Mehanizam za usmjeravanje pogleda virtualnog agenta temeljem zvučnog i vizualnog ulaza

Bakula, Slaven

Master's thesis / Diplomski 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:235:801522

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Slaven Bakula

Zagreb, 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Mentor:

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

Student:

Slaven Bakula

Zagreb, 2022.

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

Zahvaljujem se profesoru Tomislavu Stipančiću na mentorstvu te Leonu Korenu i Andriji Ričku na pomoći i savjetima pri izradi ovog diplomskog rada.

Zahvaljujem se svojoj obitelji, posebno roditeljima na podršci tijekom studiranja te svojim prijateljima koji su mi ovo razdoblje učinili nezaboravnim.

Zahvaljujem se također djevojci Karli na razumijevanju i potpori za vrijeme studiranja i pisanja diplomskog rada.

Slaven Bakula

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

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum:	Prilog:
Klasa: 602-14/22-6/1	
Ur. broj: 15-1703-22-	

DIPLOMSKI ZADATAK

Student: **SLAVEN BAKULA** Mat. br.: 0035208867

Naslov rada na hrvatskom jeziku: **Mehanizam za usmjeravanje pogleda virtualnog agenta temeljem zvučnog i vizualnog ulaza**

Naslov rada na engleskom jeziku: **A mechanism for directing the virtual agent's gaze based on sound and visual input**

Opis zadatka:

Virtualni softverski agent PLEA je implementiran u sklopu Laboratorija za projektiranje izradbenih i montažnih sustava. PLEA koristi multi-modalni pristup prilikom zaključivanja o trenutnom emocionalnom stanju osobe s kojom komunicira temeljen na zvučnim i vizualnim informacijama. Korisnici mogu tako komunicirati sa sustavom razmjenjujući neverbalne komunikacijske znakove u vidu različitih izražaja lica pomoću posebnog sučelja.
S gledišta neverbalne komunikacije osoba vrlo često koristi smjer pogleda da ukaže na nešto u okolini. Također, kontakt očima uvelike utječe na kvalitetu ostvarene komunikacije.
U radu je potrebno povezati smjer od kuda dolazi zvuk s pogledom virtualnog softverskog agenta tako da PLEA usmjeri pogled u tome smjeru. Prilikom akvizicije zvučnih podražaja iz okoline potrebno je koristiti Lyra modul koji sadrži više mikrofona pomoću kojih je moguće odrediti smjer zvuka. Osim toga, potrebno je uključiti funkcionalnost da PLEA pogleda u smjeru lica osobe koju je sustav pronašao koristeći računalni model za zaključivanje o emocionalnom stanju osobe u interakciji. Time će biti ostvarena veza između stvarne i virtualne okoline implementirane koristeći Unreal Engine i Meta Human simulacijske platforme.
Dobiveno softversko rješenje je potrebno eksperimentalno evaluirati uključivši ljudske subjekte i opremu dostupnu u sklopu Laboratorija za projektiranje izradbenih i montažnih sustava.
U radu je potrebno navesti korištenu literaturu te eventualno dobivenu pomoć.

Zadatak zadan: 5. svibnja 2022.
Rok predaje rada: 7. srpnja 2022.
Predviđeni datum obrane: 18. srpnja do 22. srpnja 2022.

Zadatak zadao: doc. dr. sc. Tomislav Stipančić

Predsjednica Povjerenstva: prof. dr. sc. Biserka Runje

SADRŽAJ

POPIS SLIKA

POPIS OZNAKA

Oznaka	Jedinica	Opis
hFOV	°	Horizontalni kut vidnog polja kamere
vFOV	°	Vertikalni kut vidnog polja kamere
hAngle	°	Kut između središta kamere do centra pravokutnika prepoznatog lica

SAŽETAK

U sklopu Laboratorija za projektiranje izradbenih i montažnih sustava implementiran je virtualni softverski agent PLEA koji koristi multi-modalni pristup prilikom zaključivanja o emocionalnom stanju osobe s kojom komunicira temeljen na zvučnim i vizualnim informacijama. Koristeći neverbalni oblik komunikacije, korisnici mogu tako ostvariti kontakt sa virtualnim agentom. Kako bi komunikacija virtualnog agenta bila što sličnija stvarnoj osobi potrebno je implementirati značajke svakodnevnog govora kao što je pogled u oči prilikom komuniciranja. U ovom radu opisano je stvaranje jednog takvog sustava koji može biti implementiran u postojeći PLEA virtualni agent. Rad opisuje kreiranje virtualnog agenta u Unreal Engine programu na udaljenom računalu. U programskom jeziku Python uz korištenje dostupnih biblioteka napisan je program za prepoznavanje lica te određivanje kuta njegovog položaja na području kamere. Preko socket komunikacije šalju se vrijednosti definirane pozicije u stvarnom vremenu i ostvaruje se reakcija virtualnog agenta. Uz vizualnu akviziciju, napravljen je i program koji koristi lokalizaciju izvora zvuka korištenjem LyraTD-MSC modula te se slanjem iznos kuta, također preko socket protokola, ostvaruje reakcija virtualnog agenta.

Ključne riječi: prepoznavanje lica, Unreal Engine, Python, socket protokol

SUMMARY

As part of the Laboratory for the Design of Manufactured and Assembled Systems, a virtual software agent PLEA has been implemented, which uses a multimodal approach when inferring the emotional state of the person with whom it communicates based on sound and visual information. Using a non-verbal form of communication, users can make contact with a virtual agent. In order for the virtual agent's communication to be as similar as possible to a real person, it is necessary to implement features of everyday speech such as eye contact when communicating. This paper describes the creation of one such system that can be implemented in the existing PLEA virtual agent. The paper describes the creation of a virtual agent in the Unreal Engine program on a remote computer. A program for face recognition and determination of the angle of its position in the camera area was written in the Python programming language with the use of available libraries. Through socket communication, the values of the defined positions are sent in real time and the reaction of the virtual agent is realized. In addition to visual acquisition, a program that uses the localization of the sound source was created using the LyraTD-MSC module, and by sending the value of the angle, also via the socket protocol, the reaction of the virtual agent is realized.

Key words: face recognition, Unreal Engine, Python, socket protocol

1 UVOD

Razvojem robotike i tehnologije općenito omogućeno je stvaranje i nekih donedavno nezamislivih ili barem teško ostvarivih projekata. Izgradnja realističnih virtualnih ljudi tradicionalno je bio veliki izazov koji je zahtijevao profesionalni studio za snimanje pokreta i ogromne resurse u 3D animaciji i dizajnu. Današnja tehnologija omogućila je stvaranje nevjerojatno realističnih virtualnih ljudi korištenjem osobnog računala. Uz to što su i vjeran prikaz razvoja ljudskih dostignuća u području računarstva, virtualni ljudi mogu imati i brojne korisne funkcije, kao što je ostvarivanje komunikacije ljudi koji nemaju mogućnost komuniciranja s drugim ljudima. Cilj diplomskog rada je dizajniranje virtualnog čovjeka te njegovo povezivanje s okolinom. Virtualni čovjek okolinom je povezan preko sustava mikrofona koji omogućuju određivanje smjera te glasnoće izvora zvuka. Na osnovu informacija koje virtualni čovjek prima, odnosno vrijednosti kuta koje se pošalju, ostvaruje se zakret prema izvoru zvuka. Drugi način povezivanja s okolinom je preko vizualnog unosa preko kamere. Pritom se određuje položaj prepoznatog lica u području kamere i na osnovu poslanih vrijednosti kuta položaja ostvaruje se reakcija virtualnog agenta.

2 KOMPONENTE I PROGRAMSKI PAKETI SUSTAVA

2.1 Unreal Engine

Unreal Engine je program za razvijanje računalnih igara razvijen od strane Epic Gamesa. Prva igra koju su razvili zvala se Unreal, koja je i porijeklo imena programa, i predstavljena je 1988. godine. Program je najprije osmišljen za razvijanje pucačkih igrica u prvom licu, a nakon toga je omogućavao bilo koji žanr igara. Neke od brojnih poznatih igrica koje su razvijene u Unreal Engine programu su: PUBG: New State, Fortnite, Tekken 7, Medal of Honor: Airborne, Shadow Warrior 3, Tony Hawk's Pro Skater 1+2 i mnoge druge. Postoji nekoliko verzija softvera otkako je predstavljena originalna verzija 1988. godine. Unreal Engine 2 predstavljen je 2002., Unreal Engine 3 2006, Unreal Engine 4 2014, dok je najnovija verzija programa, Unreal Engine 5 predstavljen u travnju 2022. godine. [1]

2.1.1 Prva generacija

Prvu generaciju UEa razvio je Tim Sweeney, osnivač Epic Gamesa. Sweeney je 1995. počeo pisati motor za igru Unreal. U početku se motor u potpunosti oslanjao na softversko renderiranje, što znači da je grafičkim izračunima upravljao CPU. S vremenom je uspio iskoristiti mogućnosti koje pružaju namjenske grafičke kartice. Među značajkama koje je posjedovao motor Unreal Enginea bile su detekcija sudara, osvjetljenje u boji, filtriranje teksture. Jedan od značajniji noviteta koje je Unreal posjedovao bila je i korištenje 16-bitne boje za razliku od 8-bitne koju je koristila konkurencija. Razvojem *Unreal Tournament*a, Epic Games se odlučio riješiti problema vezanih za potrebom visokoperformantnih računala, kao i problema vezanih za online igranje. Neke od igara koje su koristile tadašnju Epic-ovu tehnologiju su: The Wheel of Time, Duke Nukem Forever; Deus Ex i dr.[1]

Slika 1. Scena iz Wheel of Time video igre (1999) [13]

2.1.2 Druga generacija

Razvoj druge generacije motora započeo je 1999. godine, a predstavljen je 2002. godine igrom American's Army. Motor se temeljio na svom prethodniku, uz napredak u uvjetima renderiranja, te poboljšanja u već postojećim alatima i značajkama. Predstavljene su i neke nove značajke poput alata za kinematsko uređivanje, sustave čestica, dodatke za izvoz u Maya i 3D Studio Max, animaciju kostura. Epic je koristio Karma motor za fizičke simulacije koji je imao dosta ograničenja koja su popravljena tek Psyonix-ovom modifikacijom Epicovog osnovnog koda. Pysonix kasnije i razvija mega 0popularnu igru Rocket League koju 2019. godine kupuje Epic Games.[1]

Slika 2. Scena iz Unreal Tournamenta 2003 [14]

2.1.3 Unreal Engine 3

Snimke zaslona Unreal Engine 3 predstavljene su u srpnju 2004. godine, kada je motor već bio u razvoju više od 18 mjeseci. Temeljio se na prvoj generaciji, s novim značajkama. Osnovne arhitekturne značajke vidljive programerima ostale su jako slične prvoj verziji, za razliku od onoga što vide igrači gdje su se dogodile velike promjene - zvučni sustav, renderer, sustav fizike. Svi proračuni osvjetljenja rađeni su po pikselu (eng per-pixel), umjesto po verteksu (eng per-vertex). Incicijalno je podržavao samo Windows, PlayStation 3 i Xbox 360, dok su iOS i Android dodani 2010. godine. Tijekom životnog vijeka UE3 uključena su značajna ažuriranja, uključujući okruženja koja se mogu uništiti, dinamiku mekog tijela (eng. soft body dynamics), simulaciju velike gužve , iOS funkcionalnost, integraciju Steamworksa, rješenje globalnog osvjetljenja u stvarnom vremenu i stereoskopski 3D na Xbox 360 putem TriOviz for Gamesa.[1]

Slika 3. Gears of War igrica razvijena u UE 3 [15]

2.1.4 Unreal Engine 4

Prva verzija Unreal Engine 4 predstavljena je ograničenom broju korisnika 2012. godine. Jedna od najvažnijih značajki vezano za UE4 bilo je globalno osvjetljenje u stvarnom vremenu korištenjem praćenja voksela, SVOG (Sparse Voxel Octree Global Illumination). UE4 uključuje novi sustav vizualnog skriptiranja, tzv. „Blueprints“, nasljednik UE3-ovog „Kismeta“, koji omogućuje razvoj logike bez korištenja programskog koda. 2012. godine Epic je objavio uklanjanje UnrealScripta iz Unreal Enginea koji biva zamijenjen sa C++ programskim jezikom.[1]

Slika 4. Fortnite, mega popularna igra razvijena u UE4[16]

2.1.5 Unreal Engine 5

Unreal Engine 5 predstavljen je 13. svibnja 2020. Rad na motoru započeo je oko dvije godine prije njegove objave. Objavljen je u ranom pristupu 26. svibnja 2021. i službeno lansiran za

programere 5. travnja 2022. Jedna od njegovih glavnih značajki je Nanite, motor koji omogućuje uvoz visokodetaljnog fotografskog izvornog materijala u igre. Nanite može uvesti gotovo bilo koji drugi već postojeći trodimenzionalni prikaz objekata i okruženja, uključujući ZBrush i CAD modele. Lumen je još jedna komponenta opisana kao "potpuno dinamično globalno rješenje osvjetljenja koje odmah reagira na promjene prizora i svjetla". Dodatne značajke planirane za Unreal Engine 5 dolaze iz Epicovih akvizicija i partnerstava. MetaHuman Creator je projekt temeljen na tehnologiji triju tvrtki koje su kupili Epic, 3Lateral, Cubic Motion i Quixel, kako bi se programerima omogućilo da brzo kreiraju realistične ljudske likove koji se zatim mogu izvesti za korištenje unutar Unreala. Kroz partnerstvo s Cesiumom, Epic planira ponuditi besplatni dodatak za pružanje 3D geoprostornih podataka za Unreal korisnike, dopuštajući im da rekreiraju bilo koji dio mapirane površine Zemlje. Epic će uključivati RealityCapture , proizvod koji može generirati 3D modele bilo kojeg objekta iz zbirke fotografija snimljenih iz više kutova.[1]

2.1.6 Sučelje Unreal Enginea

Slika 5. Unreal Engine sučelje

Dijelovi Unreal Engine sučelja su:

1) Traka izbornika (eng. Menu Bar)
2) Glavna traka (eng. Main Toolbar)
3) Prikaz razine (eng. Level Viewport)
4) Outliner
5) Ploča detalja (eng. Details Panel)
6) Pretraživač sadržaja (eng. Content Browser)

2.2 Terminologija korištena u Unreal Engine-u

Unreal Engine posjeduje skup specifičnih termina i izraza koje treba znati kako bi mogli razumjeti i koristiti program. Ovdje ćemo nabrojati neke koji su bitni za ovaj zadatak.

Projekt (eng. project) sadrži sav sadržaj, odnosno sadrži mnoštvo dokumenata na disku, poput Blueprintsa i Materiala. Pretraživač sadržaja (eng. Content Beowser) sadrži istu strukturu direktorija kao dokument Project na disku.

Blueprint predstavlja sustav vizualnog skriptiranja koji koristi sučelje bazirano na čvorovima za stvaranje gameplay elemenata unutar Unreal Editora. Umjesto klasičnog programiranja, redak po redak, sve se obavlja vizualno: povucite i ispustite čvorove, postavite njihova svojstva u korisničkom sučelju i povucite žice za povezivanje.

Objekt (eng, Object) najjednostavniji je oblika klase u Unreal Engineu. U suštini, djeluju kao blokovi za građenje i sadrže osnovne funkcije. Gotovo sve u Unreal Engineu potječe iz objekata.

Klasa (eng. Class) određuje svojstva i ponašanje objekta ili Actora u Unreal Engineu. Klase su hijerarhijski raspoređene, što znači da klasa potječe iz svoje roditeljske klase (eng. parent class).

Actor je bilo koji objekt smješten unutar razine (eng. Level), poput statičke mreže (eng. static mesh), startne pozicije igrača, kamere, osvjetljenja… Mogu biti stvoreni ili uništeni kodiranjem gameplaya (ili preko Blueprintsa ili C++ programiranjem).

