna alatna traka blendera

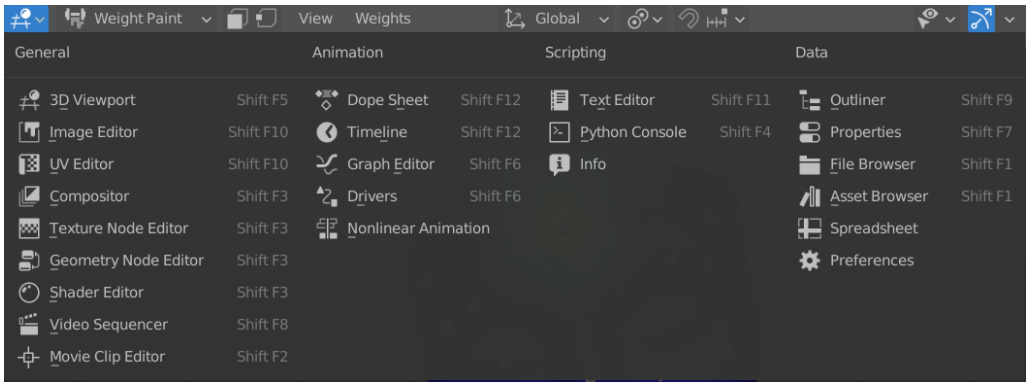
Približavanjem kursora kutevima radnog prostora pojavljuje se znak „+“ kojim je moguće raspodijeliti prostor na više radnih prostora. Time se omogućava istodobni rad s više značajki.

Ispod osnovne alatne trake dati su različiti načini rada (slika 17). Bitno ih je razlikovati jer svaki način nudi različit pristup objektima s kojima se upravlja, pa tako „Object Mode“ dozvoljava pozicioniranje, rotaciju i skaliranje objekata, „Edit Mode“ dozvoljava izmjenu geometrije, a „Pose Mode“ izmjenu poza. Svaki način rada također ima svoj set alata i opcija.



Slika 17 Načini rada u blenderu

Odmah pored načina rada nalaze se i vrste uređivača (slika 18). Uređivači nude različite prikaze istih podataka kao što je 3D pogled ili 2D pogled.



Slika 18 Vrste uređivača u blenderu

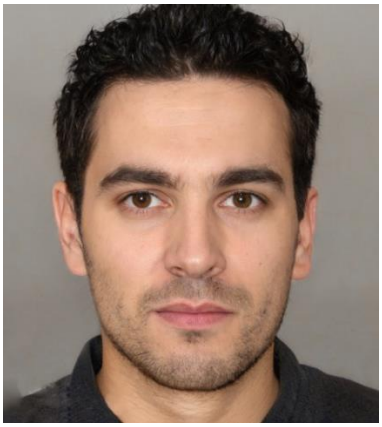
Osim toga, Blender nudi opciju dodavanja vlastitih „HotKeys“ funkcija koje služe za brže ostvarivanje repetitivnih radnji pomoću tipkovnice i/ili računalnog miša. Najčešće korišten HotKey je tipka CTRL + tipka Z, čime se poništava zadnja radnja. Osim toga, moguće je sakriti nepotrebne funkcije i radne prostore.

U ovom radu će se objasniti način dodavanja i upravljanja 3D modelima, odnosno 3D modelom lica. S obzirom na kompleksnost blendera i širok raspon značajki, primijenjeni su najbrži i najjednostavniji načini izvođenja različitih zadataka, bez obzira na njihovu nekonvencionalnost.

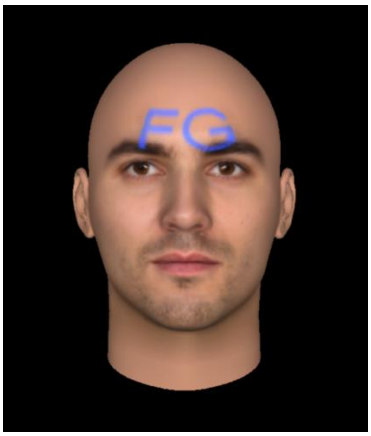
4.1. Generiranje i adaptacija lica

Blender omogućava svojim korisnicima da sami izrađuju svoje površine za oblikovanje koje će se kroz ovaj rad spominjati kao „mesh“ objekti. No, mnogi internetski izvori nude gotove blender modele sa svrhom ubrzavanja rada s drugim blender značajkama. Za izradu lica koristio se FaceGen modeller[4], aplikacija koja generira 3D model glave iz slike lica. Lice sa slike 19 je odabrano sa stranice Face Generator[5] koja koristi umjetnu inteligenciju u stvaranju lica koja ne postoje koristeći prethodno objašnjene metode strojnog učenja. Lice se po uputama označi unutar FaceGen-a koji zatim generira 3D model (slika 20). Za svrhu analize lica u blenderu dovoljan je samo dio datog modela, pa od spremljene glave se koristi samo .OBJ datoteka.

[4] https://facegen.com/modeller.htm
[5] https://generated.photos/face-generator

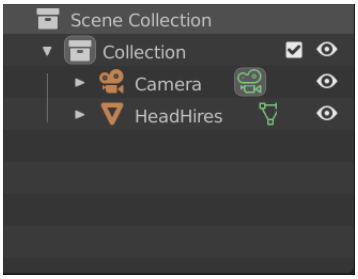


Slika 19 Odabrano lice, Izvor:Autor



Slika 20 Generiran 3D model, Izvor:Autor

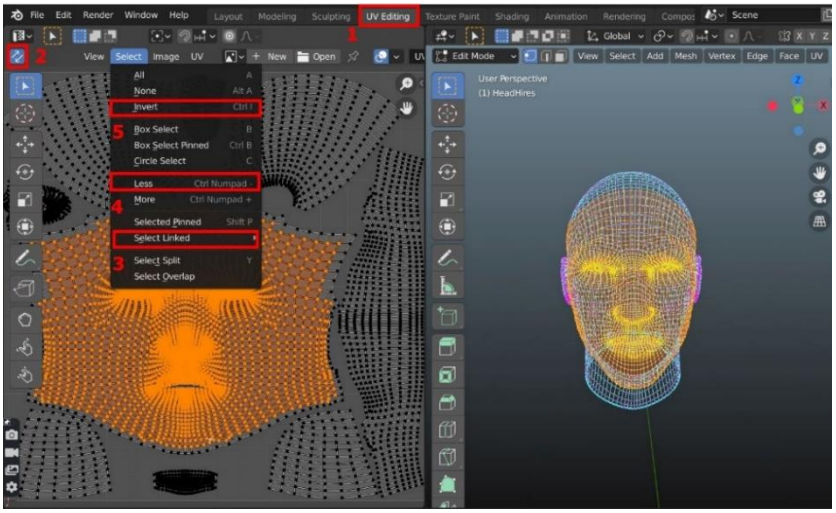
U blender se unosi spremljeni mesh glave. Za potrebe analize lica, nisu potrebni drugi objekti glave (zubi, jezik, vrat, pozadina glave) te se brišu u „Scene collection“ prozoru koji prikazuje sve objekte u radnom prostoru. Naposljetku bi popis svih objekata na sceni trebao izgledati kao na slici 21. U istom prozoru je moguće mijenjati nazive svih objekata.



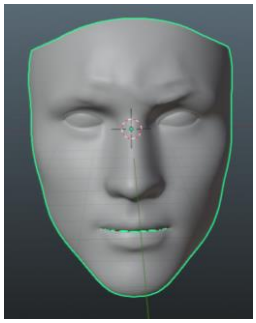
Slika 21 Scene collection nakon brisanja drugih objekata, Izvor:Autor

Zatim je potrebno doraditi osnovni mesh glave. Naime, od cijele glave je potrebno samo lice. Osim toga, dobiveni model dolazi sa spojenim usnama koje je potrebno razdvojiti kako bi se usta u animiranju mogla otvarati. Prateći sliku 22, prvo je potrebno otvoriti „UV Editing“ radni prostor. Prebacivanje mesh-a u novi radni prostor radi se preko gumba „UV Sync Selection“ u gornjem lijevom kutu, zatim označavanjem bilo koje točke središnjeg dijela, može se aktivirati „Select Linked“ funkcija koja će označiti sve povezane točke. Potrebno je naglasiti da su se automatski označile i spojne točke koje se moraju eliminirati funkcijom „Less“. Označen dio se preokreće funkcijom „Invert“ čime se odabire dio koji se naposljetku briše u desnom

„Layout“ prozoru desnim klikom i funckijom „Delete vertices“. Na posljetku bi mesh lica trebao izgledati kao na slici 23.



