SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1092

Upravljanje gestama virtualnim uređajem MicroSoot

Tomislav Ljubej

Zagreb, lipanj 2015.

Zahvaljujem profesorici Željki Mihajlović i tvrtki AVL AST d.o.o za pruženu pomoć pri izradi ovog rada.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA DIPLOMSKI RAD PROFILA

Zagreb, 6. ožujka 2015.

Predmet: **Diplomski rad**

DIPLOMSKI ZADATAK br. 1092

Pristupnik: **Tomislav Ljubej (0036455331)**
Studij: Računarstvo
Profil: Računarska znanost

Zadatak: **Upravljanje gestama virtualnim uređajem MicroSoot**

Opis zadatka:

Proučiti mehanizme upravljanja gestama u virtualnom okruženju. Razraditi virtualno okruženje uređaja MicroSoot te razraditi prepoznavanje niza gesti kojima je omogućena interakcija i izvođenje definiranih radnji nad ovim uređajem. Ostvariti programsku implementaciju razrađenih gesti te demonstrirati rezultate na slučaju uporabe održavanja leće uređaja. Načiniti ocjenu ostvarenih rezultata.
Izraditi odgovarajući programski proizvod. Rezultate rada načiniti dostupne putem Interneta. Radu priložiti algoritme, izvorne kodove i rezultate uz potrebna objašnjenja i dokumentaciju. Citirati korištenu literaturu i navesti dobivenu pomoć.

Zadatak uručen pristupniku: 13. ožujka 2015.
Rok za predaju rada: 30. lipnja 2015.

Mentor:

Prof. dr. sc. Željka Mihajlović

Predsjednik odbora za diplomski rad profila:

Prof. dr. sc. Siniša Srbljić

Djelovođa:

Doc. dr. sc. Tomislav Hrkać

SADRŽAJ

1. Uvod

U zadnjih nekoliko godina počelo se razvijati sve više novih tehnologija za upravljanje računalom ili virtualnim svjetovima unutar računala. Sve su popularniji načini upravljanja koji se baziraju na gestama, obično gestama ruke ili prstiju.

Takvi načini upravljanja su doživjeli veliki uspjeh na "pametnim telefonima" međutim na stolnim računalima još uvijek su najpopularniji načini upravljanja tipkovnica i miš. Unatoč tomu polako se razvijaju novi načini upravljanja računalom. Jedan od njih je uređaj Leap Motion koji omogućava praćenje ruku i prstiju u 3D prostoru pomoću infracrvene kamere. Drugi način praćenja položaja ruke i prstiju u prostoru su razni senzori (akcelerometar, žiroskop, magnetometar itd.) postavljeni na ruku u obliku rukavice i prste te pomoću istih odrediti rotaciju i poziciju šake i prstiju.

U ovom radu će se koristiti Leap Motion te "rukavica za virtualnu stvarnost" koja se razvija na Fakultetu elektrotehnike i računarstva za istraživanje novih načina upravljanja računalom na konkretnom primjeru uređaja MicroSoot. Korisnik će upravljati virtualnim modelom uređaja MicroSoot u svrhu učenja kako raditi sa ili održavati pravi uređaj.

Razradit će se algoritmi za prepoznavanje gesti te će se napraviti usporedba Leap Motiona i rukavice.

2. Upravljanje gestama virtualnim uređajem MicroSoot

2.1. Tehnologije

2.1.1. Programski alat Unity

Unity je alat za izradu interaktivnih 2D i 3D aplikacija, prvenstveno igara. Omogućava znatno jednostavniju i bržu izradu aplikacija i igara u odnosu na konkurentne alate (npr. Unreal Engine ili Cryengine) te omogućavaju isporučivanje aplikacija na mnogo platformi: Windows, Linux, OSX, iOS i Android. Također nudi mogućnosti pisanja skripti u programskom jeziku C# ili Javascript.

Bazni alat je besplatan za komercijalne i nekomercijalne svrhe, dok se neke naprednije opcije poput kompajliranja projekta u oblaku (engl. cloud) i neki dodaci koji olakšavaju izradu pojedinih tipova igara posebno naplaćuju.