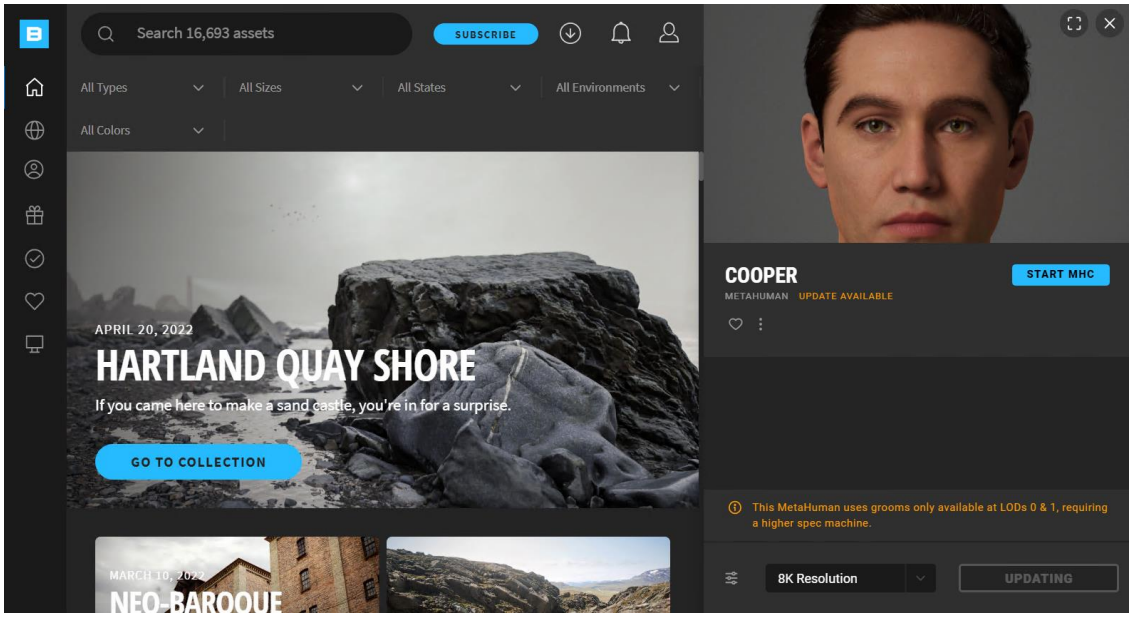
Komponente (eng. Components) dio su funkcionalnosti koji mogu biti dodani Actoru. Kada dodamo komponentu Actoru, on koristi funkcionalnosti koje pružaju komponente.

Razina (eng. Level) predstavlja područje gameplaya koji definiramo. Razine sadrže sve što igrač može vidjeti ili s čime može ostvariti interakciju.

Svijet (eng. World) sadrži sve razine koje čine igricu. Riječ igrica uporabljena je iz razloga što se Unreal Engine u principu koristi za razvijanje video igrica.

2.3 Quicksel Bridge

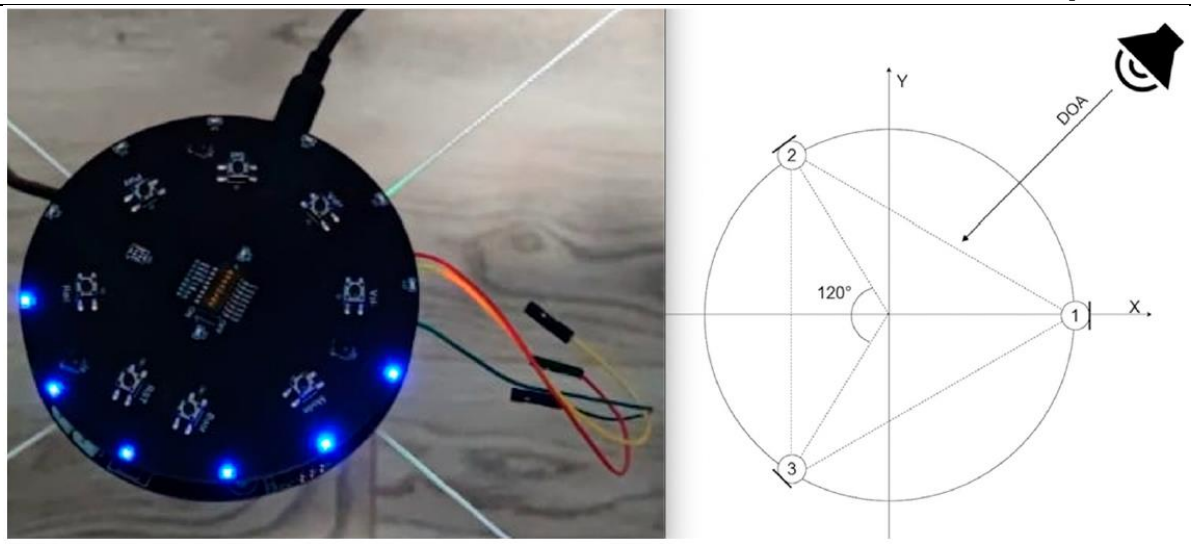
Quicksel Bridge je desktop aplikacija koja služi za pretraživanje, skidanje, uvoz i izvoz Megascan instanci. Osim za Unreal Engine 4, kompatibilan je i sa drugim alatima i programima, kao što su Unity 3d, 3D Studio Max, Autodesk Maya, SideFX Houdini, Blender, Maxon Cinema 4D, Isotropix Clarisse, Marmoset TB3.



Slika 6. Quicksel Bridge sučelje

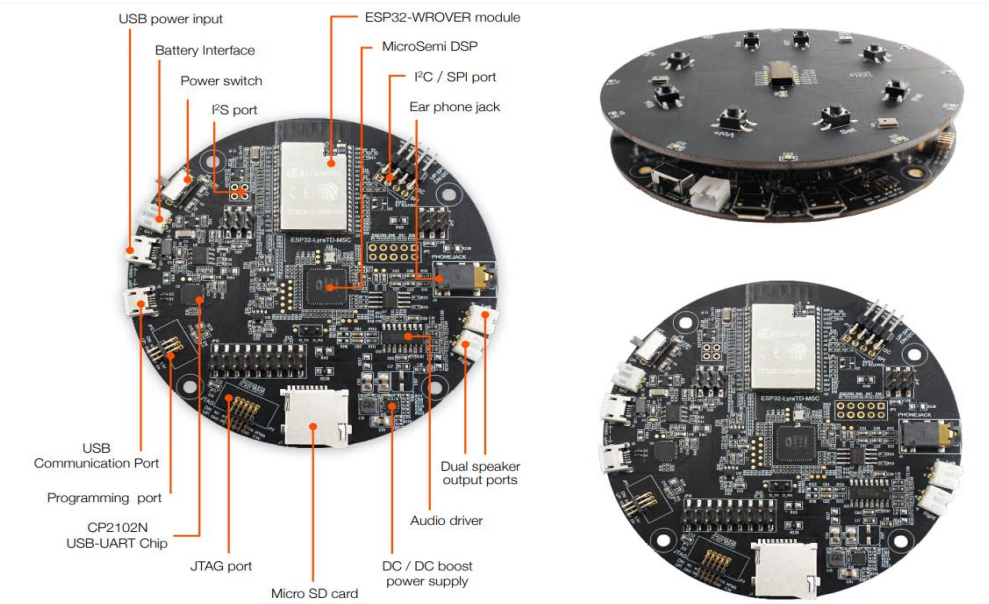
2.4 LyraTD-MSC modul

LyraTD-MSC modul služi za akviziciju zvuka i estimaciju smjera dolaska zvuka (DOA), eng. direction of arrival. Modul se sastoji od 3 mikrofona smještena na istom radijusu i pod kutom od 120˚ između 2 susjedna mikrofona. Modul je također odgovoran i za filtriranje te konvertiranje zvuka u digitalni format. Posjeduje mogućnost i prepoznavanja govora te komunikacije.



Slika 7. Računanje kuta zvuka LyraTD-MSC modulom

Softver koji kontrolira procesuiranje zvučnih signala i kuta smjera izvora zvuka (DOA) napisan je u C programskom jeziku koristeći ESP IoT Development Framework. Ulazni DOA kut filtriran je niskoprolaznim filterom implementiranim u mikrokontroler. Za određivanje smjera izvora zvuka korišten je čip za digitalnu obradu signala (DSP, eng. digital signal processing) s unaprijed definiranim algoritmom za izračunavanje DOA kuta. Algoritam se temelji na principu po kojem zvučni valovi prije dopiru do jednog mikrofona od drugih. Izračunavanje kuta vrši se korištenjem zakona trigonometrije. Filtri su projektirani kao kontinuirani, te su transformirani u diskretni sustav pomoću bilinearne transformacije. Granične frekvencije su empirijski odabrane iz podataka na temelju prosječnog vremena ljudske interakcije, što je empirijski određeno.



Slika 8 Priključci LyraTD-MSC modula [3]

2.5 Python programski jezik

Python je objektno orijentirani programski jezik koju je razvio Guido van Rossum 1991. godine. Objektno orijentirano programiranje (OOP) predstavlja računalni model programiranja koji organizira dizajn softvera oko podataka ili objekata, umjesto funkcija i logike. Objekt označava polje podataka koji ima jedinstvene atribute i ponašanje. Ovakav pristup manipuliranja objektima prikladan je za kompleksne i velike programe koji su aktivno ažurirani i održavani. Prednosti objektno orijentiranog programiranja su:

- Modularnost - enkapsulacija omogućuje objektima da budu samostalni, što olakšava rješavanje problema i zajednički razvoj.
- Ponovna upotreba - kod se može ponovno koristiti nasljeđivanjem, što znači da tim ne mora pisati isti kod više puta.
- Produktivnost - programeri mogu brže konstruirati nove programe korištenjem više knjižnica i koda koji se može ponovno koristiti.
- Lako nadogradiv i skalabilan - programeri mogu samostalno implementirati funkcionalnosti sustava..
- Sigurnost - Korištenjem enkapsulacije i apstrakcije složeni kod je skriven, održavanje softvera je lakše i internetski protokoli su zaštićeni.
- Fleksibilnost - polimorfizam omogućuje da se jedna funkcija prilagodi klasi u koju je smještena. Različiti objekti također mogu proći kroz isto sučelje. [4]

Mnoge su mogućnosti uporabe Python programskog jezika. Može se koristiti na serveru za kreiranje web aplikacija; može se povezati sa sustavima baze podataka te čitati i mijenjati datoteke; koristi se i za rukovanje velikim pdacima te za izvođenje složenih matematičkih operacija; također se uporabljuje za brzo izvođenje prototipa (eng. rapid prototyping) ili za razvoj softvera za proizvodnju. [5]

2.6 Socket komunikacija

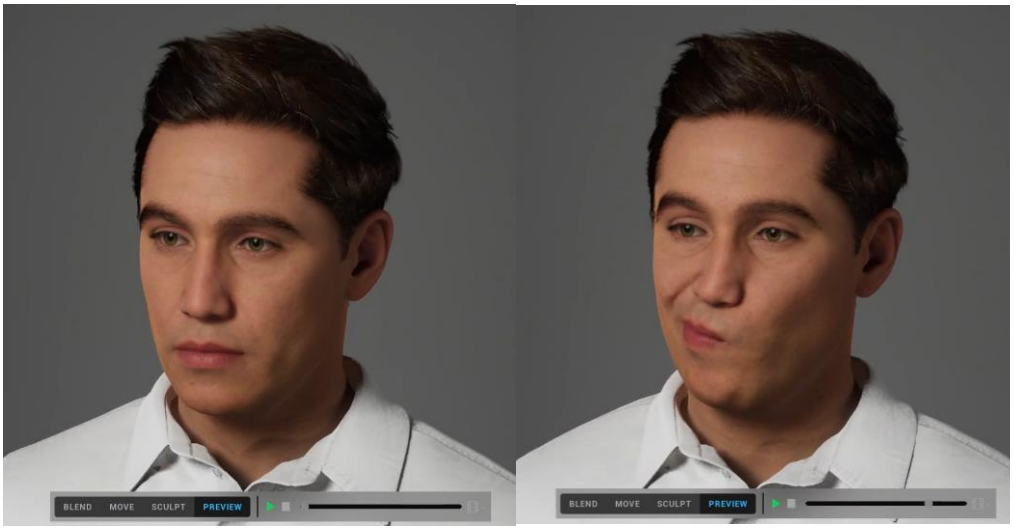
Socketi pružaju sučelje za programiranje mreža na transportnom sloju. Mrežna komunikacija pomoću socketa jako je slična izvođenju U/I datoteka. Mrežni socketi su softverske strukture unutar mrežnog čvora (eng. network node) računalne mreže i služe kao krajnja točka mrežnog slanja i primanja podataka. Riječ socket najčešće se odnosi na internet socket ili TCP socket. Socket je definiran socket adresom koju sačinjavaju transportni protokol, IP adresa i broj porta.

Komunikacija se vrši između servera i clienta, gdje client potražuje od servera zahtjev za spajanjem. Server je program koji je pokrenut na nekom računalu i sadrži socket vezan za specifični port. Server čeka i sluša slanje zahtjeva clienta prema socketu.

Slika 9. Socket protokol

3 KREIRANJE VIRTUALNOG ČOVJEKA

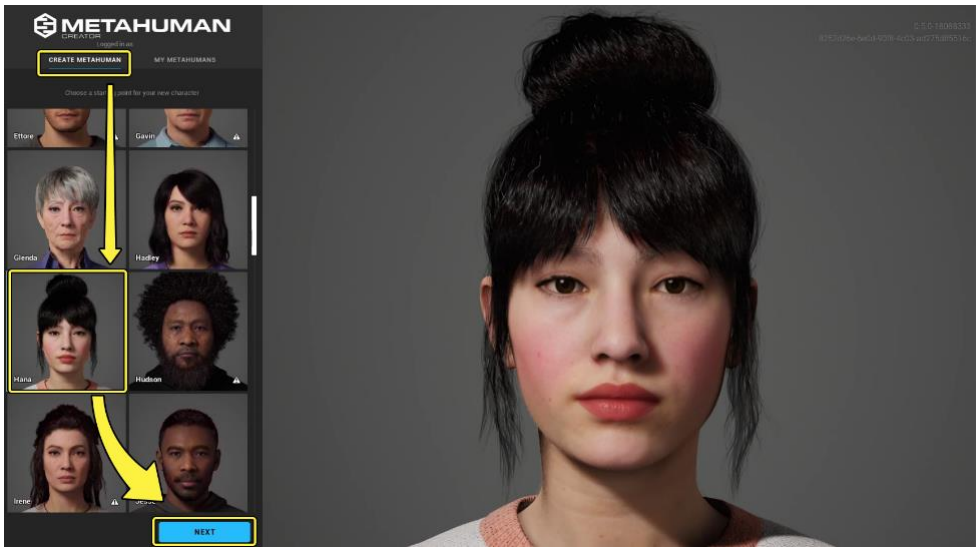
Kreiranje virtualnog čovjeka predstavlja prvu fazu u izradi projekta. Samo kreiranje digitalnog čovjeka izvršeno je pomoću MetaHuman Creatora. MetaHuman Creator je web alat koji omogućuje stvaranje vlastitih digitalnih ljudi na intuitivan i jednostavan način.



Slika 10. Izgled virtualnog čovjeka

3.1 MetaHuman Creator

MetaHuman Creator je besplatni alat koji se pokreće na oblaku (eng. cloud) i reproducira se na internet pretraživač pomoću tzv. *pixel streaming* tehnologije. Pomoću MetaHuman Creatora moguće je urediti tjelesne proporcije, facijalne karakteristike, kosu i mnogo toga. MetaHuman Creator posjeduje niz različitih preddefiniranih likova, od kojih odabiremo jednog kao polazišnu točku za stvaranje vlastitog virtualnog čovjeka, odnosno MetaHumana.