Slika 22 Koraci u doradi lica, Izvor:Autor



Slika 23 Konačni izgled mesh-a lica, Izvor:Autor

4.2. Postavljanje armature

Nakon što je mesh potpuno dorađen, potrebno je izraditi i postaviti armaturu. Armatura je vrsta objekta u Blender-u koja služi za upravljanje i namještanje mesh objekata (eng. *rigging*). U ovom radu koristi se vrsta armatura „Single Bone“, odnosno kost. Svaka kost će se definirati na način da prati koordinate jedne karakteristične crte lica i upravlja jednim dijelom mesh-a. Potrebno je postaviti armaturu u 3D okruženje te povezati ukupnu armaturu s mesh-om lica.

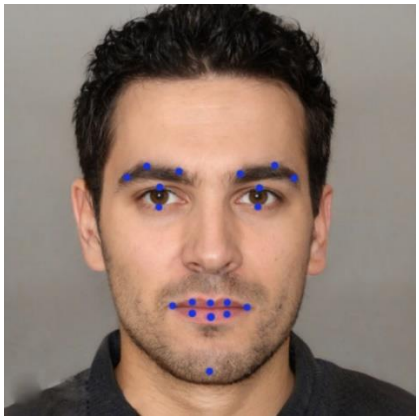
Praktično rješenje za praćenje točaka se naširoko koristi u filmskoj industriji. Naime, na praćen objekt se u stvarnosti postave oznake te se kasnije računalno obrađuju putanje tih oznaka. Te putanje se kasnije mogu implementirati na drugi objekt i time ga animirati. Opisana metoda je jedna od mnogih u računalnom generiranju slika, većini poznat kao CGI (eng. *Computer-generated imagery*). Primjer praćenja i animiranja lica u filmskoj industriji je vidljiv na slici 24.

Slika 24 Praćenje lica u filmu [20]

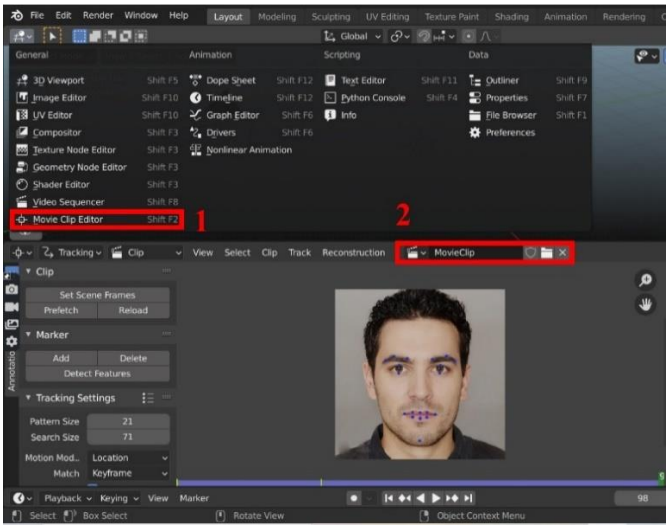
Blender nudi značajku „Tracking“ koja omogućuje opisan proces te će se za potrebe ovog rada koristiti kao način pozicioniranja karakterističnih točaka lica koje se žele pratiti. Tim postupkom se izbjegava manualno postavljanje točaka na lice, što je zbog mnogo izbočina na licu iscrpan proces.

U nastavku su opisani koraci prostornog postavljanja armature s pripadajućom slikom:

1. Na slici lica je potrebno u odvojenoj aplikaciji označiti buduće pozicije kostiju koje će se povezati s željenim točkama. Točke su uzete sa slike 14 te je za neke točke odabrana srednja vrijednost radi uniformnije animacije mesh-a. S obzirom da je aplikacija namijenjena za video podatke, sliku je potrebno kopirati nekoliko puta te sve kopije otvoriti u „Movie Clip Editor“ uređivaču.

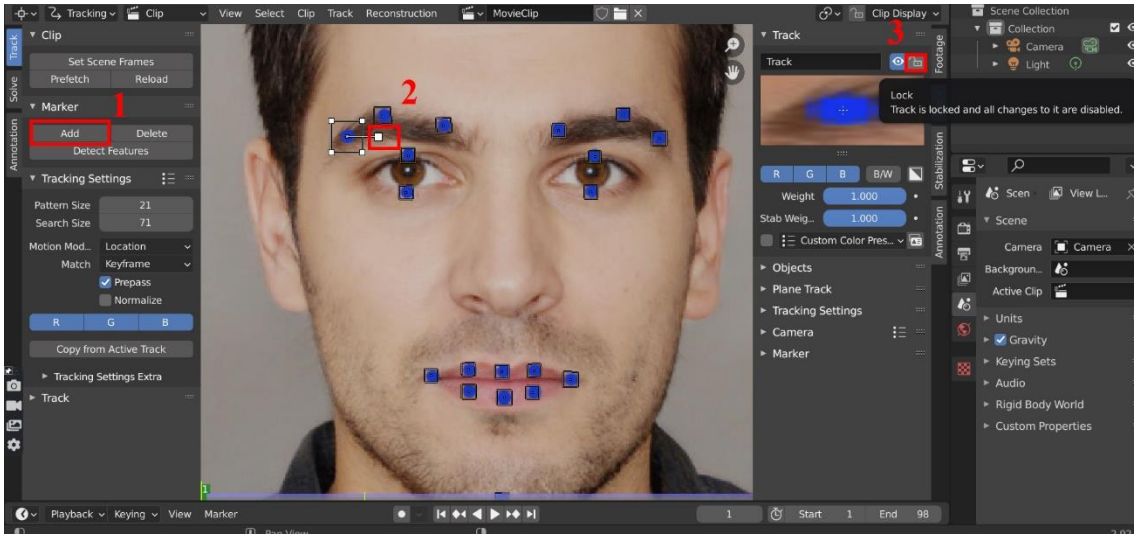


Slika 25 Označene točke lica, Izvor: Autor



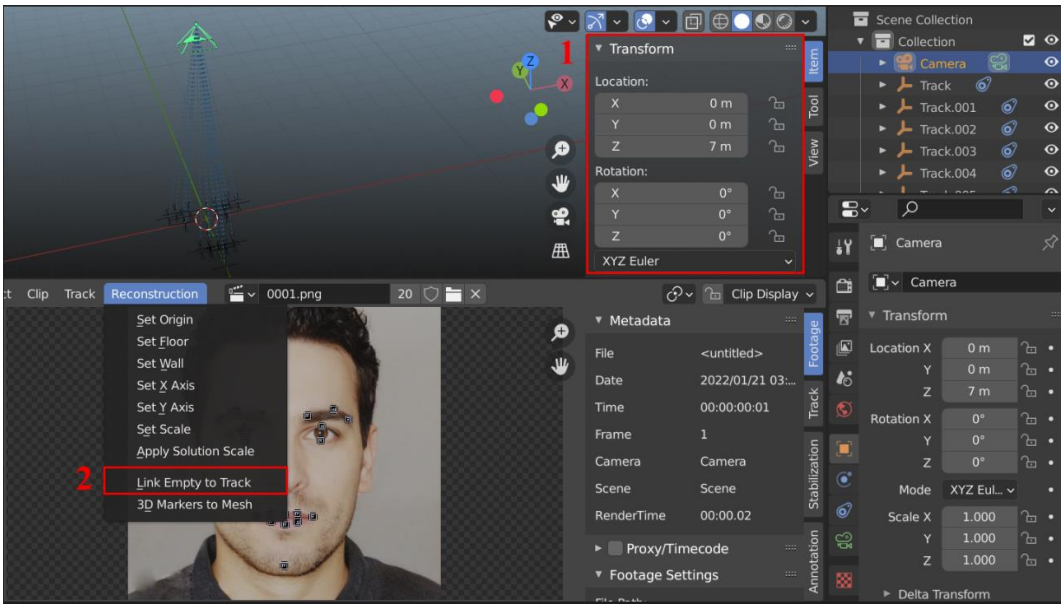
Slika 26 Koraci za pripremu videa, Izvor:Autor

2. Pomoću gumba „Add“ i klikom na središte točke, dodaje se oznaka za svaku točku na licu. Veličina oznake se prilagodi veličini točke s izduženim krakom, te se po potrebi popravi pozicija. Naposljetku se pozicija oznake zaključa.