Unatoč tome Unity već nakon instalacije dolazi sa mnogo komponenata i alata koji uvelike olakšavaju i ubrzavaju razvoj te omogućavaju brzu izradu prototipa, što pogoduje akademskim projektima poput ovog.

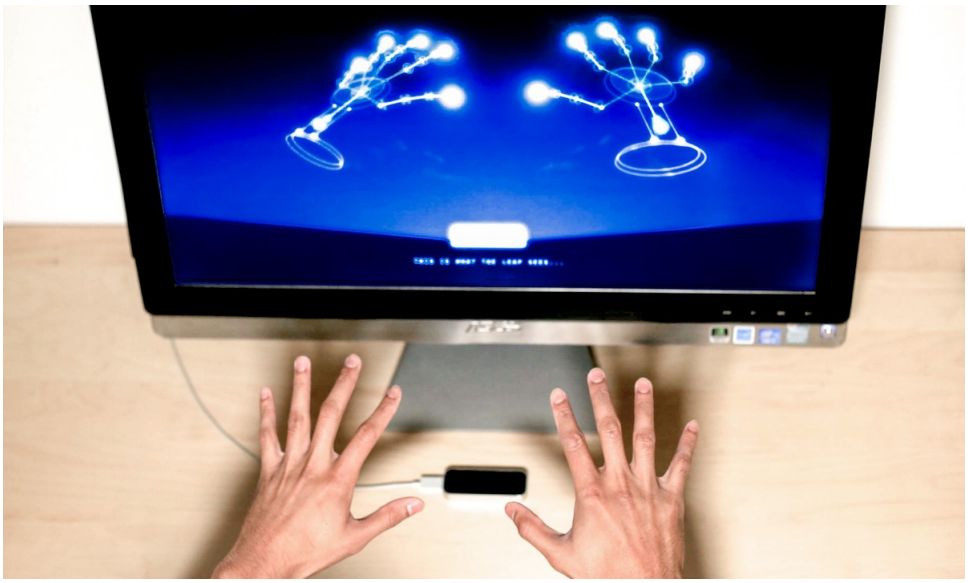
2.1.2. Uređaj Leap Motion

Leap Motion je uređaj koji pomoću infracrvenih LED dioda i dvije infracrvenih kamere može istodobno pratiti položaj obje ruke i svih prstiju.

Radi na način da infracrvene LED diode obasjavaju prostor infracrvenom svjetlošću, a dvije infracrvene kamere (koje imaju široko područje snimanja kako bi se mogao koristiti čim veći prostor za praćenje) snimaju u 300 sličica u sekundi te šalju podatke na računalo. Na računalu se pomoću naprednih algoritama računalnog vida određuje pozicija i rotacija šake, podlaktice te svih prstiju. Korisnik tada može koristiti izračunate podatke za svoje potrebe u mnogo programskih jezika.

Prednost uređaja je jednostavnost namještanja i korištenja te što ne troši mnogo struje i ne zauzima mnogo prostora.

Nedostaci su povremena nepreciznost u određivanju položaja prstiju te situacije u kojima je jedna ruka zaklonjena (npr. ako stavimo jednu ruku iznad druge uređaj neće moći odrediti položaj ruke koja je dalje od uređaja jer ju prva ruka zaklanja).



Slika 2.1: Primjer korištenja Leap Motion uređaja.

2.1.3. Uređaj Microsoft Kinect

Microsoft Kinect je multifunkcionalni uređaj koji prvenstveno služi prepoznavanju ljudskih oblika i lica sa svrhom da korisnik može upravljati konzolom XBox ili računalom bez tradicionalnih upravljača (kontroler, tipkovnica i miš).

Kinect se sastoji od kamere, mikrofona te dubinskog senzora. Dubinski senzor radi na principu da obasjava prostor ispred sebe s mrežom malih infracrvenih snopova te onda pomoću infracrvene kamere može izgraditi trodimenzionalnu sliku prostora ispred sebe.

Microsoft isto tako pruža programsko sučelje za lakše korištenje Kinecta: nudi mogućnost prepoznavanja i praćenja do 6 ljudi u isto vrijeme, mogućnost prepoznavanja lica, jednostavno dobivanje 3D dubinske slike itd.