Slika 11. Odabir početnog lika

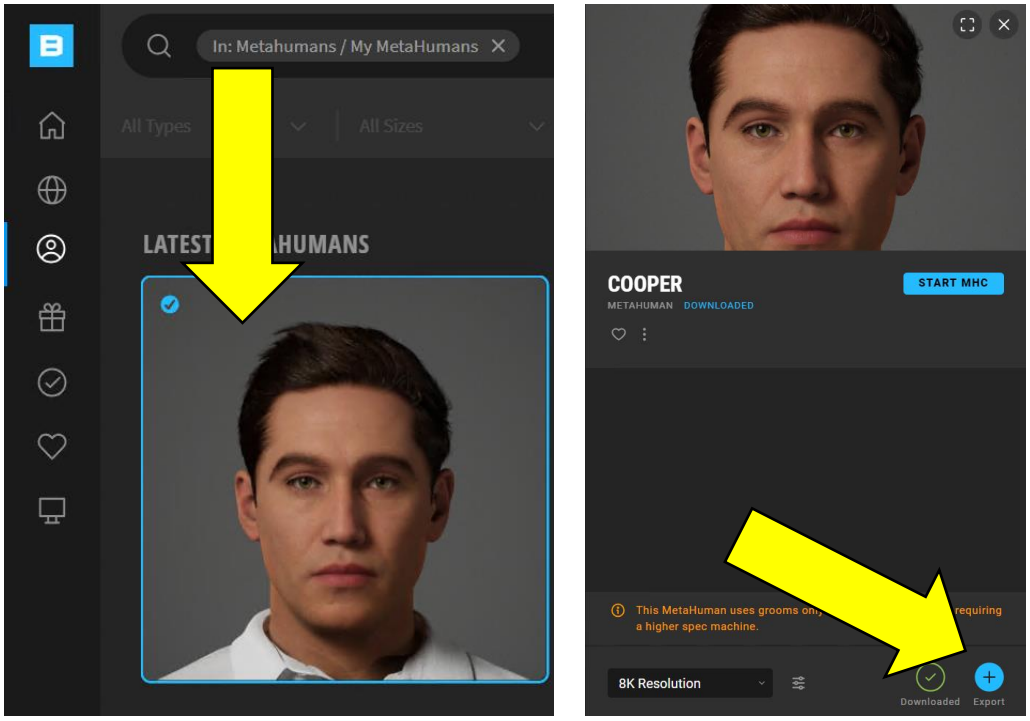
Slika 12. MetaHuman Creator sučelje

Na slici 3 prikazano je sučelje MetaHuman Creatora. Sučelje se sastoji od sljedećih dijelova:

1) Naslovna traka - Sadrži brzu vezu za povratak na galeriju My MetaHumans, ime vašeg lika i gumbe Pomoć, Poništi (eng. Undo) i Ponovi (eng. Redo).
2) Izbor atributa i svojstava MetaHumana - Sadrži sve prilagodljive atribute za vaše MetaHumans, podijeljene u tri velike grupe: lice, kosa i tijelo. Svaki od ovih atributa može se dodatno prilagoditi postavljanjem različitih svojstava.
3) Osvjetljenje vidnog polja i postavke kvalitete - Sadrži kontrole osvjetljenja i okoliša, postavke kamere, renderiranje i LOD (Level of Details) kontrole, gumbe za uključivanje/isključivanje kose i glinenog materijala kao i prečac za skrivanje referentne ploče s prečacima.
4) Renderirani pregled MetaHumana - Pregledava MetaHuman u stvarnom vremenu.
5) Referentna ploča prečaca - Prikazuje popis kontrola tipkovnice i miša za MetaHuman Creator.
6) Kontrole kiparstva i animacije – Mogućnost odabira između različitih načina uređivanja MetaHumana kao što su miješanje (eng. blend), premještanje (eng. move) i oblikovanje (eng. sculpt). Također sadrži gumbe za pokretanje, pauziranje i zaustavljanje animacija za pregled, kao i vremensku traku pročišćavanja animacije i izbornik u kojem možete odabrati različite animacije za pregled.

3.2 Izvoz virtualnog čovjeka u Unreal Engine

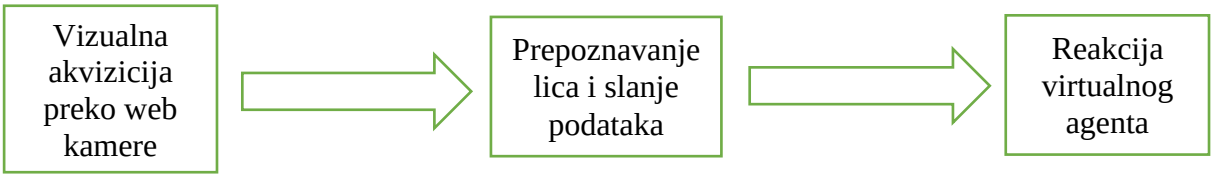
Nakon što se izvrši oblikovanje virtualnog čovjeka potrebno ga je uvesti u glavni program, odnosno u Unreal Engine. Uvoz se vrši pomoću programa Quicksel Bridge. Kako bismo vidjeli svoje MetaHuman kreacije prijavljujemo se istim računom kao u web sučelju MetaHuman Creatora. Prije izvoza, potrebno je najprije pokrenuti glavni program u Unreal Engine-u budući da se izvoz vrši u trenutno otvoreni, aktivni program.



Slika 13. Izvoz MetaHuman-a u UE

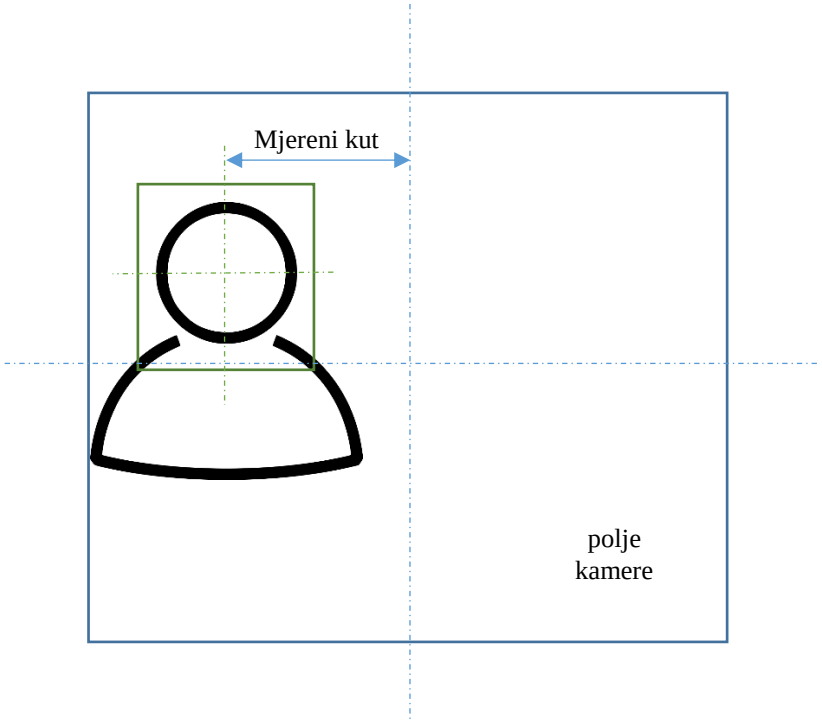
4 VIZUALNA AKVIZICIJA

Shema postupka vizualne akvizicije prikazana je na slici ispod:



Slika 14. Prikaz procesa vizualne akvizicije

Web kamera predstavlja ulazni uređaj za određivanje položaja lica. U programskom paketu Python obrađuje se slika sa web kamere, odnosno prepoznavanje lica te određivanje koordinata položaja lica u horizontalnom smjeru. Preko socket protokola podaci se šalju na udaljeno računalo spojeno preko VPN-a gdje se učitavaju na Unreal Engine projektu. U konačnici, virtualni agent prima brojčane vrijednosti horizontalnog položaja lica te se ostvaruje pomak očiju prema smjeru lica prepoznatog na web kameri.



Slika 15. Princip određivanja položaja lica na kameri

4.1 Python kod za vizualnu akviziciju

Učitavaju se potrebne biblioteke za izvođenje programa, prikazane na slici ispod.

```
import cv2
import sys
import os
import math
import eventlet
import socketio
import threading
```

Slika 16. Učitavanje biblioteka za program detekcije lica

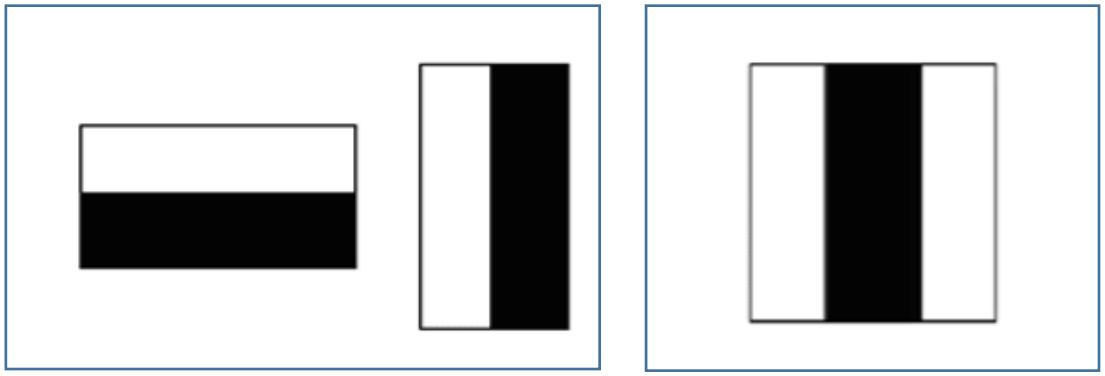
OpenCV (Open Source Computer Vision Library) softverska je biblioteka otvorenog koda za računalni vid i strojno učenje. Napravljen je kako bi osigurao zajedničku infrastrukturu za aplikacije računalnog vida i ubrzao korištenje strojne percepcije u komercijalnim proizvodima. Biblioteka sadrži više od 2500 optimiziranih algoritama, što uključuje opsežan skup klasičnih i najsuvremenijih algoritama računalnog vida i strojnog učenja.[7]

Biblioteka socketio koristi se za transportni protokol koji omogućuje dvosmjernu real-time komunikaciju baziranu na eventima, između klijenata (eng. clients) i servera.

```
def resource_path(relative_path):
    try:
        base_path = sys._MEIPASS
    except Exception:
        base_path = os.path.abspath(".")
    return os.path.join(base_path, relative_path)

import os.path
print(os.path.isfile(resource_path('haarcascade_frontalface_al
t.xml')))
```

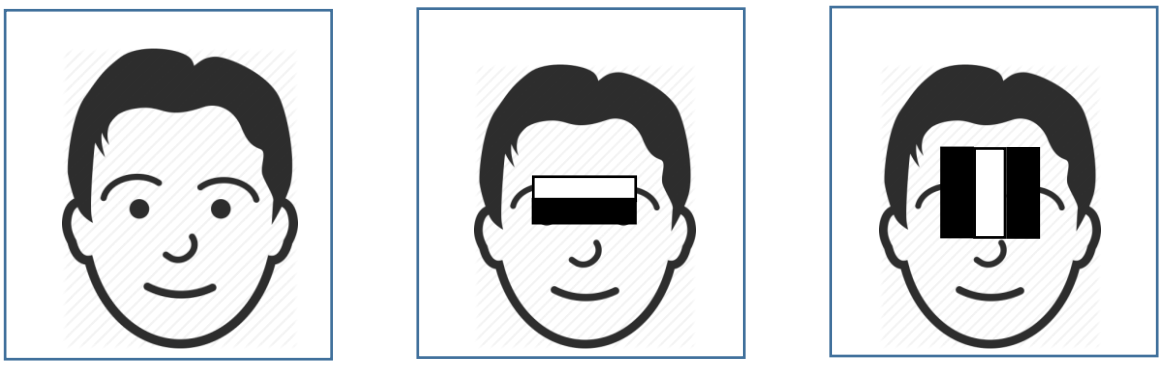
```
face_cascade =
cv2.CascadeClassifier(resource_path('haarcascade_frontalface_a
lt.xml'))
face_cascade.load(resource_path('haarcascade_frontalface_alt.x
ml'))
```

Prethodno prikazani kod služi za učitavanje Haar Cascade algoritma. Koristi se da identificiranje lica na statičnoj slici ili na real-time videu. Algoritam koristi značajke prepoznavanja rubova i linija. Iako je stariji algoritam koji je predstavljen još 2001. godine, i dalje predstavlja dobrog „suparnika“ novijim algoritmima baziranim na konvolucijskim neuronskim mrežama. Repozitorij sadrži modele spremljene u XML formatu koji mogu biti učitani korištenjem OpenCV metoda. Algoritam koristi tzv. Haar značajke za traženje rubova.



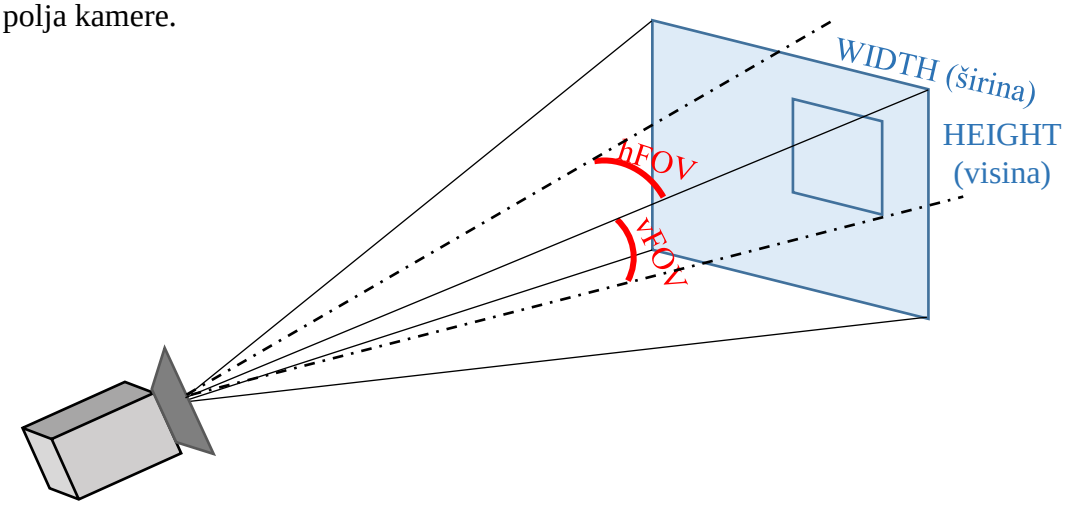
Slika 17. Haar značajke prve faze (lijevo) i Haar značajke druge faze (desno) [8]

Postupak prepoznavanja lica temelji se na dva svojstva lica. Prvo se odnosi na to da je područje oko očiju tamnije od okolnog područja lica. Drugo svojstvo se odnosi na to da je područje oko očiju tamnije od mosta nosa. Prolaskom značajki iz prve faze kroz cijelu sliku nailazi se na potencijalne kandidate lica. Druga faza se ne uključuje sve dok se ne zadovolje uvjeti prve faze. Oba uvjeta moraju biti zadovoljena pretragom kako bi se moglo zaključiti da se radi o licu. **Error! Reference source not found.**



Slika 18. Pronalazak lica primjenom Haar Cascade algoritma

Na sljedećem dijelu koda prikazano je pokretanje web kamere, određivanje rezolucije prikazane slike. Također su i određene varijable koje će biti korištene pri računanju koordinata položaja lica na polju kamere. Varijable hFOV i vFOV označavaju kut horizontalnog i vertikalnog vidnog polja kamere.



Slika 19. Dimenzijske karakteristike kamere

```
cap = cv2.VideoCapture(0)
res = (1280, 720)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, res[0])
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, res[1])
hFOV = 62 / 2
vFOV = 49 / 2
WIDTH = res[0] / 2
HEIGHT = res[1] / 2
```

Kako bi podaci mogli biti poslani na udaljeno računalo te u Unreal Engine projekt potrebno je napraviti server. Za to je korištena prethodno opisana socketio biblioteka.

```
sio = socketio.Server(cors_allowed_origins='*')
app = socketio.WSGIApp(sio)
```

Klasa socketio.Server() stvara server kompatibilan sa Pythonovm standardnom bibliotekom. U dodatku sa serverom potrebno je odabrati i asinkroni framework ili server za korištenje s njim. Socketio.ASGIApp klasa tranformira server u standardnu ASGI aplikaciju. On može djelovati kao međuprogram, prosljeđujući sav promet koji nije namijenjen socketio serveru na drugu aplikaciju, što omogućuje socketio poslužiteljima da se jednostavno integriraju u postojeće

WSGI ili ASGI aplikacije.

```
@sio.event
async def connect(sid, environ):
    print(f'Someone connect with id: {sid} ')

@sio.event
def disconnect(sid):
    print(f'Someone disconnect with id: {sid} ')
```

Eventi connect i disconnect pokreću se automatski kada se klijent povezuje ili prekida vezu sa serverom. Connect event idealno je mjesto za provođenje provjere autentičnosti korisnika i bilo kakvog potrebnog mapiranja između korisničkih entiteta u aplikaciji i sid-a koji je dodijeljen klijentu. Sid predstavlja jedinstveni identifikator koji je dodijeljen svakoj vezi s klijentom. Argument environ je rječnik u standardnom WSGI formatu koji sadrži informacije o zahtjevu, uključujući HTTP zaglavlja. [10]

```
@sio.on('ue_ready')
def another_event(sid, data):
    print(f'ue is connected with id {sid} and is ready to receive
data')
    x = threading.Thread(target=send_frame_data)
    x.start()
```

Stvoren je event naziva ue_ready koji sadrži funkciju nakon spajanja ispisuje da je veza uspostavljena. Funkcija sadrži klasu threading.Thread() kojoj je ulazni argument funkcija send_frame_data(). Naredbom start, thread se ne pokreće odmah, već je na listi čekanja i pokreće se čim prije moguće.