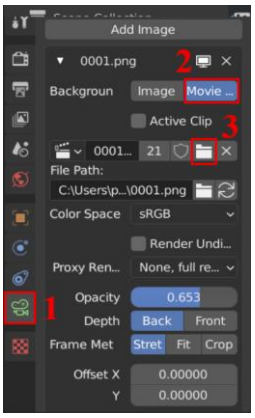
Slika 27 Dodavanje oznaka svim točkama lica , Izvor:Autor

3. U usporedno otvorenom „3D Viewport-u“ je potrebno namjestiti kameru tako da bude u središtu koordinatnog sustava na nekoj proizvoljnoj visini sa svim rotacijama jednakim nuli. Zatim se sve oznake skupa označe te se omogući „Link Empty To Track“ čime se unose sve oznake kao projekcija kamere.

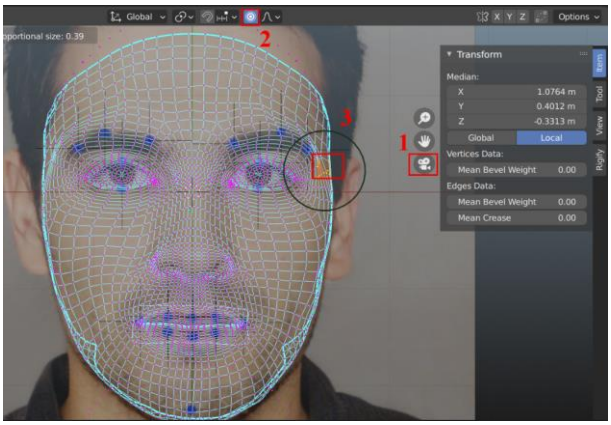


Slika 28 Projiciranje točaka, Izvor:Autor

4. S obzirom da FaceGen modeller nije pretjerano precizan, potrebno je prilagoditi mesh lica za što će poslužiti dodavanje pozadinske slike kao na slici 29. Potrebno je prebaciti način rada u „Edit Mode“ te prikazati mesh lica kao mrežu (tipka Z + „Wireframe“) kako bi se vidjela pozadinska slika. Mesh se zatim prilagođava na način da se omogući proporcionalno uređivanje (eng. *proportional editing*) te se obilježi dio koji je potrebno premještati (slika 30). Tipkom G („Grab“) se pojavi kružnica koja reprezentira veličinu dijela mesh-a koji će se pomicati zajedno s označenim točkama. Sama kružnica se korigira kotačićem na računalnom mišu. Nakon završetka ovog koraka potrebno je vratiti se u „Object Mode“.

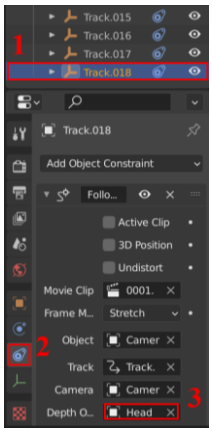


Slika 29 Dodavanje pozadinske slike, Izvor:Autor



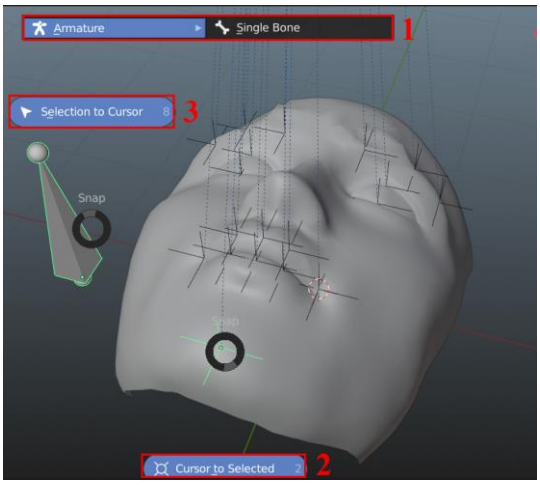
Slika 30 Uređivanje mesh-a lica, Izvor:Autor

5. Svaku oznaku je zatim potrebno označiti te dodati ograničenje dubine. Kao podloga dubine odabrat će se mesh lica kako bi se oznake pravilno rasporedile u prostoru.



Slika 31 Dodavanje dubine oznakama, Izvor:Autor

6. Nakon što se kost doda (tipka SHIFT + tipka A), potrebno ju je spojiti s oznakama. Prvo se označi jedna točka i odabire se „Cursor_to Selected“ (tipka SHIFT + tipka S), nakon čega se označi kost i odabire „Selected_to Cursor“. Proces sa slike 32 se ponovi za svaku kost i točku.

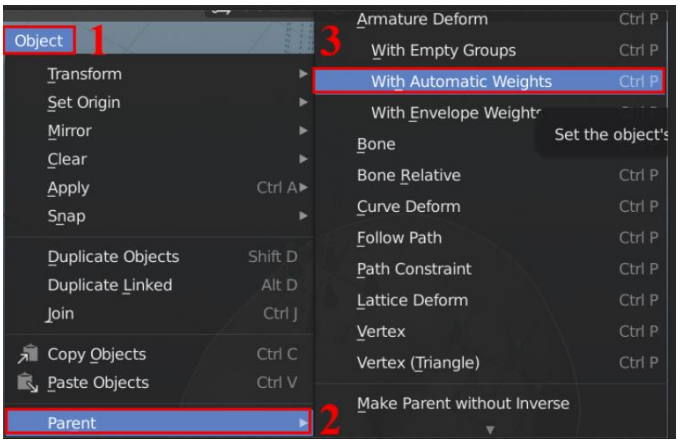


Slika 32 Dodavanje kostiju oznakama, Izvor:Autor

Nakon što su sve kosti smještene na jedno mjesto, potrebno je spojiti sve kosti u jednu armaturu što se postiže označavanjem svih kostiju te biranjem opcije „Join“ u „Object“ izborniku.

4.3. Povezivanje armature s licem

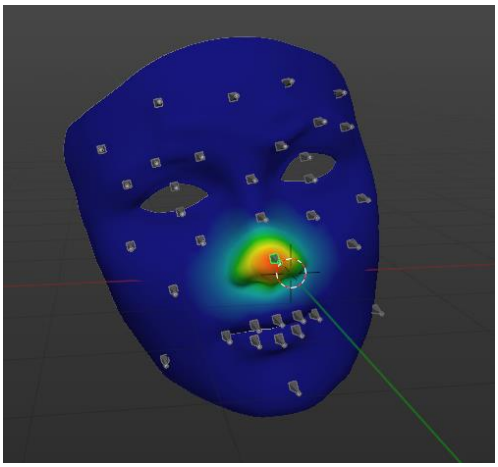
Iako je armatura postavljena na točne pozicije, ona nije povezana sa samim mesh-om, odnosno pomicanjem armature se ne postiže upravljanje mesh-om. U Blenderu postoji više opcija za povezivanje armature s licem. Za početak će se koristiti „Parent“ opcija u „Object“ izborniku za postavljanje automatskih utega „With Automatic Weights“. Prvo se označava lice, a zatim se držeći tipku SHIFT označi i armatura.



Slika 33 Dodavanje automatskih utega, Izvor:Autor

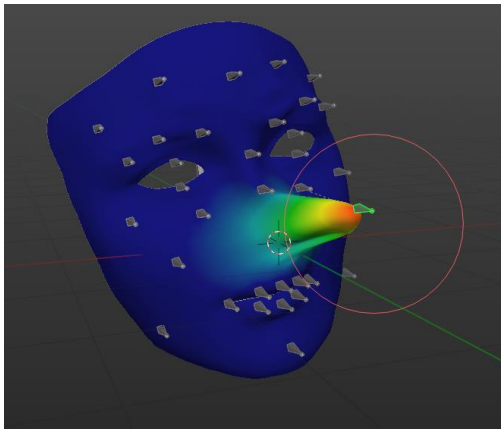
Time Blender automatski detektira i dodaje određeni dio mesh-a svakoj kosti, no taj postupak rezultira određenim greškama koje je potrebno prepraviti.