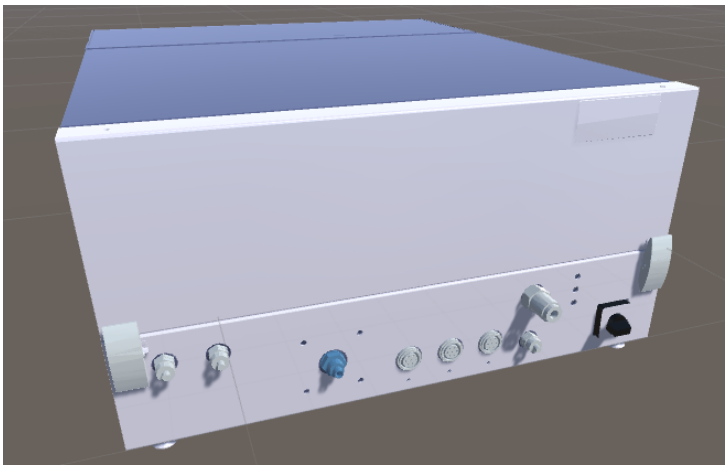
Za potrebe ovoga radi koristit će se samo mali podskup tih mogućnosti.

Koristit će se novije izdanje uređaja koje dolazi s konzolom Xbox One kojeg odlikuje veća rezolucija i time bolja detekcija čovjeka.

Slika 2.2: Uređaj Microsoft Kinect, novije izdanje.

2.1.4. Uređaj Micro Soot

Micro Soot je uređaj tvrtke AVL koji mjeri koncentraciju čađi u ispušnim plinovima motora. U ovome radu koristimo virtualni model uređaja Micro Soot kako bi korisniku demonstrirali način održavanja uređaja. Korisnik će na ekranu vidjeti virtualni model ruke koja će pratiti sve geste koje korisnik napravi svojom pravom rukom. Tom virtualnom rukom će moći rotirati uređaj ili ga približiti ekranu te obavljati određene zadatke poput otvaranja poklopca uređaja, čišćenja dijelova uređaja, pritiskanja gumba itd.



Slika 2.3: Virtualni model uređaja Micro Soot koji koristimo u radu.

2.1.5. Mikrokontroler Arduino

Arduino je mikrokontroler što zapravo znači da je računalo s prilično ograničenom procesorskom moći u odnosu na prosječno stolno računalo ali se zato odlikuje vrlo malom potrošnjom energije, vrlo je prenosiv te prodaje se po relativno niskoj cijeni. Zbog tih svojstava koristi se u izradi ugrađenih sustava u kojima upravlja ili prati rad više elektroničkih uređaja.

Za razliku od komercijalnih mikrokontrolera Arduino je zbog svoje cijene i jednostavnosti korištenja posebno prikladan za edukaciju i akademske projekte.

2.1.6. Rukavica za virtualnu stvarnost

Rukavica za virtualnu stvarnost razvijena na FER-u radi na bitno drugačijem principu od uređaja Leap Motion.

Za izvedbu rukavice korišteno je mnogo uređaja:

- IMU senzor, model MPU-9150
- Arduino mikrokontroler
- Bend senzori
- Uređaj Microsoft Kinect