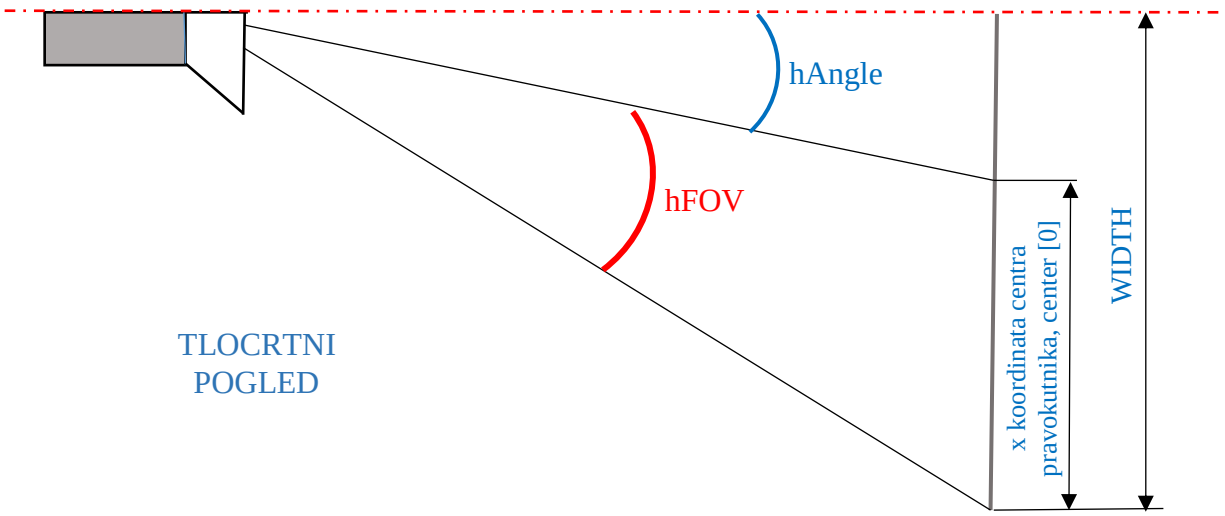
```
def send_frame_data():

    while True:
        # capture frame by frame
```

```
        ret, frame = cap.read()

        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        faces = face_cascade.detectMultiScale(gray,
                                              scaleFactor=1.1,
                                              minNeighbors=5,
                                             minSize=(280,280))
```

Funkcija send_frame_data () definirana je kao beskonačni loop. Metoda cap.read () provjerava je li okvir (eng. frame) ispravno učitan i vraća boolean vrijednost True (ukoliko je ispravno učitan) ili False (ukoliko to nije slučaj). Zbog lakše detekcije rubova slika se prevodi u grayscale i sprema u varijablu gray. Metoda face_cascade.detectMultiScale() koristi se za pronalaženje lica. Ulazni parametri metode su redom: matrica tipa CV_8U koja sadrži sliku na kojoj se prepoznaje lice; scaleFactor je parametar koji specificira koliko se reducira veličina slike pri svakom njenom skaliranju; minNeighbours je parametar koji specificira koliko susjeda treba imati svaki kandidat za pravokutnik kako bi se zadržao (veća vrijednost rezultira u manjem broju detekcija, ali većoj kvaliteti), minSize ograničava najmanju veličinu pravokutnika gdje se svi manji pravokutnici od postavljene vrijednosti ignoriraju. Veličina pravokutnika je dinamična, ovisno o udaljenosti od kamere mijenja se i veličina pravokutnika, veći je ako smo bliže kameri i obrnuto.

```
        for (x, y, w, h) in faces:
            cv2.rectangle(frame,(x,y),(x+w,y+h),(0,255,0),2)
            # x i y koordinata centra
            center = (int(x + 0.5 * w), int(y + 0.5 * h))
            hAngle = (1 - center[0] / WIDTH) * hFOV
            #vAngle = (1 - center[1] / HEIGHT) * vFOV
            angleA = (90 - hAngle) * math.pi / 180
            angleDegrees = angleA * 180 / math.pi
            angle1=(-1/2)*(90-angleDegrees)
```

or petlja definirana je unutar funkcije *send_frame_data* () i prolazi kroz varijablu *faces*. Metoda cv2.rectangle sadrži ulazne parametre, poredane redom: slika na kojoj se prikazuje pravokutnik (u ovom slučaju to je prepoznato lice); početne koordinate pravokutnika (gornji lijevi kut); konačne koordinate pravokutnika (donji lijevi kut); kombinaciju RGB sheme boja te debljinu linija pravokutnika. Zatim se računa centar pravokutnika koji se nalazi oko prepoznatog lica te kut između centra slike i centra pravokutnika.



Slika 20. Izračunavanje kuta od centra slike kamere do centra lica

Kut između centra slike kamere i centra pravokutnika prepoznatog lica određen je na temelju sličnosti trokuta kako slijedi:

$$\frac{WIDTH}{hFOV} = \frac{WIDTH - center[0]}{\text{hAngle}} \tag{1}$$

$$hAngle = \frac{WIDTH - center[0]}{\text{WIDTH}} * hFOV \tag{2}$$

$$hAngle = \left(1 - \frac{center[0]}{\text{WIDTH}}\right) * hFOV \tag{3}$$

Potom se metodom sio.emit()klijentu šalje float varijabla kuta te se prikazuje video.

```
sio.emit('eye_angle', angle1)
            sio.sleep()
```

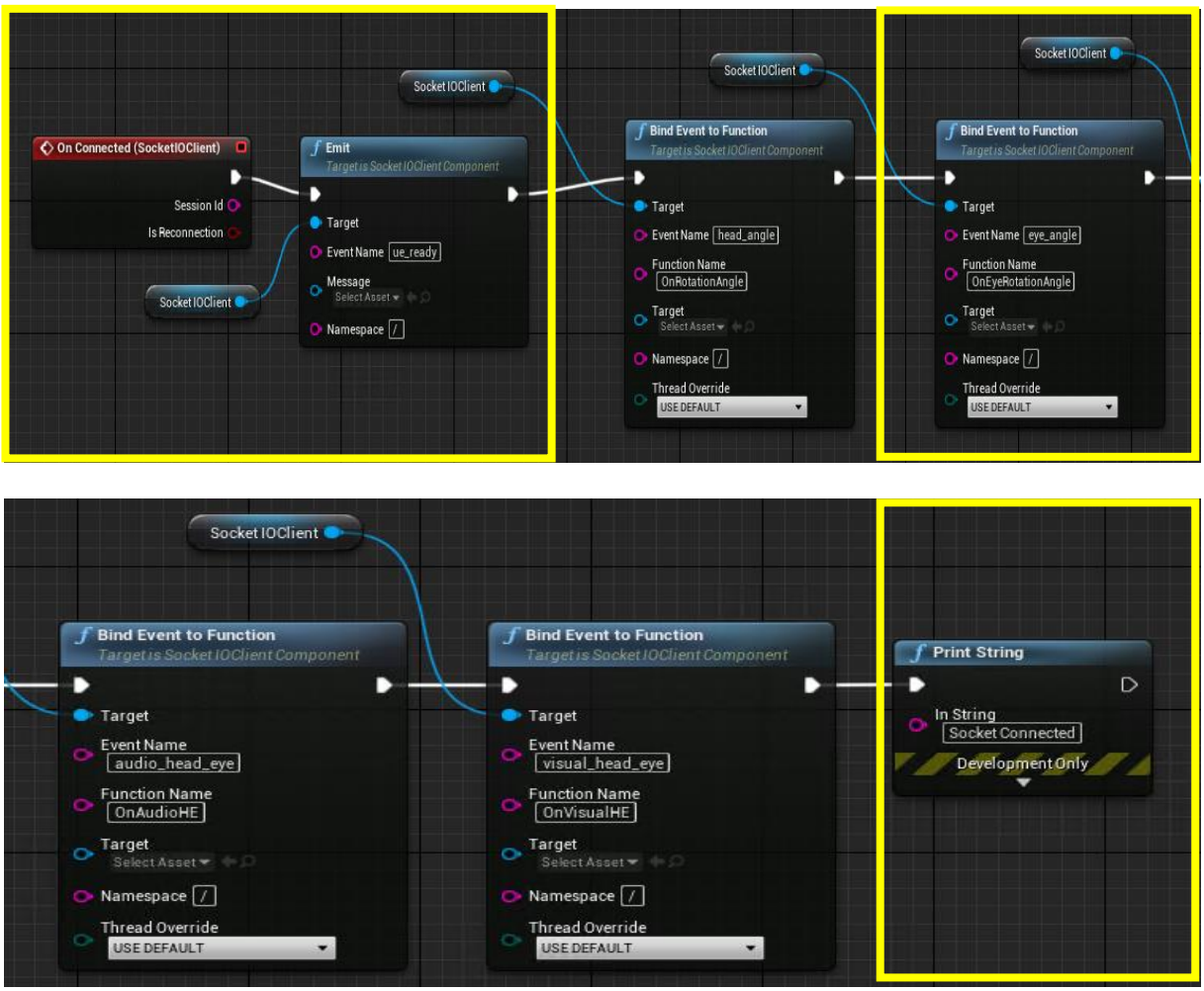
```
cv2.imshow('video', frame)
```

4.2 Unreal Engine program za pokret očiju na temelju vizualne akvizicije

Slanjem vrijednosti dobivenog kuta položaja lica ostvaruje se pokret očiju virtualnog agenta u programu Unreal Engine.

4.2.1 Blueprint za primanje podataka preko socket protokola

Unreal Engine također posjeduje socketIO biblioteku za slanje/primanje podataka. Izrađen je Blueprint sa socketio funkcijama. Unreal Engine predstavlja drugi kraj, odnosno klijent socket komunikacije.

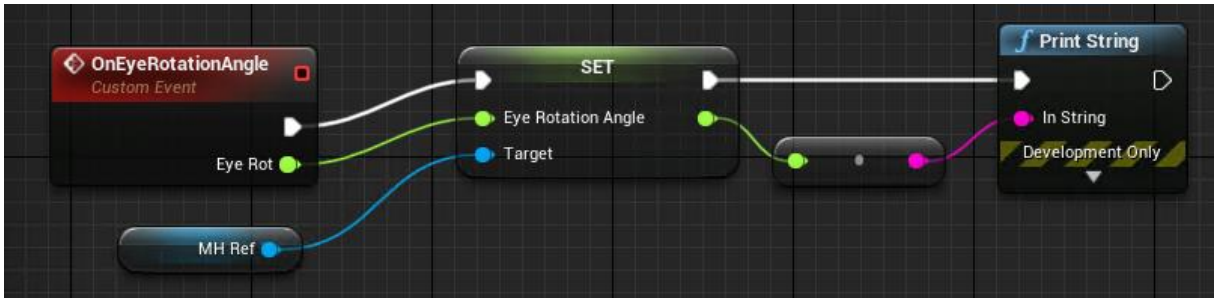


Slika 21. SocketIO Blueprint za primanje podataka za pokret očiju

Blueprint započinje eventom koji se pokreće spajanjem servera na klijent. Zatim se emitira događaj naziva *ue_ready* definiran u Python kodu servera te se povezuje događaj (eng. event) *eye_angle* koji prima vrijednost kuta položaja lica sa funkcijom *OnEyeRotationAngle*. Nakon uspješnog spajanja ispisuje se poruka „ Socket Connected“. SocketIOClient komponenta je SocketIO biblioteke. Nakon što ju dodamo potrebno je upisati IP adresu te odabrati slobodni port.

Slika 22. Postavljanje IP adrese i porta SocketIO veze

4.2.2 Povezivanje podataka sa servera sa MetaHuman-om

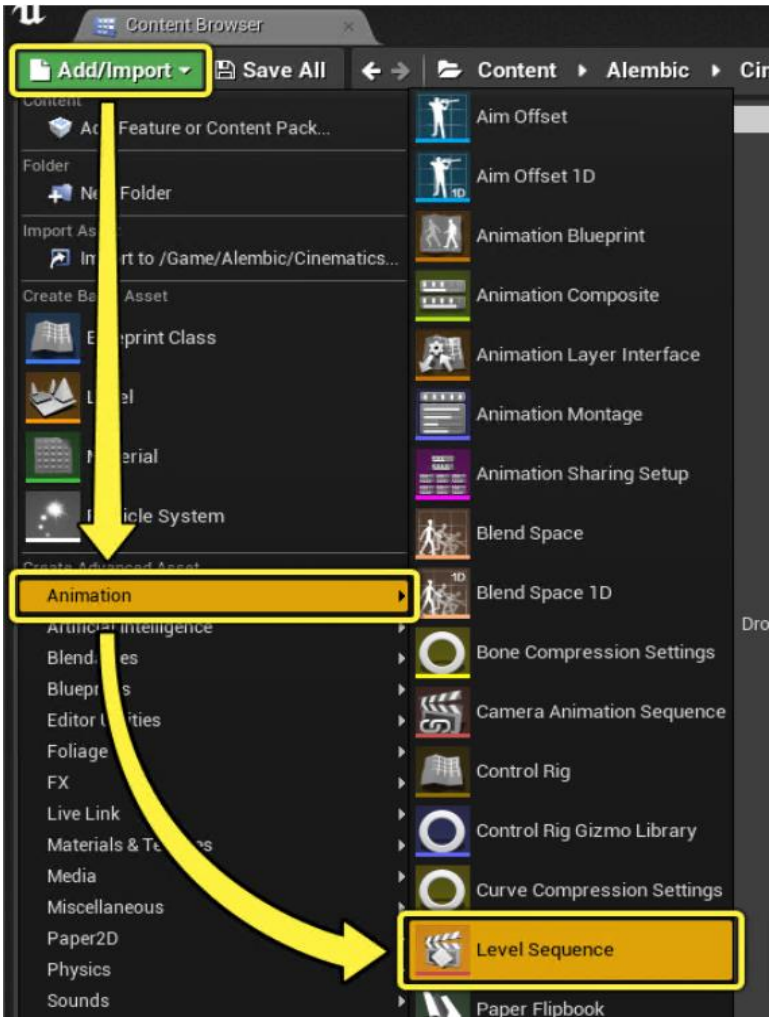


Slika 23. Povezivanje ulazne vrijednosti kuta vizualnog inputa sa instancom Actora

Na prethodnoj slici prikazano je povezivanje događaja naziva *OnEyeRotationAngle* koja posjeduje kao input poslanu vrijednost iznosa kuta *EyeRot* (koja odgovara vrijednosti poslane varijable *Angle1* iz Python skripte). Zatim se taj input povezuje sa varijablom iz Blueprinta Actora. *Print String* je funkcija za ispisivanje vrijednosti poslanog kuta.

4.2.3 Kreiranje animacije treptaja očiju

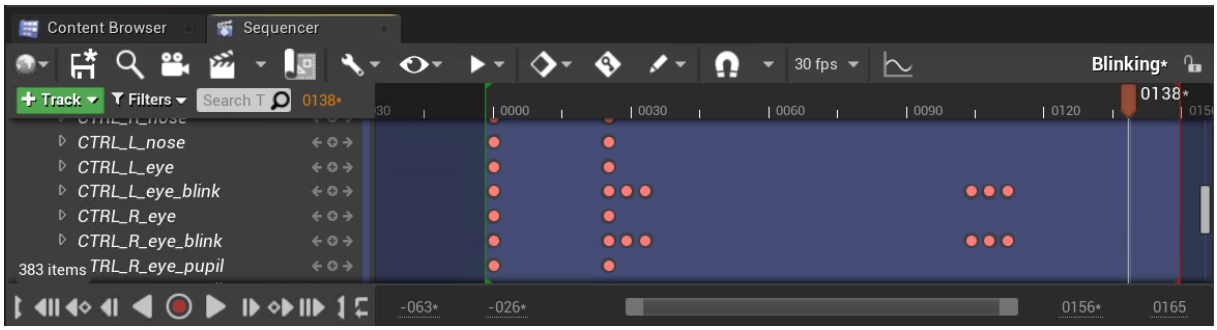
Automatske animacije pokreta mogu se praviti korištenjem Sequencer-a. Sequencer omogućuje korisniku mogućnost kreiranja animacija pokreta korištenjem specijaliziranog sequence uređivača.



Slika 24. Dodavanje novog sequencer-a

Nakon kreiranja novog sequencera potrebno je dodati instancu koju želimo animirati. U ovom slučaju, to je kreirani MetaHuman. Osnovni način na koji se dodaju pokreti u sequenceru naziva se keyframing. Keyframing se odvija na način da se sprema svaka promjena pokreta određenog

dijela tijela te se pokretanjem animacije interpolira prelazak iz jednog položaja u sljedeći. Animacija se pokreće u beskonačnoj petlji.