Za početak je potrebno označiti armaturu i mesh lica odabrati način rada „Weight Paint“. U tom načinu moguće je kistom dodavati jakost utjecaja neke kosti na neki dio mesh-a te se taj utjecaj može zamišljati kao uteg. Odnosno, svakoj kosti se dodjeljuje točno određeni dio lica na koji ona utječe, s time da najviše utječe na dio najbliže kosti. Na slici 34 je prikazana raspodjela utega za kost na vrhu nosa. Vidljivo je kako boje gradiraju od crvene prema plavoj, što označava jakost utjecaja od najjačeg do najslabijeg.



Slika 34 Utjecaj kosti na vrhu nosa na mesh lica, Izvor: Autor

Izmjenjivanje između kostiju se postiže držanjem tipke CTRL i lijevim klikom na kost. Pritiskom tipke G i pomicanjem kursora može se vidjeti na koji način pomicanje kosti utječe na mesh (slika 35).



Slika 35 Prikaz utjecaja pomaka kostiju na mesh, Izvor: Autor

Time se osigurava da kost nije zahvatila krivi dio mesh-a, jer i najmanjim odstupanjem može doći do preračunavanja više različitih utega i time značajnih deformacija. Vraćanje kosti na svoje originalno mjesto se postiže tipkom ESC.

Desnim klikom miša bilo gdje unutar radnog prostora otvora se prozor za parametre kista kojim se postavljaju utezi. U njemu je moguće korigirati jakost utega, veličinu kista i snagu kista. Postavi li se jakost utega na nulu, moguće je brisati utege, dok korigiranjem snage kista, moguće je postići bolje prijelaze utega. To je bitno s obzirom da svaki mišić na licu lagano utječe na

druge mišiće. Primjerice, podizanjem obrva se pomiče i čelo. Zbog kompleksnosti ljudskog lica i među utjecaja mišića, u koraku postavljanja armature dodane su dodatne kosti radi točnije prvotne raspodjele utega.

Nakon što se za svaku kost definira njena zona utjecaja, moguće je animirati mesh.

5. POVEZIVANJE KARAKTERISTIČNIH TOČAKA S ARMATUROM

U nastavku će se opisati struktura koda koji očitava JSON datoteke te primjenjuje vrijednosti koordinata svake točke na korelirajuće kosti. Kod se upisuje u radnom prostoru „Scripting“ koji u blenderu služi za kodno upravljanje objektima i njihovim značajkama.

Početni dio koda sadrži sve biblioteke koje su potrebne za ostvarivanje python programiranja u blenderu, prijenos JSON datoteka i dodavanja matematičkih funkcija. Na slici 36 su prikazane odgovarajuće biblioteke te funkcija za očitavanje JSON-a. Potrebno je uskladiti ime videa i armature te mapu u kojoj se nalazi.

```
1  import bpy
2  import json
3  import os
4  import os.path
5  from os import path
6  import math
7  import mathutils
8
9  FileIdentifier = "Movie01"
10 DestArmName = "Armature"
11
12
13 JSON_FOLDER = 'C:/Users ...'
14
15 totalFiles = 0
16
17 for base, dirs, files in os.walk(JSON_FOLDER):
18     for Files in files:
19         totalFiles += 1
20
```

Slika 36 Biblioteke i uvođenje JSON datoteka, Izvor: Autor

Nakon toga se definiraju matematičke funkcije za dohvaćanje i preoblikovanje koordinata te dobivanje kutova prilikom pomicanja cijele glave. Jedna od funkcija prikazana je na slici 37.

```
52 def GetPoint (Array, index):
53     baseIndex = index*3
54     x = float(Array[baseIndex])
55     y = float(Array[baseIndex + 1])
56     reliability = float(Array[baseIndex + 2])
57     return [x,y,reliability]
```

Slika 37 Funkcija za dohvaćanje koordinata iz JSON datoteka, Izvor: Autor

Zatim je potrebno definirati klase unutar koda. Naime, python je objektno orijentiran programski jezik, odnosno kod se temelji na upravljanju objekata i njegovih atributa unutar

neke klase. Na slici 38 je definirana klasa PersonJSONData čija je uloga definirati osnovne objekte lica kao što su početna pozicija i trenutna pozicija te načini njihovog upravljanja.

```
 93  class PersonJSONData:
 94    def __init__(self, pose, face, EarLobeToEarLobedistanceInBlenderUnits, TieEyelidsTogether):
 95      self.poseStart = pose
 96      self.faceStart = face
 97      self.poseCurrent = pose
 98      self.faceCurrent = face
 99      self.TieEyelidsTogether = TieEyelidsTogether
100      self.distancBetweenEars = abs(GetPoint(pose, 18)[0] - GetPoint(pose, 18)[1])
101      self.StartNosePosition = GetPoint(pose, 0)
102      self.StartFaceNosePosition = GetPoint(face, 30)
103      self.StartRightEyePosition = GetPoint(pose, 15)
104      self.StartLeftEyePosition = GetPoint(pose, 16)
105      self.LastGoodHeadTilt = 0
106      self.StartHeadTilt = self.getHeadTiltSideToSide()
107      self.EarLobeToEarLobedistanceInBlenderUnits = EarLobeToEarLobedistanceInBlenderUnits
```

Slika 38 Klasa PersonJSONData sa svojim atributima, Izvor: Autor

Svaki objekt u klasi ima svoje atribute, primjerice koordinate. Stoga se funkcijom __init__ dodjeljuju vrijednosti tim atributima, među kojima su „pose“ i „face“ definirani objekti sa svojim izračunatim atributima unutar OpenPose biblioteke. Odnosno, definirani objekti unutar klase PersonJSONData nasljeđuju atribute iz OpenPose biblioteke te se dodatno definiraju vrijednosti atributa ostalih objekata koji se ne mogu naslijediti jer nisu unutar OpenPose-a.

Osim definiranja objekata i njihovih atributa, potrebno je definirati funkcije kojima se može upravljati datim objektima. Primjerice na slici 39, definirana je funkcija za postavljanje trenutne pozicije lica.

```
104    def SetCurrentPose(self, pose, face):
105      self.poseCurrent = pose
106      self.faceCurrent = face
```

Slika 39 Funkcija vezana uz objekte poseCurrent i faceCurrent koja postavlja trenutnu poziciju lica, Izvor: Autor

Daljnjim definiranjem svih potrebnih funkcija i matematičkih operacija, mogu se definirati funkcije koje dohvaćaju koordinate svih željenih točaka. Kao što je već spomenuto, u ovom radu se nisu uzimale sve točke lica, već srednja vrijednost između dviju točaka kako bi

pomicanje izgledalo prirodnije. Primjer jedne takve funkcije za dohvaćanje pozicije srednje točke donje usne je prikazan na slici 40.

```
183    def getLowerLipCenterPosition(self, correction):
184      output = self.ConvertPixelPointToBlenderUnits(self.getFacePosition(57,66, "nose"))
185      return multiply(output, correction)
```

Slika 40 Funkcija za dohvaćanje pozicije srednje točke donje usne, Izvor: Autor

Na sličan način se definiraju sve ostale funkcije za dohvaćanje koordinata za svaku kost koju se želi animirati.