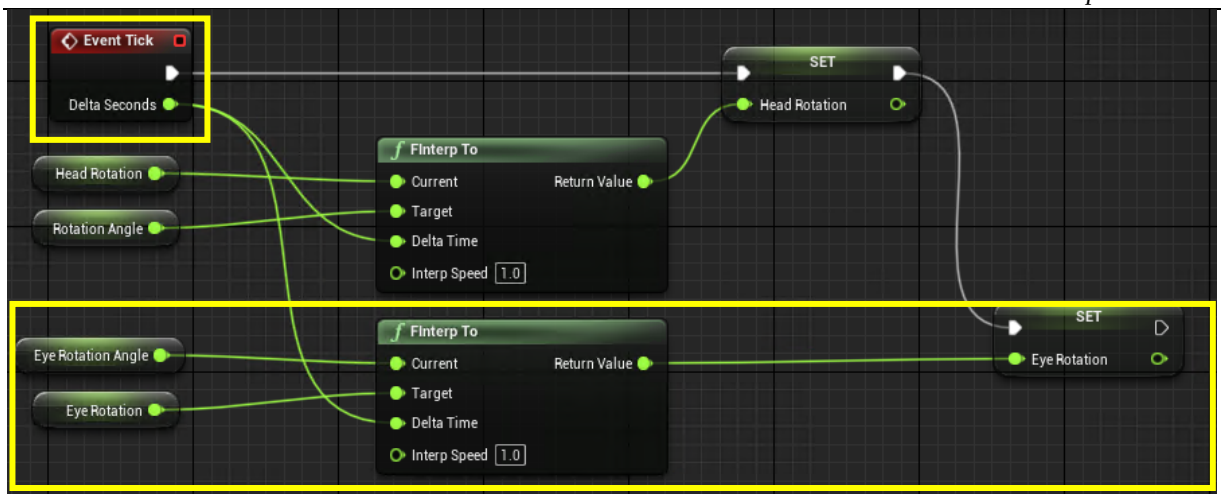


Slika 25. Kreiranje animacije treptanja u sequenceru

Animacija se kreira promjenom vrijednosti varijabli *CTRL_L_eye_blink* i *CTRL_R_eye blink*. Mijenjanje njihovih vrijednosti odvija se na Face_Control_Rig-u koji pokazuje sve pomične elemente lica.

4.2.4 Blueprint Actora (MetaHumana)

U Event Graph Blueprint dodan je Event i skup funkcija koji služi za vremensku interpolaciju vrijednosti varijable za rotaciju očiju *EyeRotation* iz postojeće u novu vrijednost. Interpolacija se vrši kako bismo izbjegli skokoviti prijelaz iz jednog položaja u drugi. Funkcija *Finterp To* ima 4 ulaza, *Current* (trenutna vrijednost), *Target* (naredna vrijednost), *Delta Time* (uzima brojčanu vrijednost vremenskog koraka za interpolaciju), *Interp Speed* (brzina interpolacije, skokovit prijelaz za vrijednost 0, veći broj znači sporija interpolacija). *Event Tick* funkcija pozvana je sa svakim frame-om. Na slici ispod, žutom su bojom označeni dijelovi koji se odnose na animaciju očiju.



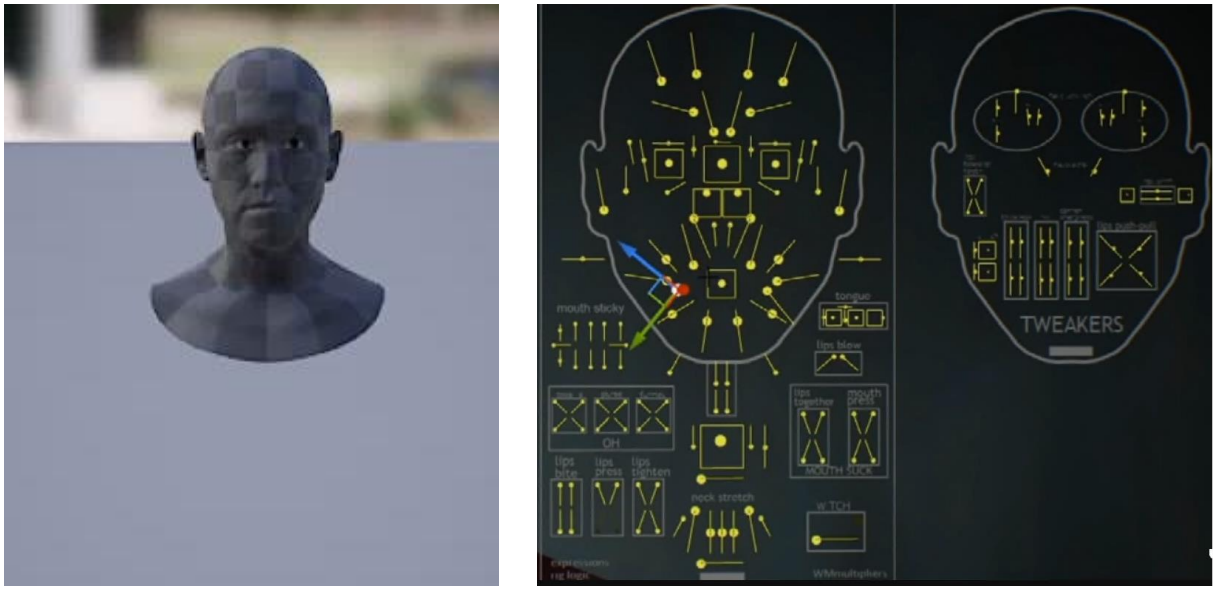
Slika 26. Event Graph Blueprinta Actora za pokretanje očiju

4.2.5 Blueprint animacije lica

Blueprint animacije lica sadrži logiku potrebnu da se ostvari animacija lica na osnovu kreirane animacijske sekvence i položaja lica na kameri. Sastoji se od *EventGraph*-a zaduženog za upravljanje događajima (eng. eventima) i *AnimgGraph*-a zasluženog za ostvarivanje animacije.

4.2.5.1 Kostur lica

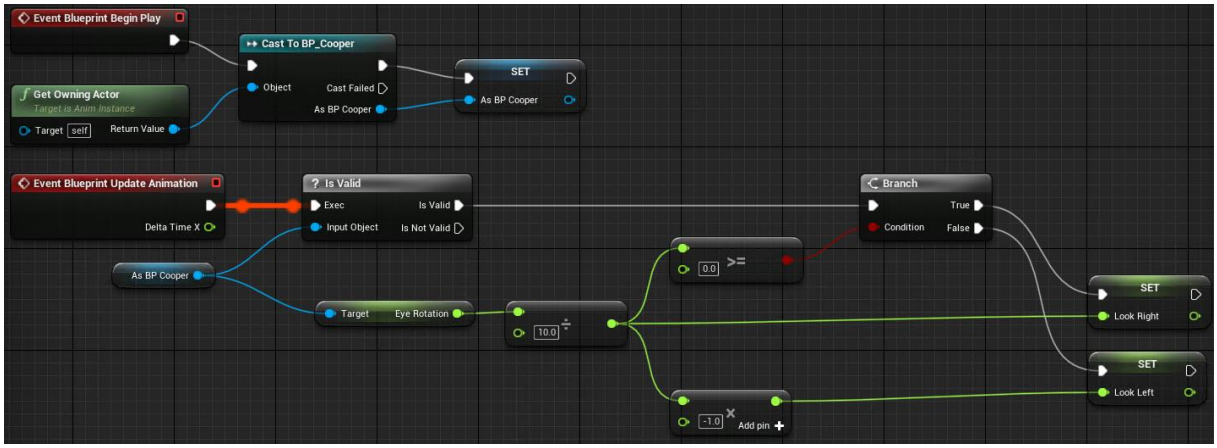
Unreal Engine animacije kreiraju se na kosturu (eng. skeleton). Kostur MetaHumana podijeljen je na kostur glave i kostur tijela. Za kreiranje animacije lica potrebno je odabrati kostur glave. Mogući pokreti na licu MetaHumana vidljivi su na tzv. *Face Control Rigu*-u.



Slika 27. Kostur lica (lijevo) i Face Control rig (desno)

4.2.5.2 EventGraph animacijskog Blueprinta lica

Event Graph sadrži događaje i funkcije koje se pozivaju kada se dogodi određeni događaj (eng. event). U ovom slučaju dva *Eventa* pokreću funkcije, *Event Blueprint Begin Play*, koji se pokreće pokretanjem igre te *Event Blueprint Update Animation*, koji se pokreće svaki put kad se ažurira animacija. Cilj Blueprinta je očitati vrijednost varijable *EyeRotation* iz Blueprinta Actora te njenu vrijednosti prepisati varijablama *LookLeft* i *LookRight*.

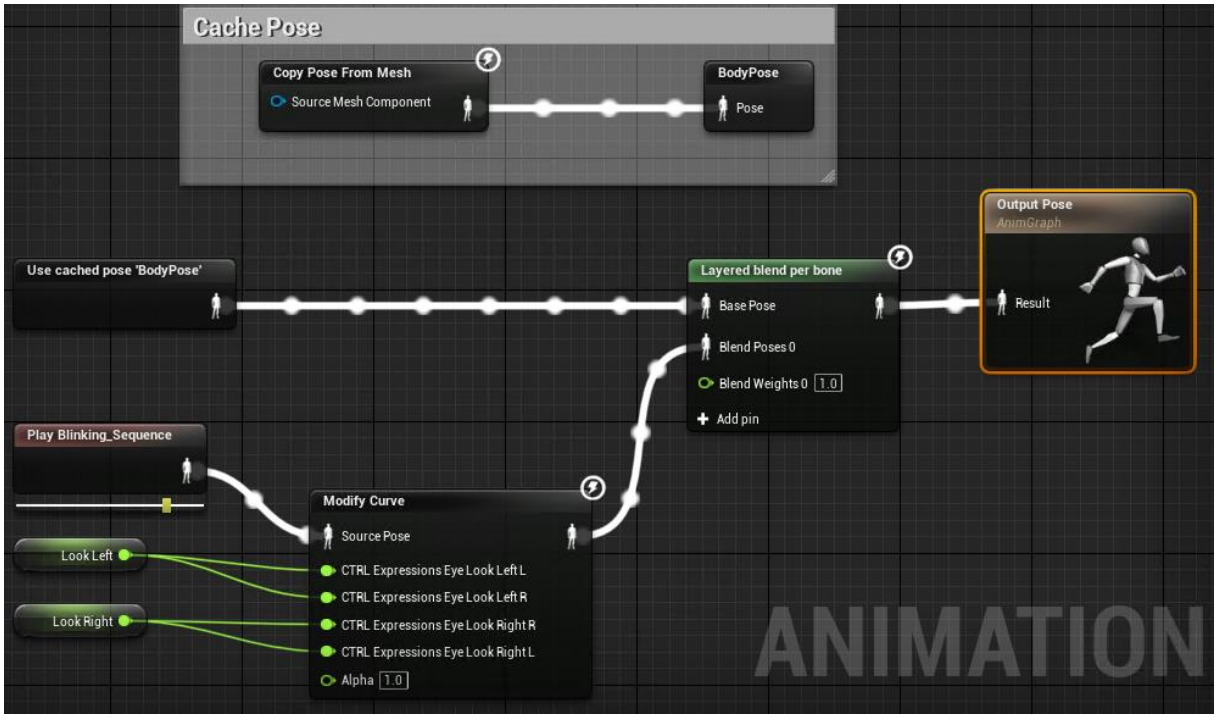


Slika 28. EventGraph animacijskog Blueprinta lica

Na slici 27 prikazan je *EventGraph* koji očitava brojčanu vrijednost varijable *EyeRotation* pri svakom ažuriranju animacije. Budući da pokreti očiju primaju vrijednost od 0-1, potrebno je tu vrijednost skalirati, odnosno umanjiti 10 puta. Ukoliko je lice u sredini kamere, vrijednost varijable *EyeRotation* iznosit će 0, pomakom u lijevu i desnu stranu ta se vrijednost povećava, odnosno smanjuje, tj. postaje neki negativan broj. Ukoliko je varijabla *EyeRotation* pozitivna, prosljeđuje se varijabli *LookRight*, koja će utjecati na pogled udesno, u suprotnom se pretvara u pozitivan broj te prosljeđuje varijabli *LookLeft* koja utječe na ostvarivanje pogleda ulijevo.

4.2.5.3 AnimGraph animacijskog Blueprinta lica

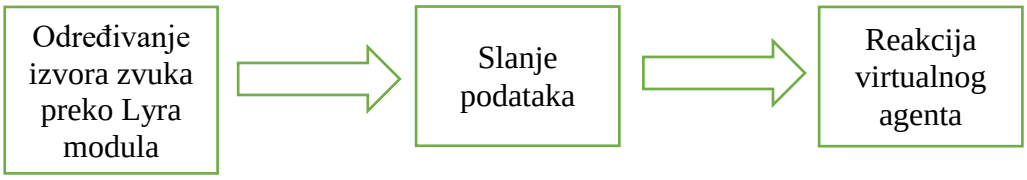
AnimGraph sadrži kosti odnosno kontrole kojima treba upravljati. Učitava se vrijednost varijabli *LookLeft* i *LookRight* te se na osnovu promjene iznosa te varijable mijenja i iznos rotacije očiju. Na *Source Pose* veže se snimljena sekvenca treptaja očiju kako bi se pokrenula na virtualnom agentu. Oči se upravljaju pojedinačno sa zasebnim varijablama za pogled ulijevo i pogled udesno. Zatim se dobivena poza stapa (eng. blend) sa trenutnim stanjem položaja tijela i spaja na *Output Pose* koji izvozi rezultat dobivene animacije.



Slika 29. AnimGraph animacijskog Blueprinta lica

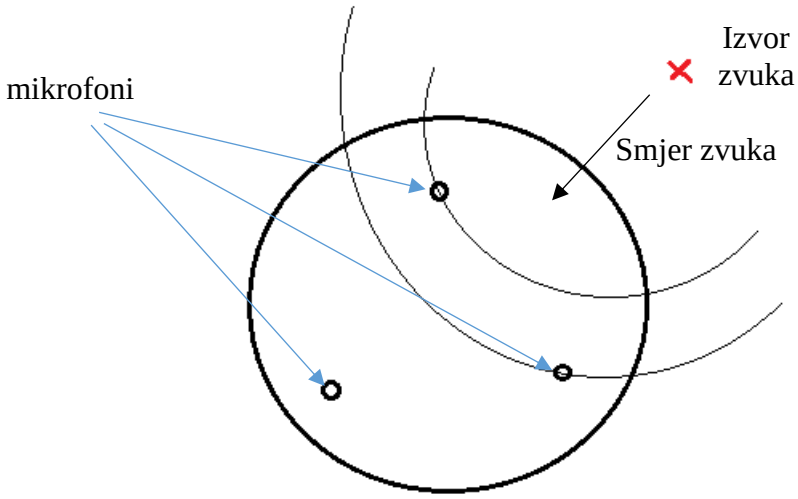
5 AUDIO AKVIZICIJA

Shema procesa zvučne akvizicije prikazana je na slici 29.



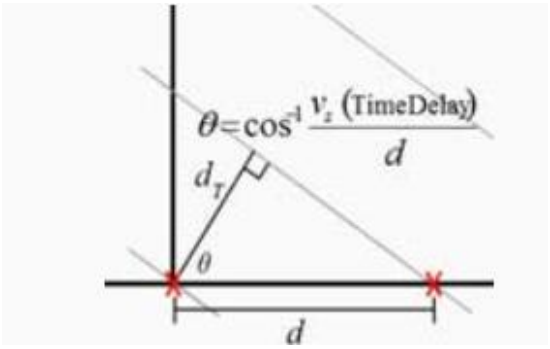
Slika 30. Proces zvučne akvizicije

Izračunavanje smjera zvuka Lyra modula temelji se na širenju zvučnog vala i vremenskoj razlici dolaska vala prema 3 kružno raspoređena mikrofona.



Slika 31. Širenje zvučnog vala i očitanje Lyra modula

Pri većoj udaljenosti zvučnog izvora, zvučni valovi se aproksimiraju ravnim linijama:



Slika 32. Aproksimacija kuta smjera dolaska zvuka[12]

Zvučni izvor smješten je pod kutom θ od linije koja povezuje dva senzora, udaljenih za duljinu d. Duljina d_T okomica je na ravninu koja presjeca prvi senzor i ravninu koja presjeca drugi senzor. [12] Ona se može izraziti kao:

$$d_T = dcos\Theta \tag{4}$$

Ova se duljina može smatrati i kao duljina koju zvuk prijeđe za τ sekundi, gdje τ predstavlja propagacijsko kašnjenje između dva senzora, a v_s je brzina zvuka:

$$d_T = v_s\tau \tag{5}$$

Izjednačavanjem desnih strana jednadžbi 4 i 5 dobije se jednadžba:

$$dcos\Theta = v_s\tau \tag{6}$$

Ako je n kašnjenje uzorkovanja, a f_s frekvencija uzorkovanja, τ je jednaka:

$$v_s\tau = \frac{n}{f_s} \tag{7}$$

Konačno dolazimo do formule za kut izvora zvuka kao funkcije vremena uzorkovanja:

$$cos\Theta = \frac{v_s n}{d f_s} \tag{8}$$

5.1 Python kod za zvučnu akviziciju

Uvoženje potrebnih biblioteka:

```
import eventlet
import socketio
import threading
import serial
import re
```

Biblioteka serial koristi se za otvaranje serijskog porta na kojem je spojen Lyra modul. Port na kojem je spojen Lyra modul može se vidjeti u *Device manageru* računala.

Biblioteka re služi za operacije regularnim izrazima, odnosno manipulaciju stringova korištenjem karakterističnih znakova za izdvajanje dijelova stringa od ostatka.

```
s = serial.Serial("COM4", 115200, timeout=1)
```

Metoda *serial.Serial()* kao ulazne parametre uzima broj porta, broj bitova po sekundi te vrijeme očitanja u sekundama.

Dio koda za stvaranje servera isti je kao i kod vizualne akvizicije.

```
sio = socketio.Server(cors_allowed_origins='*')
app = socketio.WSGIApp(sio)

@sio.event
async def connect(sid, environ):
    print(f'Someone connect with id: {sid} ')

@sio.event
def disconnect(sid):
    print(f'Someone disconnect with id: {sid} ')

@sio.on('ue_ready')
def another_event(sid, data):
    print(f'ue is connected with id {sid} and is ready to receive
data')
    x = threading.Thread(target=read_sound_angle)
    x.start()
```

Zatim se definira funkcija koja sadrži beskonačnu petlju za učitavanje podataka s Lyra modula. Učitani podaci se najprije pretvaraju u varijablu tipa string kako bi mogli biti manipulirani regex metodom, a zatim se šalje vrijednost kuta u iznosu ±180˚.

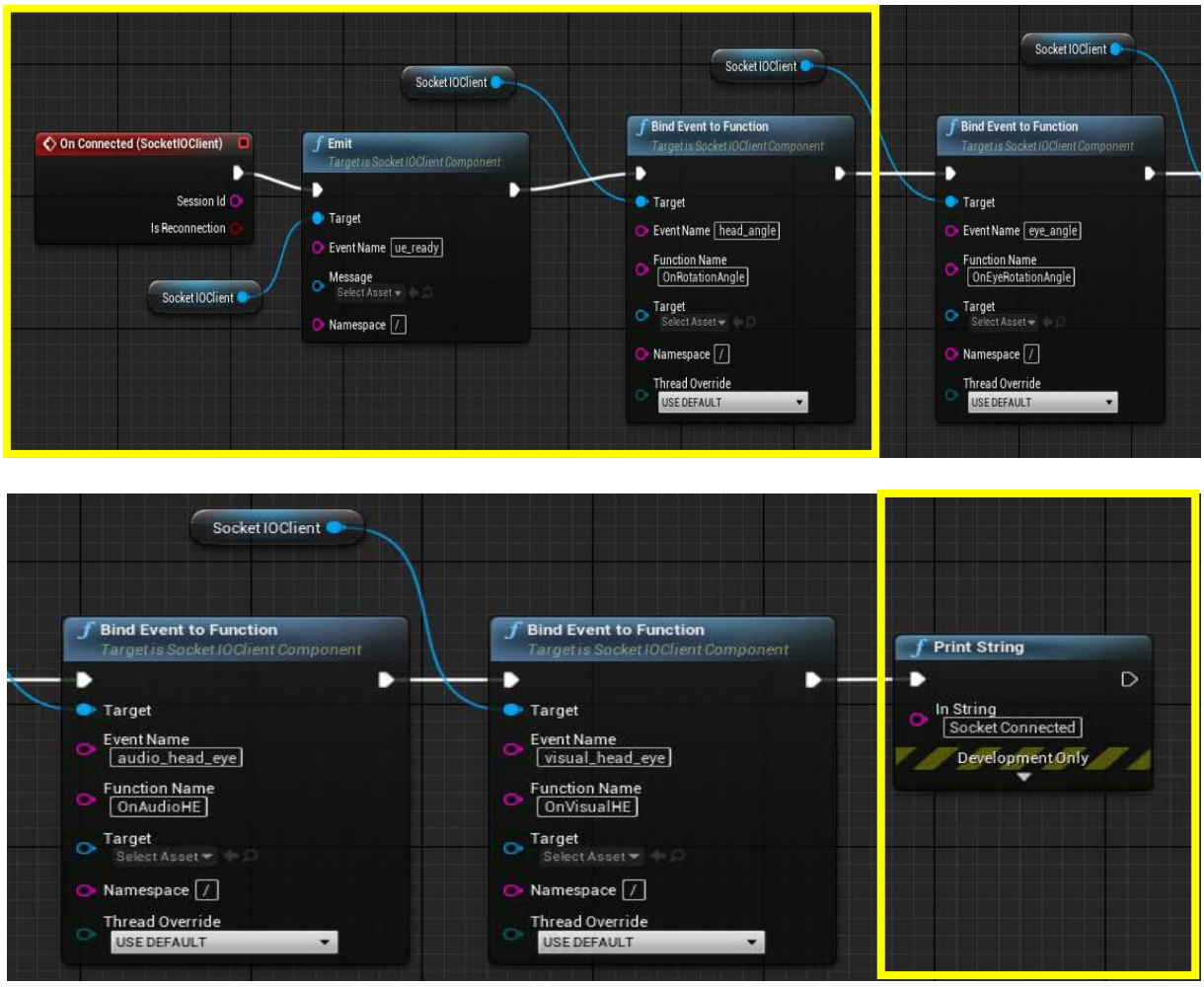
```
        if re.match(r'.*DOA: .*', line):
              rec = re.findall(r'(-?\d+\.\d+)', line)
              if len(rec) < 2:
                  return 0.0
```

```
        else:
            sio.emit('head_angle',180-float(rec[0]))
            print(float(rec[0])-180)
        sio.sleep()
```

5.2 Unreal Engine program za pokret očiju na temelju vizualne akvizicije

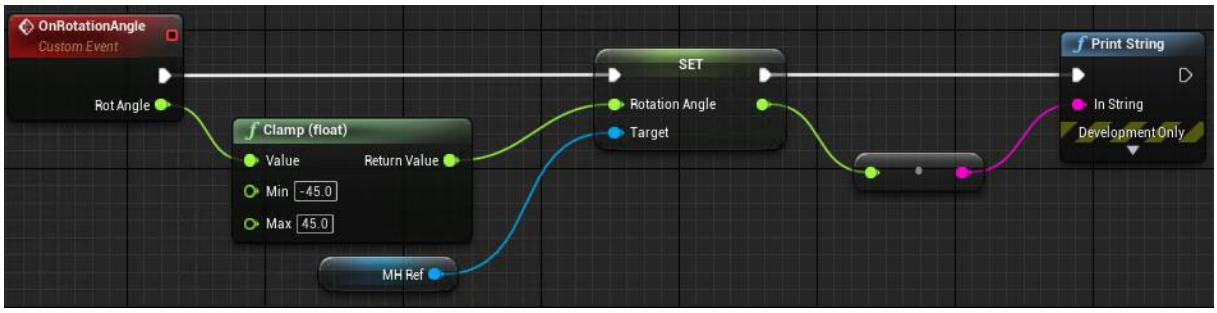
Slanjem vrijednosti dobivenog kuta smjera zvuka ostvaruje se okret glave virtualnog agenta u programu Unreal Engine u smjeru izvora zvuka.

5.2.1 Blueprint za socketIO komunikaciju



Slika 33. Blueprint za primanje podataka za rotaciju glave

Kao i kod vizualne akvizicije, blueprint započinje eventom koji se pokreće spajanjem servera na klijent. Zatim se emitira događaj naziva *ue_ready* definiran u Python kodu servera te se povezuje događaj (eng. event) *head_angle* koji prima vrijednost kuta smjera zvuka sa funkcijom *OnRotationAngle*.

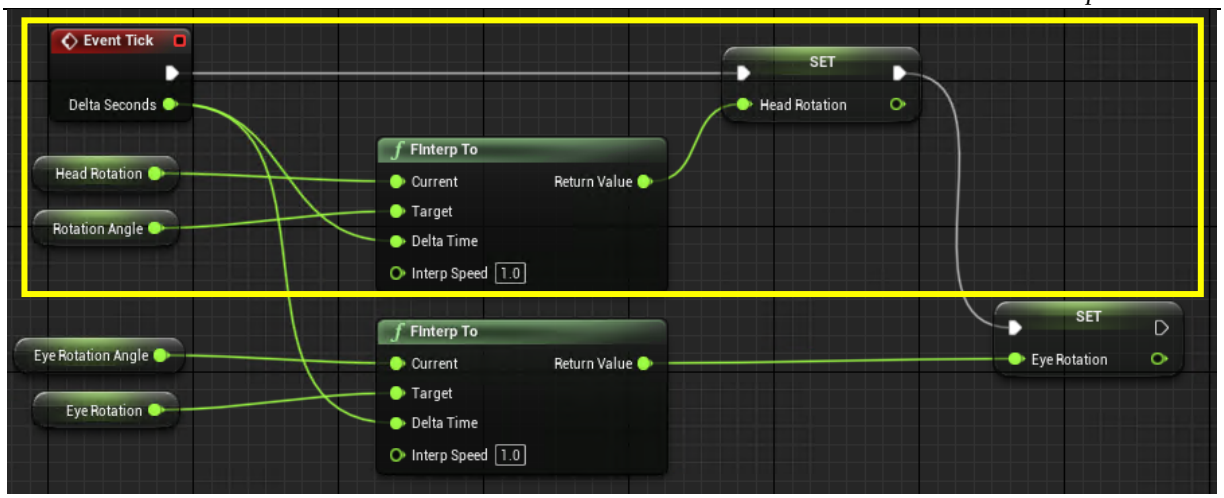


Slika 34. Povezivanje ulazne vrijednosti kuta audio inputa sa instancom Actora

Na slici 34 prikazano je povezivanje događaja naziva *OnRotationAngle* koja posjeduje kao input poslanu vrijednost iznosa kuta *RotAngle* (koja odgovara vrijednosti poslanih poadataka kuta iz Python skripte). Budući da kut poslan s Lyra modula može iznositi ±180˚, potrebno ga je ograničiti, a kao granice su postavljene vrijednosti ±45˚. U slučaju vrijednosti većih ili manjih od graničnih, uzimaju se granične vrijednosti. Zatim se taj output povezuje sa varijablom *RotationAngle* iz Blueprinta Actora. *Print String* je funkcija za ispisivanje vrijednosti poslanog kuta.

5.2.2 Blueprint Actora

U Event Graph Blueprint dodan je Event i skup funkcija koji služi za vremensku interpolaciju vrijednosti varijable za rotaciju glave *Head Rotation* iz postojeće u novu vrijednost. Na slici ispod, žutom su bojom označeni dijelovi koji se odnose na animaciju očiju.



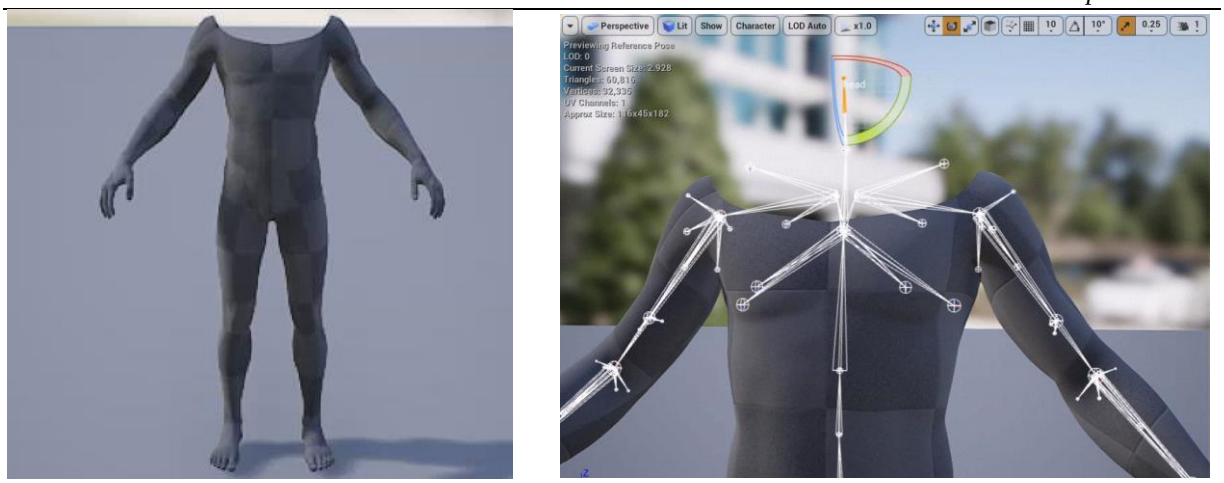
Slika 35. Event Graph Blueprinta Actora za promjenu zakreta glave

5.2.3 Animacijski Blueprint zakreta glave

Blueprint animacije zakreta glave sadrži logiku potrebnu da se ostvari animacija rotacije na osnovu smjera zvuka očitanog na Lyra modulu. Također se sastoji od *EventGraph*-a zaduženog za upravljanje događajima (eng. eventima) i *AnimgGraph*-a zasluženog za ostvarivanje animacije.

5.2.3.1 Kostur tijela

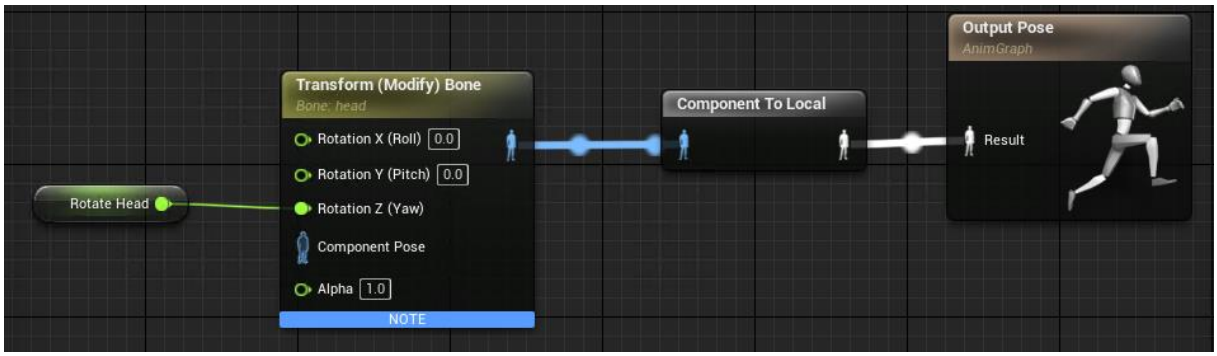
Najprije je potrebno stvoriti novi animacijski Blueprint koji za ciljni kostur (eng. Target Skeleton) ima odabran kostur tijela. Kako bi se ostvario pokret određenog dijela tijela, najprije je potrebno odrediti koje kosti su odgovorne za pokret dijela. U ovom slučaju, potrebno je omogućiti okretanje glave lijevo i desno. Provjerom se ustanovi da je za to potrebno rotirati kost naziva *head* oko osi Z.