Zatim je potrebno povezati kosti sa samom lokacijom točaka kao na slici 41. Odnosno kost kojoj je dodijeljeno ime „LipLowerMiddle“ poprima vrijednost lokacije „lower_lip_center“.

```
305  lower_lip_center =  DestArm.pose.bones["LipLowerMiddle"]
```

Slika 41 Povezivanje lokacije karakteristične točke s odgovarajućom kosti, Izvor: Autor

Nakon što se sve lokacije povežu s kostima, za svaku JSON datoteku se definiraju vrijednosti tih lokacija, odnosno koordinate. Na slici 42 prikazano je kako se iz objekta p1 dohvaćaju x i z koordinate pomoću funkcije getLowerLipCenterPosition lokacije lower_lip_center koja je povezana s kosti „LipLowerMiddle“. S „Offset“ vrijednošću se može korigirati koliko točno se pozicija translatira s obzirom da kost nije savršeno pozicionirana, odnosno 1 bi značilo da se koordinate točno prate, manje vrijednosti umanjuju, dok veće uvećavaju vrijednost koordinata.

```
374          Offset = p1.getLowerLipCenterPosition(0.5)
375          lower_lip_center.location.x = Offset[0]
376          lower_lip_center.location.z = Offset[1]
```

Slika 42 Dodjeljivanje vrijednosti lokacijama kostiju, Izvor: Autor

Objekt p1 je prethodno definiran kao na slici 43.

```
357         if FirstTime:
358             p1 = PersonJSONData(pose, face, EarLobeToEarLobedistanceInBlenderUnits, TieEyelidsTogether)
359             FirstTime = False
360         else:
361             p1.SetCurrentPose(pose, face)
```

Slika 43 Objekt p1, Izvor: Autor

Dosad je obrađena sljedeća struktura koda:

Slika 44 Dio koda za manipulaciju podataka unutar JSON datoteka, Izvor: Autor

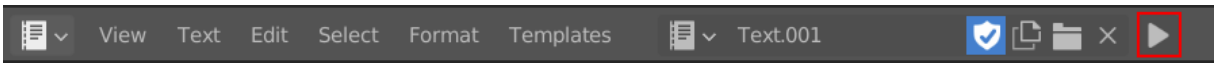
Taj dio koda je dio za manipulaciju JSON datotekama i podacima unutar njih te nakon što se dobro definira, potrebno je animirati promjenu lokacija.

Svaki video se sastoji od *n* izreza slika, odnosno od *n* „frame-ova“ gdje svaki frame sadržava sve trenutne lokacije kostiju. Dodjeljivanje lokacija frame-ovima se postiže kao na slici 45.

```
473         if (CurrentFrame % mouth_KeyFrame_Every_Nth_Frame == 0) and (keyFrame):
474             lower_lip_center.keyframe_insert(data_path='location',frame= StartFrameNumber + CurrentFrame)
```

Slika 45 Dodjeljivanje lokacija frame-ovima, Izvor: Autor

Animiranje se postiže izmjenom frame-ova, odnosno promjenom lokacija. Nakon što je kod potpuno gotov, njegova aktualizacija se postiže pokretanjem u alatnoj traci radnog prostora „Scripting“ (slika 46).



Slika 46 Aktualizacija koda, Izvor: Autor

Animaciju je moguće pokrenuti, zaustaviti, ubrzati i usporiti unutar donje alatne trake blendera za upravljanje animacijom i brojem frame-ova (slika 47). Desno na alatnoj traci se prikazuje broj frame-ova unutar animacije, što je potrebno uskladiti s brojem JSON-a ili postaviti na što veći broj.

Slika 47 Alatna traka za upravljanje animacijom, Izvor: Autor

Cijeli kod se može vidjeti unutar poglavlja PRILOG.

6. ZAKLJUČAK

Umjetna inteligencija je postala moćan alat za velik broj različitih zadataka, s naglaskom na računalni vid i analizu vizualnih podataka. Konvolucijske neuronske mreže spadaju u moderne metode strojnog učenja za prepoznavanje obrazaca unutar slikovnih podataka te su vrlo precizne i efikasne za probleme praćenja karakterističnih crta ljudskog lica.

U strojnom učenju najčešće se primjenjuje programiranje u python programskom jeziku zbog njegove jednostavne i fleksibilne strukture, te radi mnogih biblioteka otvorenog izvora. Stoga, rad u python-u olakšava korištenje kompleksnih algoritama strojnog učenja.

U ovom radu se primijenila kombinacija python-a i strojnog učenja radi ostvarivanja efikasnog mapiranja karakterističnih crta lica. Zatim se dobivena mapa podataka koristila za animiranje 3D modela u Blender-u. Blender je svestran i besplatan softver koji pretežito služi za 3D modeliranje i njihovu animaciju. S obzirom na veliki broj značajki koje blender nudi, snalaženje i rad u njemu je relativno težak zadatak stoga su se u ovom radu birale nekonvencionalne metode obavljanja nekih zadataka.

Naposljetku, krajnji python kod ima dvostruku ulogu – dohvaćanje i strukturiranje mape podataka s koordinatama karakterističnih crta lica te animiranje istih unutar blendera. Kod, kao i mesh model, je univerzalno primjenjiv na svaki video zapis ljudskog lica.

LITERATURA

[1] »Wikipedija: Umjetna inteligencija,« [Mrežno]. Available: https://hr.wikipedia.org/wiki/Umjetna_inteligencija. [Pokušaj pristupa 08 01 2022].

[2] S. Dick, »Artificial intelligence,« HDSR, 2 7 2019. [Mrežno]. Available: https://hdsr.mitpress.mit.edu/pub/0aytgrau/release/2. [Pokušaj pristupa 08 01 2022].

[3] N. O' Mahony, S. Campbell, A. Carvalho, S. Harapanahalli, G. Velasco Hernandez, L. Krpalkova, D. Riordan i J. Walsh, »Deep learning vs. Traditional Computer Vision,« 39 10 2019. [Mrežno]. Available: https://arxiv.org/abs/1910.13796. [Pokušaj pristupa 16 01 2022].

[4] A. Vasuki i S. Govindaraju, »Deep Neural Networks for Image,« u *Deep Learning for Image Processing Applications*, Amsterdam, IOS Press, 2017, pp. 27-50.

[5] S. J. Russell i P. Norvig, Artificial Intelligence: A Modern Approach, New Jersey: Prentice-Hall, Inc , 1995.

[6] S. Shalev-Shwartz i S. Ben-David, Understanding machine learning, New York: Cambridge University Press, 2014.

[7] »Training, validation, and test sets,« Wikipedia, [Mrežno]. Available: https://en.wikipedia.org/wiki/Training,_validation,_and_test_sets. [Pokušaj pristupa 16 01 2022].

[8] Q. Liu i Y. Wu, »ResearchGate: Supervised Learning,« 01 2012. [Mrežno]. Available: https://www.researchgate.net/publication/229031588_Supervised_Learning. [Pokušaj pristupa 16 01 2022].

[9] B. Dalbelo Bašić , M. Čupić i J. Šnajder, »Umjetne neuronske mreže,« Svibanj 2008. [Mrežno]. Available: https://www.fer.unizg.hr/_download/repository/UmjetneNeuronskeMreze.pdf.

[10] »Machine learning,« Gooogle Developers, [Mrežno]. Available: https://developers.google.com/machine-learning/crash-course/introduction-to-neural-networks/anatomy.

[11] A. Tch, »The mostly complete chart of Neural Networks, explained,« Towards data science, 4 Kolovoz 2017. [Mrežno]. Available: https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464.

[12] »Convolution,« Wikipedia, Srpanj 2022. [Mrežno]. Available: https://en.wikipedia.org/wiki/Convolution#Visual_explanation.

[13] B. Rohrer, »How do Convolutional Neural Networks work?,« [Mrežno]. Available: https://e2eml.school/how_convolutional_neural_networks_work.html.

[14] J. Wang, R. Turko, O. Shaikh, H. Park, N. Das, F. Hohman, M. Kahng i P. Chau, »CNN Explainer,« Georgia Tech, Oregon State, [Mrežno]. Available: https://poloclub.github.io/cnn-explainer/.

[15] »Python,« Wikipedia, [Mrežno]. Available: https://en.wikipedia.org/wiki/Python_(programming_language).

[16] »NumPy,« [Mrežno]. Available: https://numpy.org/arraycomputing/.

[17] »Dlib,« [Mrežno]. Available: http://dlib.net.

[18] »OpenCv,« [Mrežno]. Available: https://opencv.org.

[19] »GitHub Openpose,« [Mrežno]. Available: https://github.com/CMU-Perceptual-Computing-Lab/openpose.

[20] »A Brief Visual History of Motion-Capture Performance on Film,« Screencrush, [Mrežno]. Available: https://screencrush.com/motion-capture-movies/. [Pokušaj pristupa 16 01 2022].

[21] J. Patterson i A. Gibson, Deep Learning: A Practitioner's Approach, Sebastopol: O'Reilly Media, 2017.

[22] K. Huang, A. Hussain, Q.-F. Wang i R. Zhang, Deep Learning: Fundamentals, Theory and Applications, Cham: Springer, 2019.

[23] N. D. Lewis, Deep learning made easy with R, AusCov, 2016.

[24] L. Deng i D. Yu, Deep learning: Methods and Applications, Redmond: now, 2014.

PRILOG

1. KOD

U nastavku je dan python kod za povezivanje karakterističnih crta lica pomoću OpenPose biblioteke s 3D modelom. Kod je namijenjen za primjenu u Blender softveru.

```
import bpy
import json
import os
import os.path
from os import path
import math
import mathutils

FileIdentifier = "Ime videa"
DestArmName = "Ime armature"

EarLobeToEarLobedistanceInBlenderUnits = 2.56


StartFrameNumber = 0
mouth_KeyFrame_Every_Nth_Frame = 4
eyes_KeyFrame_Every_Nth_Frame = 15


JSON_FOLDER = 'C:Folder s JSON datotekama'

totalFiles = 0

for base, dirs, files in os.walk(JSON_FOLDER):
    for Files in files:
        totalFiles += 1


print('Total number of files',totalFiles)

NumberOfFramesToTransfer = totalFiles
keyFrame = True


TieEyelidsTogether = True
EyelidBlinkTHreshold = .01


def combineRReliability(pt1, pt2):
    rel1 = pt1[2]
    rel2 = pt2[2]
    if rel1 < rel2:
        relOut = rel1
    else:
        relOut = rel2
    return relOut

def GetPoint (Array, index):
    baseIndex = index*3
    x = float(Array[baseIndex])
    y = float(Array[baseIndex + 1])
    reliability = float(Array[baseIndex + 2])
    return [x,y,reliability]

def rotatePoint(point, center, angle):
    angle = math.radians(angle )
    rotatedX = math.cos(angle) * (point[0] - center[0]) - math.sin(angle) * (point[1]-center[1])
+ center[0];
```

```
    rotatedY = math.sin(angle) * (point[0] - center[0]) + math.cos(angle) * (point[1] -
center[1]) + center[1];
    return [rotatedX,rotatedY,point[2]]

def AverageTwoPoints(pt1, pt2):
    avgX = (pt1[0] + pt2[0])/2
    avgY = (pt1[1] + pt2[1])/2
    return [avgX, avgY,combineRReliability(pt1,pt2)]

def DifferenceBetweenPoint(pt1, pt2):
    diffX = (pt1[0] - pt2[0])
    diffY = (pt1[1] - pt2[1])
    return [diffX, diffY,combineRReliability(pt1,pt2)]

def InverseXandY(pt):
    x = pt[0]*-1
    y = pt[1]*-1
    return [x, y, pt[2]]

def InverseX(pt):
    x = pt[0]*-1
    y = pt[1]
    return [x, y, pt[2]]

def multiply(pt, val):
    x = pt[0]*val
    y = pt[1]*val
    return [x, y, pt[2]]


class PersonJSONData:
  def __init__(self, pose, face, EarLobeToEarLobedistanceInBlenderUnits, TieEyelidsTogether):
    self.poseStart = pose
    self.faceStart = face
    self.poseCurrent = pose
    self.faceCurrent = face
    self.TieEyelidsTogether = TieEyelidsTogether
    self.distancBetweenEars = abs(GetPoint(pose, 18)[0] - GetPoint(pose, 18)[1])
    self.StartNosePosition = GetPoint(pose, 0)
    self.StartFaceNosePosition = GetPoint(face, 30)
    self.StartRightEyePosition = GetPoint(pose, 15)
    self.StartLeftEyePosition = GetPoint(pose, 16)
    self.LastGoodHeadTilt = 0
    self.StartHeadTilt = self.getHeadTiltSideToSide()
    self.EarLobeToEarLobedistanceInBlenderUnits = EarLobeToEarLobedistanceInBlenderUnits

  def SetCurrentPose(self, pose, face):
    self.poseCurrent = pose
    self.faceCurrent = face

  def GetCurrentPointHeadTiltCorrected(self, Point):
      angle = self.getHeadTiltSideToSide()
      PointCorrrected = rotatePoint(GetPoint(self.faceCurrent, 8), Point,(angle*-1))
      return PointCorrrected

  def GetStartPointHeadTiltCorrected(self, Point):
      angle = self.StartHeadTilt
      PointCorrrected = rotatePoint(GetPoint(self.faceStart, 8), Point,(angle*-1))
      return PointCorrrected

  def GetDistanceBetweenTwoCurrentPoints(self, Pnt1, Pnt2):
      x2 = Pnt2[0]
      x1 = Pnt1[0]
      y2 = Pnt2[1]
      y1 = Pnt1[1]
      dist = math.sqrt((x2 - x1)**2 + (y2 - y1)**2)
      return dist

  def ConvertPixelPointToBlenderUnits(self, pnt):
      pntx = pnt[0] * (self.EarLobeToEarLobedistanceInBlenderUnits/self.distancBetweenEars)
      pnty = pnt[1] * (self.EarLobeToEarLobedistanceInBlenderUnits/self.distancBetweenEars)
```

```
      reliability = pnt[2]
      return [pntx,pnty,reliability]

  def getHeadTiltSideToSide(self):
    leftEar = GetPoint(self.poseCurrent, 18)
    rightEar = GetPoint(self.poseCurrent, 17)
    deltaY = leftEar[1] - rightEar[1]
    deltaX = leftEar[0] - rightEar[0]
    if leftEar[2] < .1 or rightEar[2] < .1:
        tiltOut = self.LastGoodHeadTilt
    else:
        tilt = math.atan(deltaY/deltaX)
        tiltOut = math.degrees(tilt*-1)
        self.LastGoodHeadTilt = tiltOut
    return tiltOut

  def getHeadTiltUpDown(self):
    CurrentNosePosition = GetPoint(self.poseCurrent, 0)
    deltaY = CurrentNosePosition[1] - self.StartNosePosition[1]
    tilt = math.atan(deltaY/self.distancBetweenEars)
    return math.degrees(tilt)

  def getHeadTiltLeftRight(self):
    CurrentNosePosition = GetPoint(self.poseCurrent, 0)
    deltaX = CurrentNosePosition[0] - self.StartNosePosition[0]
    tilt = math.atan(deltaX/self.distancBetweenEars)
    return math.degrees(tilt)

  def getJawTiltUpDown(self):
    NoseToJawStartDist = self.GetDistanceBetweenTwoCurrentPoints(GetPoint(self.faceStart,
8),GetPoint(self.faceStart, 30))
    NoseToJawCurrentDist = self.GetDistanceBetweenTwoCurrentPoints(GetPoint(self.faceCurrent,
8),GetPoint(self.faceCurrent, 30))
    JawDelta = NoseToJawStartDist - NoseToJawCurrentDist
    PivottoEndOfJaw = self.distancBetweenEars - .125*self.distancBetweenEars
    Jawtilt = math.atan(JawDelta/PivottoEndOfJaw)
    return math.degrees(Jawtilt)

  def getFacePosition(self, ptnum1, ptnum2, RelativeTo):
    ptAvgStart = AverageTwoPoints(GetPoint(self.faceStart, ptnum1),GetPoint(self.faceStart,
ptnum2))
    ptAvgCurrent = AverageTwoPoints(GetPoint(self.faceCurrent, ptnum1),GetPoint(self.faceCurrent,
ptnum2))

    if RelativeTo == "lefteye":
        ptDiffCurrent = DifferenceBetweenPoint(ptAvgCurrent,GetPoint(self.poseCurrent, 16))
        ptDiffStart = DifferenceBetweenPoint(ptAvgStart,self.StartLeftEyePosition)
    elif RelativeTo == "righteye":
        ptDiffCurrent = DifferenceBetweenPoint(ptAvgCurrent,GetPoint(self.poseCurrent, 15))
        ptDiffStart = DifferenceBetweenPoint(ptAvgStart,self.StartRightEyePosition)
    else:
        ptDiffCurrent = DifferenceBetweenPoint(ptAvgCurrent,GetPoint(self.faceCurrent, 30))
        ptDiffStart = DifferenceBetweenPoint(ptAvgStart,self.StartFaceNosePosition)


    DeltaPixels = DifferenceBetweenPoint(ptDiffStart, ptDiffCurrent)
    return DeltaPixels

  def getLowerLipCenterPosition(self, correction):
    output = self.ConvertPixelPointToBlenderUnits(self.getFacePosition(57,66, "nose"))
    return multiply(output, correction)

  def getLowerLipCenterLeftPosition(self, correction):
    output = self.ConvertPixelPointToBlenderUnits(self.getFacePosition(56,65, "nose"))
    return multiply(output, correction)

  def getLowerLipCenterRightPosition(self, correction):
    output = self.ConvertPixelPointToBlenderUnits(self.getFacePosition(58,67, "nose"))
    return multiply(output, correction)

  def getUpperLipCenterPosition(self, correction):
```

```
    output = self.ConvertPixelPointToBlenderUnits(self.getFacePosition(51,62, "nose"))
    return multiply(output, correction)

  def getUpperLipCenterLeftPosition(self, correction):
    output = self.ConvertPixelPointToBlenderUnits(self.getFacePosition(52,63, "nose"))
    return multiply(output, correction)

  def getUpperLipCenterRightPosition(self, correction):
    output = self.ConvertPixelPointToBlenderUnits(self.getFacePosition(50,61, "nose"))
    return multiply(output, correction)


  def getLipLeftPosition(self, correction):
    output = self.ConvertPixelPointToBlenderUnits(self.getFacePosition(54,64, "nose"))
    return multiply(output, correction)

  def getLipRightPosition(self, correction):
    output = self.ConvertPixelPointToBlenderUnits(self.getFacePosition(48,60, "nose"))
    return multiply(output, correction)



  def getEyeBrowLeftOuter(self, correction):
    output = self.ConvertPixelPointToBlenderUnits(self.getFacePosition(25,26, "lefteye"))
    return multiply(output, correction)

  def getEyeBrowLeftCenter(self, correction):
    output = self.ConvertPixelPointToBlenderUnits(self.getFacePosition(24,24, "lefteye"))
    return multiply(output, correction)

  def getEyeBrowLeftInner(self, correction):
    output = self.ConvertPixelPointToBlenderUnits(self.getFacePosition(22,23, "lefteye"))
    return multiply(output, correction)



  def getEyeBrowRightOuter(self, correction):
    output = self.ConvertPixelPointToBlenderUnits(self.getFacePosition(17,18, "righteye"))
    return multiply(output, correction)

  def getEyeBrowRightCenter(self, correction):
    output = self.ConvertPixelPointToBlenderUnits(self.getFacePosition(19,19, "righteye"))
    return multiply(output, correction)

  def getEyeBrowRightInner(self, correction):
    output = self.ConvertPixelPointToBlenderUnits(self.getFacePosition(20,21, "righteye"))
    return multiply(output, correction)


  def getEyelidRight(self, correction):
    if self.TieEyelidsTogether:
        eyeR = self.getFacePosition(37,38, "righteye")
        eyeL = self.getFacePosition(43,44, "lefteye")
        output = self.ConvertPixelPointToBlenderUnits(AverageTwoPoints(eyeR,eyeL))
    else:
        output = self.ConvertPixelPointToBlenderUnits(self.getFacePosition(37,38, "righteye"))
    return multiply(output, correction)


  def getEyelidLeft(self, correction):
    if self.TieEyelidsTogether:
        eyeR = self.getFacePosition(37,38, "righteye")
        eyeL = self.getFacePosition(43,44, "lefteye")
        output = self.ConvertPixelPointToBlenderUnits(AverageTwoPoints(eyeR,eyeL))
    else:
        output = self.ConvertPixelPointToBlenderUnits(self.getFacePosition(43,44, "lefteye"))
    return multiply(output, correction)


  def getEyelidLowerRight(self, correction):
    output = self.ConvertPixelPointToBlenderUnits(self.getFacePosition(40,41, "righteye"))
```