Slika 36. Kostur tijela (lijevo) i kost glave (desno)

5.2.3.2 AnimGraph animacijskog Blueprinta tijela

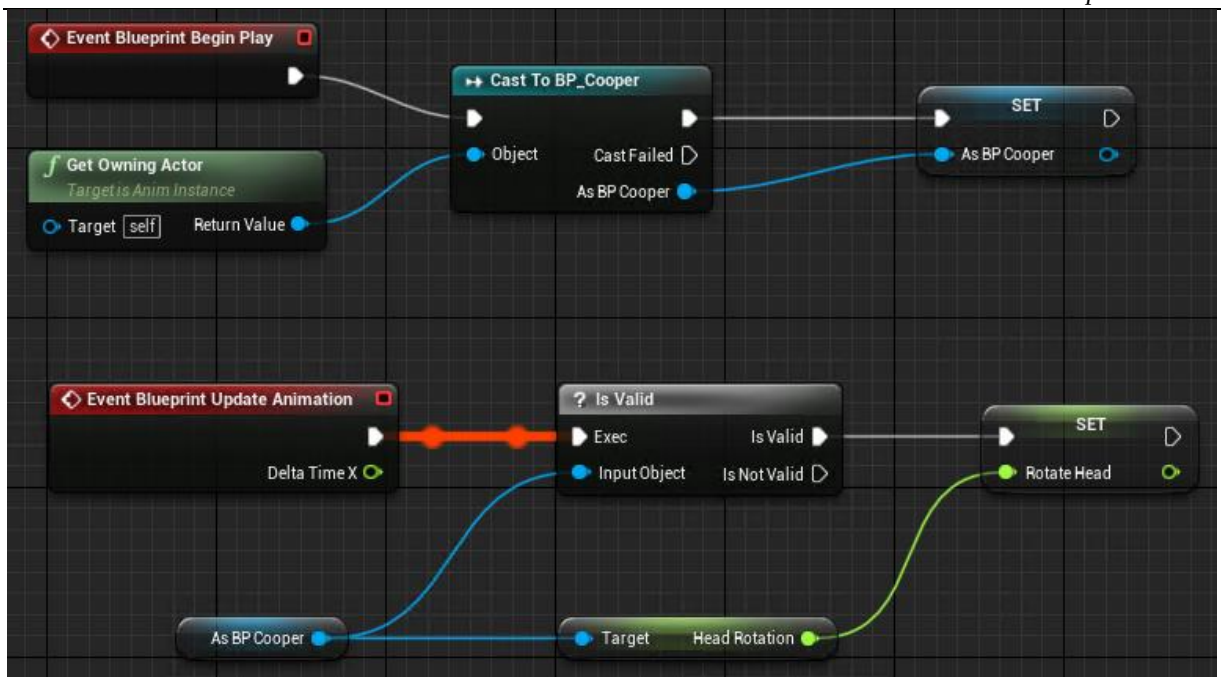
AnimGraph sadrži kosti koje treba kontolirati kako bi se ostvario pokret. Učitava se vrijednost varijable *RotateHead* te se na osnovu promjene iznosa te varijable mijenja i iznos rotacije kosti *head* oko osi Z.



Slika 37. AnimGraph animacijskog Blueprinta tijela

5.2.3.3 Event Graph animacijskog Blueprinta tijela

Cilj Blueprinta je očitati vrijednost varijable *HeadRotation* iz Blueprinta Actora te njenu vrijednosti prepisati varijabli *RotateHead* koja je prikazana u prethodnom poglavlju.



Slika 38. Event Graph animacijskog Blueprinta tijela

6 OSTVARIVANJE ISTOVREMENE ANIMACIJE GLAVE I OČIJU

Do sada smo napravili da se na vizualni input ostvaruje pokret očiju virtualnog agenta, dok se na zvučni podražaj ostvaruje rotacije glave u smjeru izvora zvuka. Malom doradom koda moguće je ostvariti istovremene pokrete glave i očiju na vizijski, odnosno na zvučni podražaj.

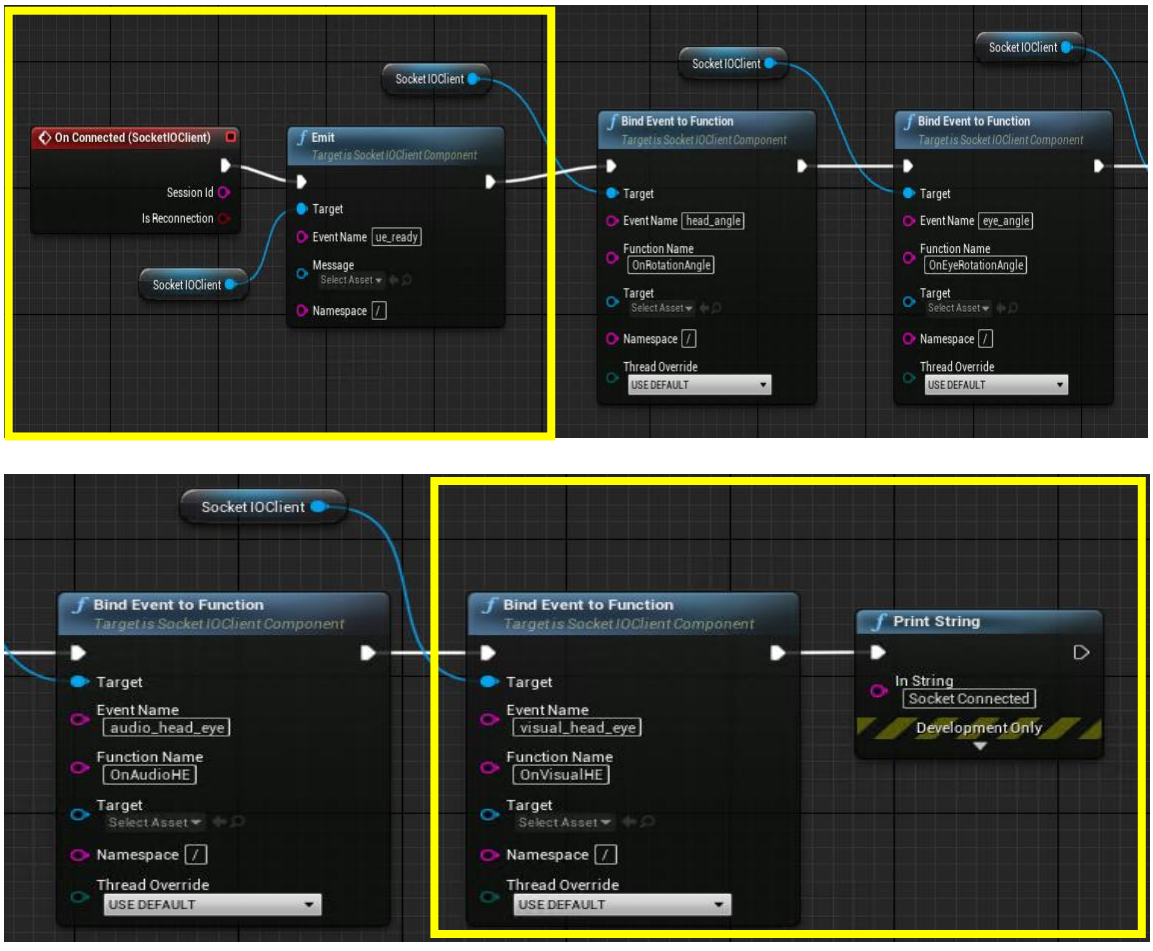
6.1 Rotacija glave i očiju na vizualni podražaj

6.1.1 Python kod

Za istovremenu rotaciju glave i očiju na vizualni podražaj stvorena je nova Python skripta. Sadržaj skripte isti je onom iz poglavlja 4.1, uz promjenu naziva eventa koji će se emitirati u Unreal Engine projekt.

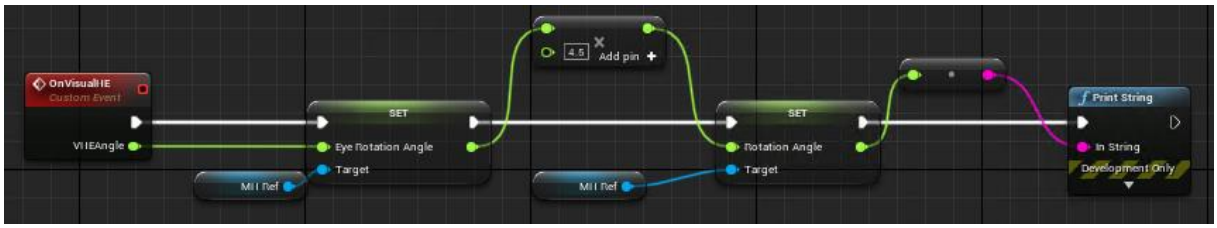
```
sio.emit('visual_head_eye', angle1)
```

6.1.2 Unreal Engine Blueprint-ovi



Slika 39. SocketIO Blueprint za primanje podataka za pokret očiju i glave na vizualni input

Na slici 39. žutom je bojom istaknut dio socketIO Blueprinta za primanje vizualnog inputa za rotaciju glave i očiju.



Slika 40. Povezivanje ulazne vrijednosti kuta vizualnog inputa sa instancama Actora

Na slici 40. prikazano je povezivanje događaja naziva OnVisualHE koja posjeduje kao input poslanu vrijednost iznosa kuta VHEAngle (koja odgovara vrijednosti poslane varijable Angle1 iz Python skripte). Taj se input povezuje sa varijablama EyeRotationAngle i RotationAngle iz Blueprinta Actora.

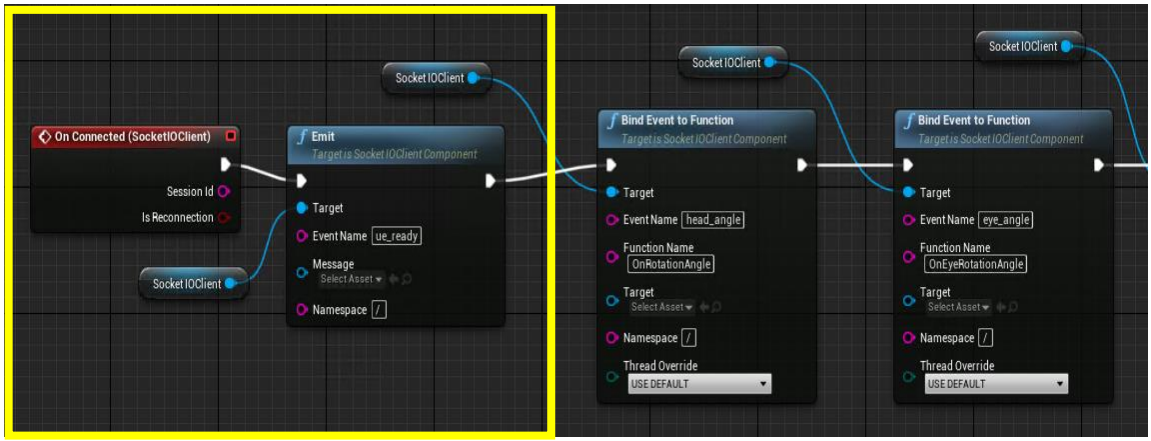
6.2 Rotacija glave i očiju na audio podražaj

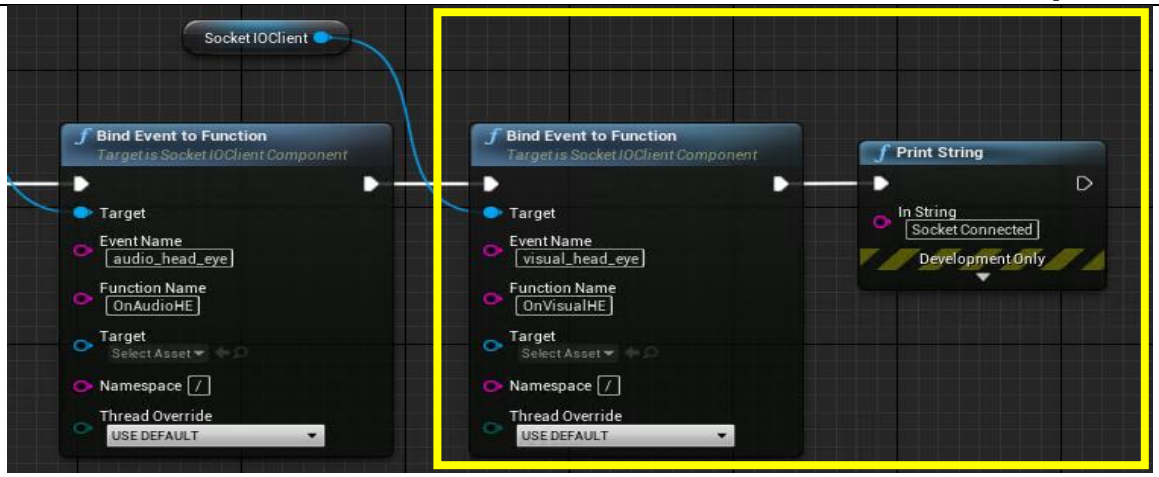
6.2.1 Python kod

Kao i kod poglavlja 6.1, stvorena je nova Python skripta. Sadržaj skripte isti je onom iz poglavlja 5.1, uz promjenu naziva eventa koji će se emitirati u Unreal Engine projekt.

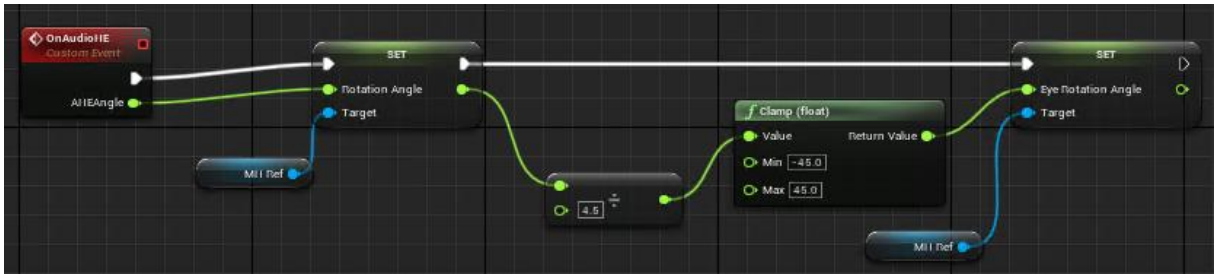
```
sio.emit('audio_head_eye', angle1)
```

6.2.2 Unreal Engine Blueprint-ovi





Slika 41. SocketIO Blueprint za primanje podataka za pokret očiju i glave na vizualni input



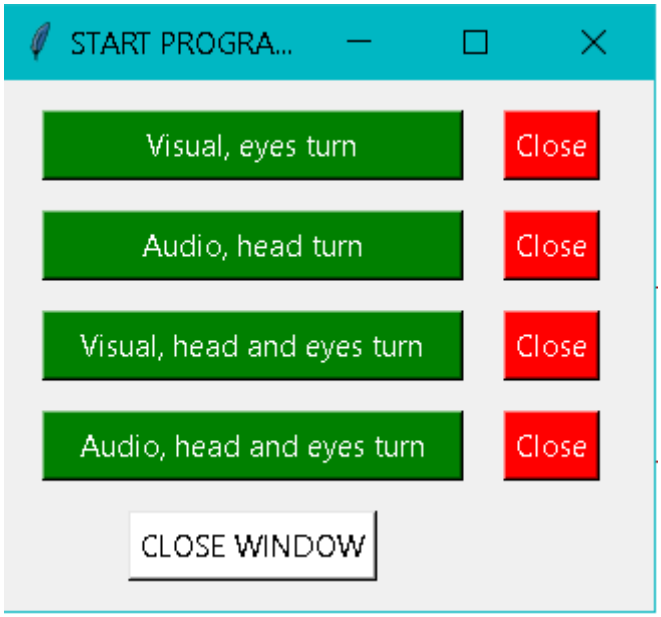
Slika 42. Povezivanje ulazne vrijednosti kuta audio inputa sa instancama Actora

Na slici 42. prikazano je povezivanje događaja naziva OnAudioHE koja posjeduje kao input poslanu vrijednost iznosa kuta AHEAngle (koja odgovara vrijednosti poslanog iznosa kuta zvuka iz Python skripte). Taj se input povezuje sa varijablama EyeRotationAngle i RotationAngle iz Blueprinta Actora.

7 POKRETANJE PROGRAMA I PRIKAZ REZULTATA

7.1 Python GUI

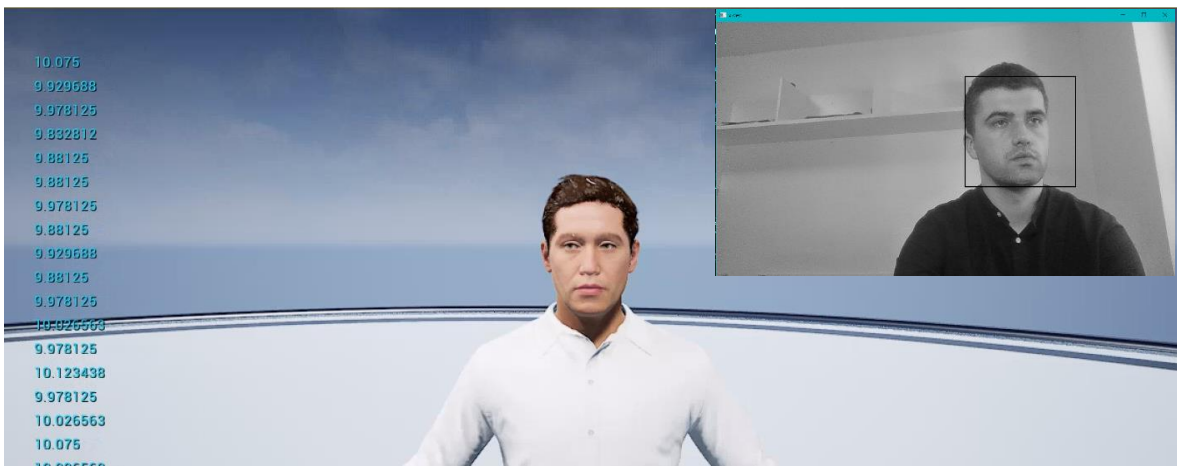
Za pokretanje Python skripti napravljen je Python GUI (eng. graphic user interface) prikazan na slici 43. Također je u Unreal Engine-u potrebno pokrenuti glavni program pritiskom na tipku Play.



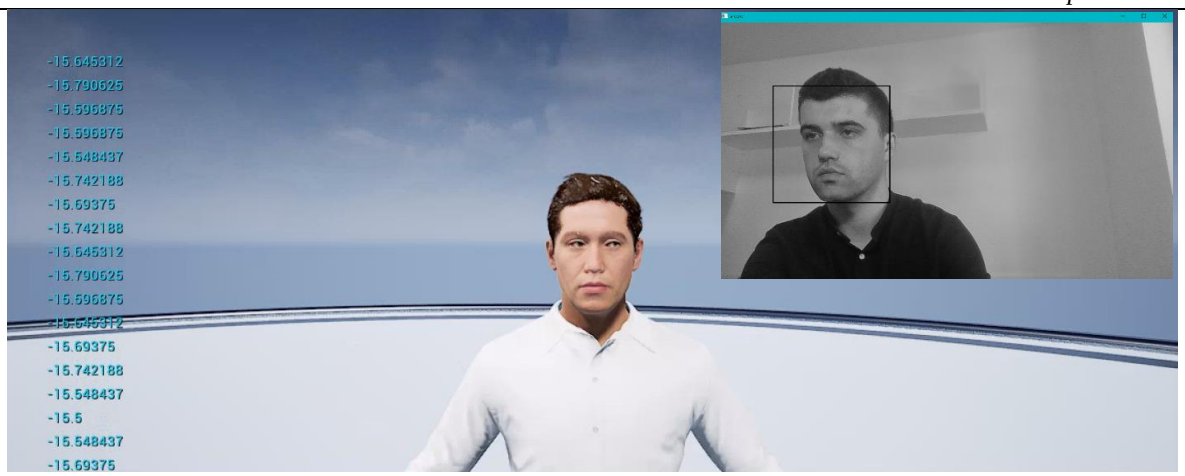
Slika 43. Python GUI

7.2 Prikaz rezultata

Na slikama 44 i 45 prikazani su rezultati detekcije položaja lica pri čemu virtualni agent očima prati horizontalni pomak lica. Kamera pokazuje zrcalnu sliku položaja osobe u prostoru, a virtualni agent prati stvarni položaj, stoga je prikaz na slikama suprotan od stvarnog položaja.

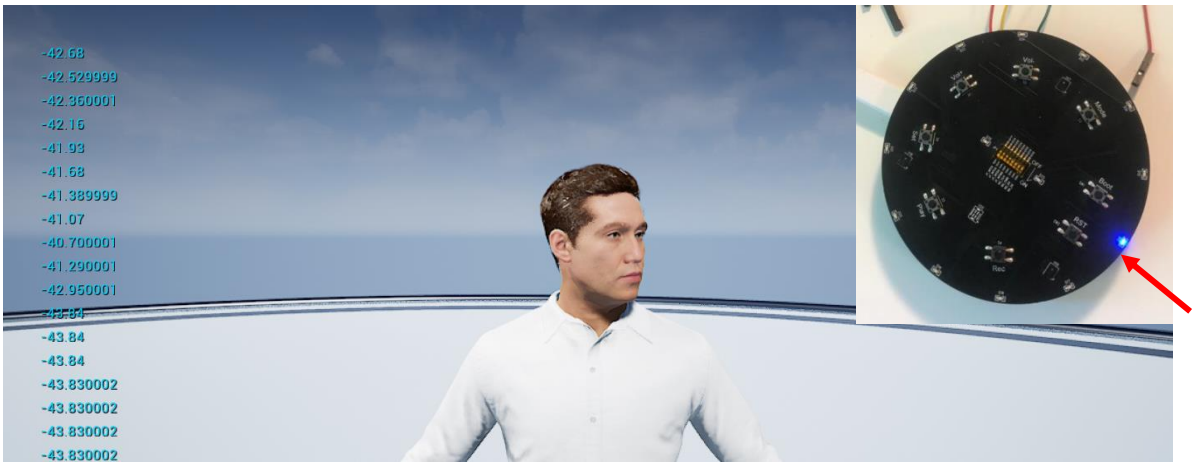


Slika 44. Praćenje položaja lica očima (lijevo)



Slika 45. Praćenje položaja lica očima (desno)

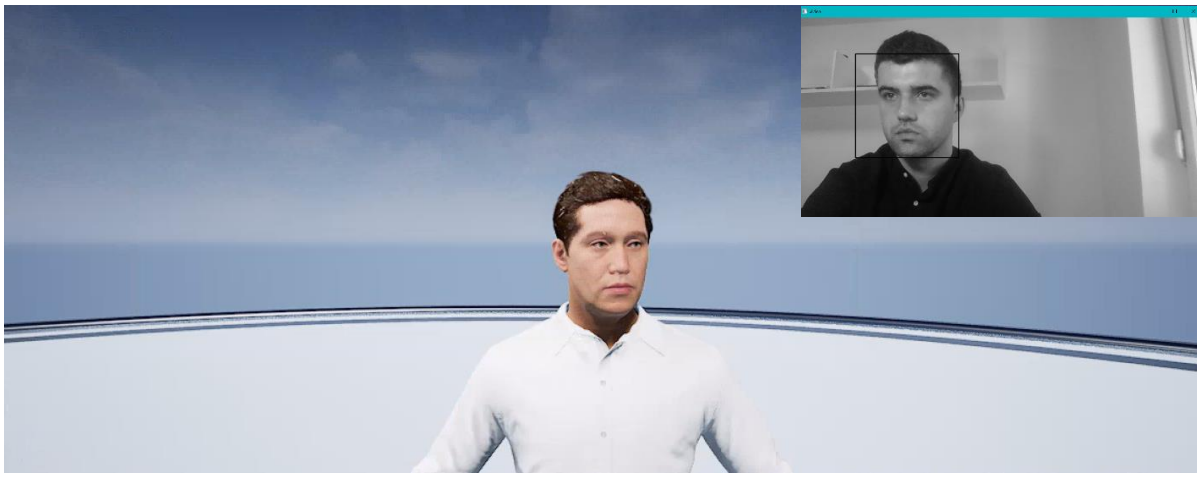
Na slikama 46 i 47 prikazani su rezultati okreta virtualnog agenta prema smjeru zvučnog izvora. Crvene strelice na slikama pokazuju smjer izvora zvuka.



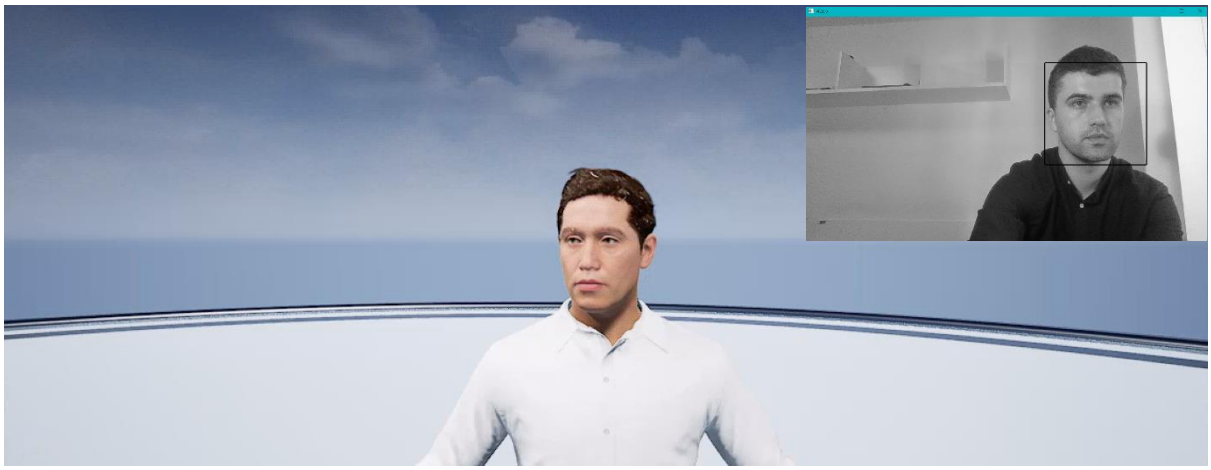
Slika 46. Okret glave prema izvoru zvuka (desno)

Slika 47. Okret glave prema izvoru zvuka (lijevo)

Na slikama 48 i 49 prikazani su rezultati ostvarivanja pokreta očiju i zakreta glave na vizualni input, odnosno promjenu položaja lica.

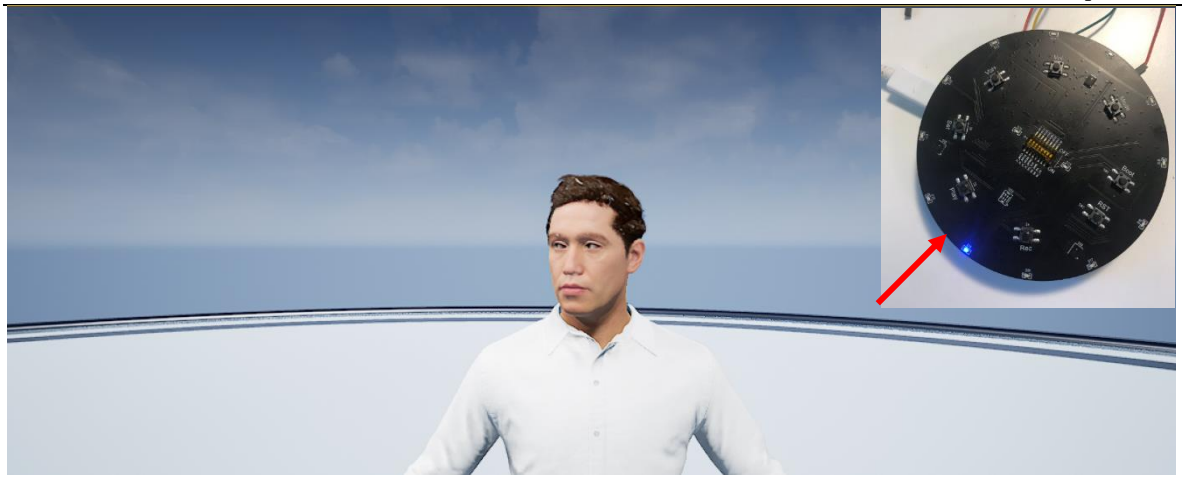


Slika 48. Praćenje položaja lica očima i glavom (lijevo)



Slika 49. Praćenje položaja lica očima i glavom (desno)

Na slikama 50 i 51 prikazani su rezultati praćenja izvora zvuka istodobnim pokretima očiju i glave. Crvenim su strelicama označeni smjerovi izvora zvuka.



Slika 50. Praćenje smjera izvora zvuka očima i glavom (lijevo)

Slika 51. Praćenje smjera izvora zvuka očima i glavom (desno)

Rezultati pokazuju uspješno implementiranje i dobru točnost praćenja tehnologije prepoznavanja lica sa zanemarivo malim vremenskim kašnjenjem odziva i u slučaju praćenja samo pokretima očiju te u slučaju praćenja pokretima očiju i glave. Također je uspješno implementiran i sustav prepoznavanja smjera zvuka te reakcija virtualnog agenta na njegovu promjenu. Za implementaciju sustava prepoznavanja smjera zvuka potrebno je zadovoljiti uvjet da se šumovi u prostoriji smanje na minimum kako bi očitanje bilo moguće.

8 ZAKLJUČAK

U ovom radu potrebno je bilo implementirati sustav prepoznavanja lica i određivanja njegova položaja u području kamere te sustav određivanja smjera izvora zvuka. Iste je bilo potrebno spojiti sa virtualnim agentom kako bi se ostvarila njegova reakcija. Za izradu ovog rada bilo je potrebno upoznati se sa novim programskim sučeljem, Unreal Engine, koji pruža velike mogućnosti u stvaranju najsuvremenijih sustava animacije i videoigara. U ovom diplomskom radu, kao i kod virtualnog agenta PLEA, mogućnosti ovog programa uporabljene su u do sad, na neki način, jedinstvenoj svrsi, što je zahtijevalo i dodatan trud u upoznavanju s radom programa, pa i samom rješavanju zadatka. Vizualno programiranje korišteno u programu Unreal Engine-a zanimljivo je u tome što omogućuje vidljivi tijek izvođenja programa. Također sam se imao priliku dublje upoznati s programskim jezikom Python i problemom prepoznavanja lica.

Opisani sustav moguće je i dalje razvijati, ponajprije korištenjem konvolucijskih neuronskih mreža u prepoznavanju lica, filtriranjem buke pri korištenju detekcija zvuka kako bi se sustav mogao koristiti i u uvjetima buke u prostoru.

LITERATURA

[1] https://en.wikipedia.org/wiki/Unreal_Engine

[2] Koren L., Stipančić T., Ričko A., Orsag L.: Person Localization Model Based on a Fusion of Acoustic and Visual Inputs, 2022.

[3] ESP32-Lyra TD-MSC User Guide, Version 2.1, 2018

[4] https://www.techtarget.com/searchapparchitecture/definition/object-oriented-programming-OOP

[5] https://www.w3schools.com/python/python_intro.asp

[6] R. Buyya, S. Selvi, X. Chu: Object Oriented Programming with Java: Essentials and Applications, McGraw Hill, New Delhi, India, 2009.

[7] https://opencv.org/about/

[8] https://towardsdatascience.com/face-detection-with-haar-cascade-727f68dafd08

[9] Paul Viola and Michael J. Jones : Robust real-time face detection. International Journal of Computer Vision, 2004.

[10] https://python-socketio.readthedocs.io/en/latest/server.html

[11] https://docs.unrealengine.com/4.27/en-US/AnimatingObjects/Sequencer/Overview/

[12] J. York: Acoustic Source Localization, Washington University, 2008

[13] https://oldpcgaming.net/the-wheel-of-time-review/

[14] http://s01.riotpixels.net/data/7c/96/7c96ae83-53da-416e-b3ae-9120867ced99.jpg/screenshot.unreal-tournament-2003.800x600.2002-05-17.25.jpg

[15] https://m.media-amazon.com/images/M/MV5BZjc4NmFlYTItODBiYS00ZDBiLTllMTktY2QyZTZjMTgxZmZhXkEyXkFqcGdeQXVyODU5ODY0ODc@._V1_.jpg

[16] https://cdn.mos.cms.futurecdn.net/AV4GuJX9Ssx5QTa9oaYUk6-1024-80.jpg.webp

PRILOZI

I. Python kod GUI-a za pokretanje skripti

```
#import tkinter
from tkinter import *
#from tkinter import ttk
#import sys
#import os
from subprocess import Popen

window=Tk()
window.title("START PROGRAM")
window.geometry('325x265')

def run():
    global process
    process=Popen('python Visual.py')

def stop():
    process.terminate()

def run1():
    global process1
    process1=Popen('python Audio.py')

def stop1():
    process1.terminate()

def run2():
    global process2
    process2=Popen('python Visual_head_eyes.py')

def stop2():
    process2.terminate()
```

```
def run3():
    global process3
    process3=Popen('python Audio_head_eyes.py')

def stop3():
    process3.terminate()

def stop4():
    window.destroy()

btn = Button(window, text="Visual, eyes turn", bg="green",
fg="white",command=run, width=25)
btn.grid(column=0, row=0, padx=2,pady=15)


btn01 = Button(window, text="Close", bg="red",
fg="white",command=stop)
btn01.grid(column=1, row=0)


btn1 = Button(window, text="Audio, head turn", bg="green",
fg="white",command=run1, width=25)
btn1.grid(column=0, row=1,padx=20)

btn11 = Button(window, text="Close", bg="red",
fg="white",command=stop1)
btn11.grid(column=1, row=1)

btn2 = Button(window, text="Visual, head and eyes turn",
bg="green", fg="white",command=run2, width=25)
btn2.grid(column=0, row=2, padx=2,pady=15)

btn21 = Button(window, text="Close", bg="red",
fg="white",command=stop2)
btn21.grid(column=1, row=2)
```

```
btn3 = Button(window, text="Audio, head and eyes turn",
bg="green", fg="white",command=run3, width=25)
btn3.grid(column=0, row=3)

btn31 = Button(window, text="Close", bg="red",
fg="white",command=stop3)
btn31.grid(column=1, row=3)


btn4 = Button(window, text="CLOSE WINDOW", bg="white",
fg="black",command=stop4)
btn4.grid(column=0, row=4, padx=45,pady=15)

window.mainloop()
```