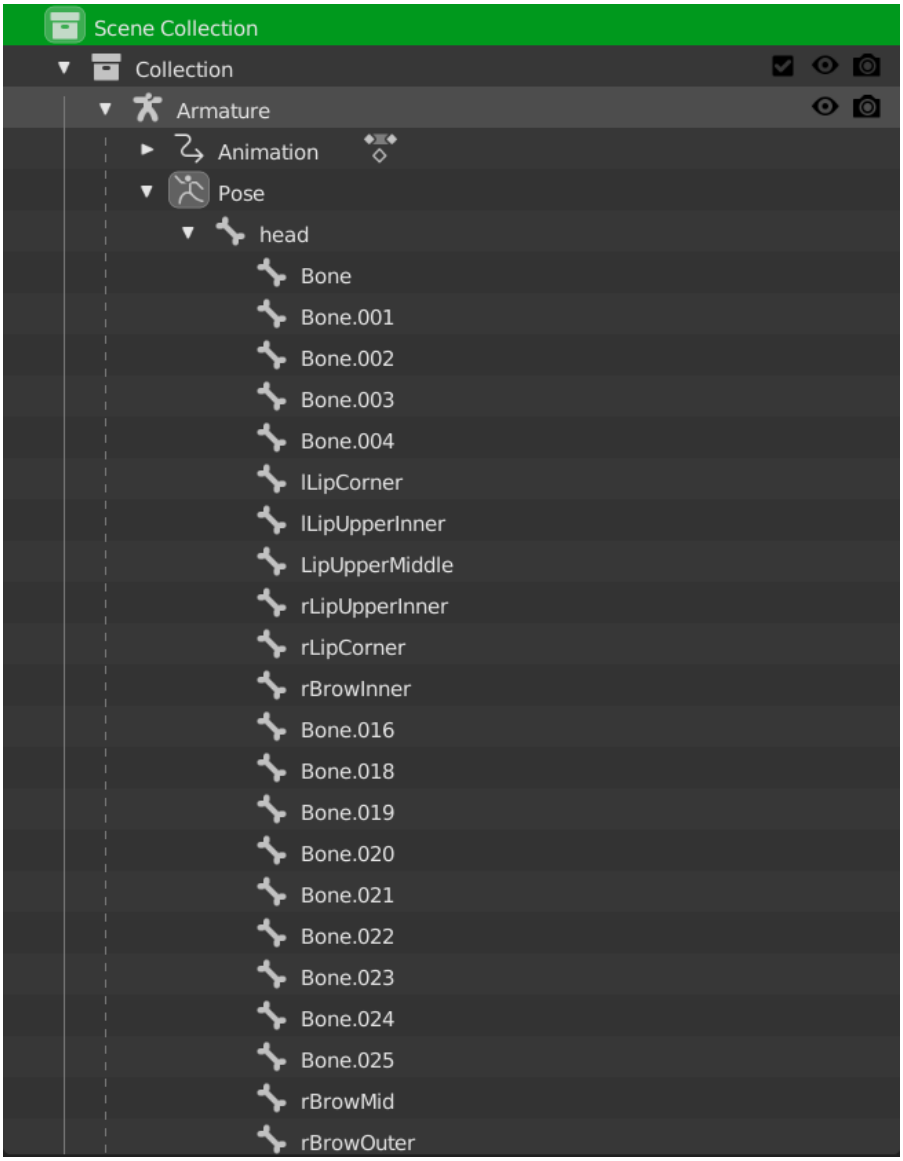
```
    return multiply(output, correction)


  def getEyelidLowerLeft(self, correction):
    output = self.ConvertPixelPointToBlenderUnits(self.getFacePosition(46,47, "lefteye"))
    return multiply(output, correction)



keepGoing = True
filenum = 0
FirstTime = True

print('')
print('Start of Everything')
print('')
DestArm  = bpy.data.objects[DestArmName]
scene = bpy.context.scene
head_bone =  DestArm.pose.bones["head"]
jaw_bone =  DestArm.pose.bones["lowerJaw"]
lower_lip_center =  DestArm.pose.bones["LipLowerMiddle"]
lower_lip_center_l =  DestArm.pose.bones["lLipLowerInner"]
lower_lip_center_r =  DestArm.pose.bones["rLipLowerInner"]
upper_lip_center =  DestArm.pose.bones["LipUpperMiddle"]
upper_lip_center_l =  DestArm.pose.bones["lLipUpperInner"]
upper_lip_center_r =  DestArm.pose.bones["rLipUpperInner"]

corner_lip_r =  DestArm.pose.bones["rLipCorner"]
Corner_lip_l =  DestArm.pose.bones["lLipCorner"]


eyebrow_outer_r =  DestArm.pose.bones["rBrowOuter"]
eyebrow_center_r =  DestArm.pose.bones["rBrowMid"]
eyebrow_inner_r =  DestArm.pose.bones["rBrowInner"]

eyebrow_outer_l =  DestArm.pose.bones["lBrowOuter"]
eyebrow_center_l =  DestArm.pose.bones["lBrowMid"]
eyebrow_inner_l =  DestArm.pose.bones["lBrowInner"]

eyelid_l =  DestArm.pose.bones["Eyelid.l"]
eyelid_r =  DestArm.pose.bones["Eyelid.r"]

eyelid_lower_l =  DestArm.pose.bones["EyelidLower.l"]
eyelid_lower_r =  DestArm.pose.bones["EyelidLower.r"]

PreviousEyelidValue = 0
CurrentFrame = 0

while keepGoing:
    numberString = str(filenum).zfill(12)
    filenum = filenum + 1
    mypath = bpy.path.abspath("//" + FileIdentifier + "_" + numberString + "_keypoints.json")

    if path.exists(mypath):

        print("We are On File: " + str(filenum))

        with open(mypath) as json_file:
            face1_dict = json.load(json_file)

        people = face1_dict['people']
        myPerson = people[0]

        pose = myPerson['pose_keypoints_2d']
        face = myPerson['face_keypoints_2d']

        if FirstTime:
            p1 = PersonJSONData(pose, face, EarLobeToEarLobedistanceInBlenderUnits,
TieEyelidsTogether)
            FirstTime = False
        else:
```

```
            p1.SetCurrentPose(pose, face)

        head_angle =
mathutils.Euler((math.radians(p1.getHeadTiltUpDown()),math.radians(p1.getHeadTiltLeftRight()),mat
h.radians(p1.getHeadTiltSideToSide())), 'XYZ')

        head_bone.rotation_mode = 'XYZ'
        head_bone.rotation_euler = head_angle

        jaw_bone.rotation_mode = 'XYZ'
        jaw_bone.rotation_euler.z = math.radians(p1.getJawTiltUpDown())

        Offset = p1.getLowerLipCenterPosition(0.5)
        lower_lip_center.location.x = Offset[0]
        lower_lip_center.location.z = Offset[1]*-1

        Offset = p1.getLowerLipCenterLeftPosition(1.3)
        lower_lip_center_l.location.x = Offset[0]
        lower_lip_center_l.location.z = Offset[1]*-1

        Offset = p1.getLowerLipCenterRightPosition(1.3)
        lower_lip_center_r.location.x = Offset[0]
        lower_lip_center_r.location.z = Offset[1]*-1

        Offset = p1.getUpperLipCenterPosition(1.3)
        upper_lip_center.location.x = Offset[0]
        upper_lip_center.location.z = Offset[1]

        Offset = p1.getUpperLipCenterLeftPosition(1.3)
        upper_lip_center_l.location.x = Offset[0]
        upper_lip_center_l.location.z = Offset[1]

        Offset = p1.getUpperLipCenterRightPosition(1.3)
        upper_lip_center_r.location.x = Offset[0]
        upper_lip_center_r.location.z = Offset[1]


        Offset = p1.getLipRightPosition(2)
        corner_lip_r.location.x = Offset[0]
        corner_lip_r.location.z = Offset[1]

        Offset = p1.getLipLeftPosition(2)
        Corner_lip_l.location.x = Offset[0]
        Corner_lip_l.location.z = Offset[1]


        eyecorrection = 2
        Offset = p1.getEyeBrowLeftOuter(1)
        eyebrow_outer_r.location.x = Offset[0]
        eyebrow_outer_r.location.z = Offset[1]

        Offset = p1.getEyeBrowLeftCenter(1)
        eyebrow_center_r.location.x = Offset[0]
        eyebrow_center_r.location.z = Offset[1]

        Offset = p1.getEyeBrowLeftInner(1)
        eyebrow_inner_r.location.x = Offset[0]
        eyebrow_inner_r.location.z = Offset[1]


        Offset = p1.getEyeBrowRightOuter(1)
        eyebrow_outer_l.location.x = Offset[0]
        eyebrow_outer_l.location.z = Offset[1]

        Offset = p1.getEyeBrowRightCenter(1)
        eyebrow_center_l.location.x = Offset[0]
        eyebrow_center_l.location.z = Offset[1]

        Offset = p1.getEyeBrowRightInner(1)
        eyebrow_inner_l.location.x = Offset[0]
        eyebrow_inner_l.location.z = Offset[1]
```

```
        Offset = p1.getEyelidLeft(10)
        eyelid_l.location.z = Offset[1]*-1

        Offset = p1.getEyelidRight(10)
        eyelid_r.location.z = Offset[1]*-1

        Offset = p1.getEyelidLowerLeft(1)
        eyelid_lower_l.location.z = Offset[1]*-1

        Offset = p1.getEyelidLowerRight(1)
        eyelid_lower_r.location.z = Offset[1]*-1


        AmountTheEyelidChanged = abs(eyelid_l.location.z - PreviousEyelidValue)
        if AmountTheEyelidChanged > EyelidBlinkTHreshold and keyFrame:
            eyelid_r.keyframe_insert(data_path='location',frame= StartFrameNumber + CurrentFrame)
            eyelid_l.keyframe_insert(data_path='location',frame= StartFrameNumber + CurrentFrame)

        PreviousEyelidValue = eyelid_l.location.z

        if (CurrentFrame % mouth_KeyFrame_Every_Nth_Frame == 0) and (keyFrame):
            lower_lip_center.keyframe_insert(data_path='location',frame= StartFrameNumber +
CurrentFrame)
            lower_lip_center_r.keyframe_insert(data_path='location',frame= StartFrameNumber +
CurrentFrame)
            lower_lip_center_l.keyframe_insert(data_path='location',frame= StartFrameNumber +
CurrentFrame)
            upper_lip_center.keyframe_insert(data_path='location',frame= StartFrameNumber +
CurrentFrame)
            upper_lip_center_r.keyframe_insert(data_path='location',frame= StartFrameNumber +
CurrentFrame)
            upper_lip_center_l.keyframe_insert(data_path='location',frame= StartFrameNumber +
CurrentFrame)
            corner_lip_r.keyframe_insert(data_path='location',frame= StartFrameNumber +
CurrentFrame)
            Corner_lip_l.keyframe_insert(data_path='location',frame= StartFrameNumber +
CurrentFrame)

        if (CurrentFrame % eyes_KeyFrame_Every_Nth_Frame == 0) and (keyFrame):
            eyebrow_outer_l.keyframe_insert(data_path='location',frame= StartFrameNumber +
CurrentFrame)
            eyebrow_center_l.keyframe_insert(data_path='location',frame= StartFrameNumber +
CurrentFrame)
            eyebrow_inner_l.keyframe_insert(data_path='location',frame= StartFrameNumber +
CurrentFrame)

            eyebrow_outer_r.keyframe_insert(data_path='location',frame= StartFrameNumber +
CurrentFrame)
            eyebrow_center_r.keyframe_insert(data_path='location',frame= StartFrameNumber +
CurrentFrame)
            eyebrow_inner_r.keyframe_insert(data_path='location',frame= StartFrameNumber +
CurrentFrame)

            eyelid_r.keyframe_insert(data_path='location',frame= StartFrameNumber + CurrentFrame)
            eyelid_l.keyframe_insert(data_path='location',frame= StartFrameNumber + CurrentFrame)

            eyelid_lower_r.keyframe_insert(data_path='location',frame= StartFrameNumber +
CurrentFrame)
            eyelid_lower_l.keyframe_insert(data_path='location',frame= StartFrameNumber +
CurrentFrame)


        CurrentFrame = CurrentFrame + 1
        bpy.context.scene.frame_set(StartFrameNumber + CurrentFrame)
        if CurrentFrame > NumberOfFramesToTransfer:
            keepGoing = False
    else:
        keepGoing = False
```

2. IMENA KOSTIJU

U nastavku su dana imena kostiju ukupne armature. Kosti bez imena nisu služile za animaciju.




Eyelid.r
EyelidLower.r
lBrowInner
lBrowMid
lBrowOuter
Eyelid.l
EyelidLower.l
LowerFace
lowerJaw
Bone.005
lLipLowerInner
LipLowerMiddle
rLipLowerInner