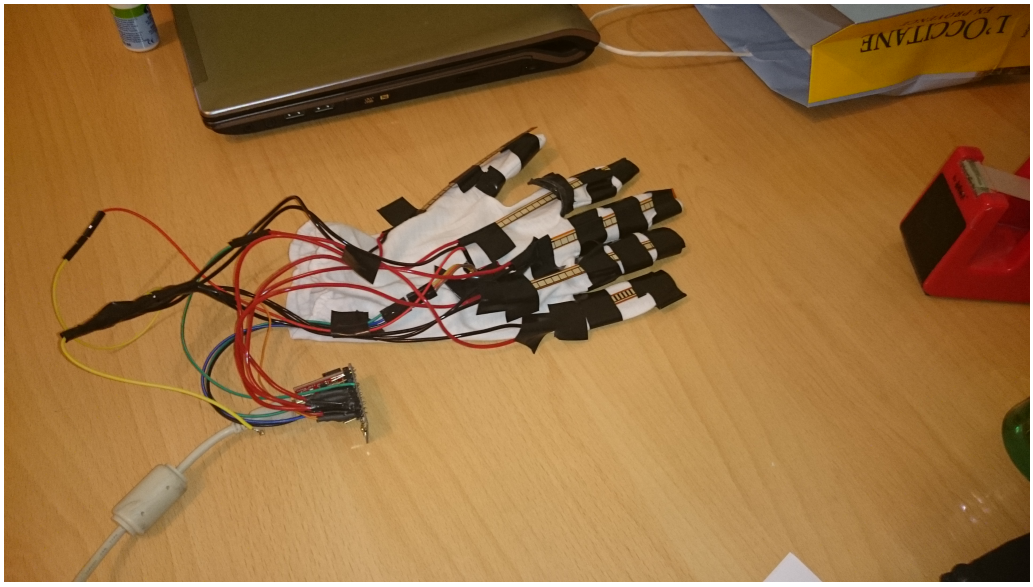
IMU senzor služi za određivanje orijentacije rukavice. Zalijepljen je na dlan rukavice te se tako pomiče i okreće s korisnikovim dlanom. To postiže kombinacijom akcelerometra, žiroskopa te magnetometra.

Bend senzori služe za mjerenje savinutosti svakog od prstiju ruke. Rade na principu da povećavaju svoj otpor proporcionalno s kutom pod kojim su svinuti. Zalijepljeni su po jedan na svaki prst tako da se savijaju kako se i prst savija.

Arduino služi povezivanju IMU i bend senzora te računanju orijentacije rukavice pomoću Kalmanovog filtra. Kalmanov filtar je estimacijski algoritam koji kao ulaz prima podatke sa šumom i nepreciznošću kroz neko vrijeme a kao izlaz daje preciznije i stabilnije rezultate od onih dobivenih samo jednim mjerenjem.

Uređaj Microsoft Kinect služi samo za određivanje pozicije ruke u prostoru.

Više o rukavici možete pronaći u literaturi[1].



Slika 2.4: Rukavica za virtualnu stvarnost.

2.2. Reprezentacija ljudske ruke u virtualnom okruženju

Ljudska ruka je složen dio ljudskog organizma. Čovjek ju može okretati u svim smjerovima, a svaki prst sadrži tri kosti koje se mogu pomicati u odnosu na ruku. Takva složenost predstavlja problem za realizaciju uređaja koji bi pratili kretnje ruku i prstiju u virtualnom okruženju.

Moguća su dva načina za rješavanje ovog problema: Jedan se bazira na detekciju položaja i orijentacije ruke i prstiju pomoću infracrvenih kamera i infracrvenih LED dioda, a drugi na postavljanju senzora na čovjekovu ruku. Oba pristupa imaju svoje prednosti i nedostatke koje ćemo razmotriti.

U ovome radu koristit ćemo alat Unity koji uvelike olakšava prikaz i animaciju 3D modela. Za implementaciju upravljanja koristit ćemo programski jezik C# te biblioteku za neuronske mreže "NeuronDotNet".

Vektori i kvaternioni

U ovome radu koristimo trodimenzionalne vektore (uređeni skup tri realne vrijednosti) za prikaz pozicije u 3D prostoru te kvaternione za opis rotacije objekta.

Svaka komponenta vektora predstavlja jednu os u prostoru (x, y ili z). Zapisujemo ih na sljedeći način:

$\vec{v} = (x, y, z)$

Kvaternioni su nešto složeniji. To su zapravo četverodimenzionalna ekstenzija kompleksnih brojeva, zapisujemo ih na sljedeći način:

$q = x\mathbf{i} + y\mathbf{j} + z\mathbf{k} + w$

Gdje vrijedi:

$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1$

U zapisu rotacije kvaternionima *x*, *y*, *z* vrijednosti predstavljaju jedinični vektor smjera objekta dok *w* predstavlja rotaciju objekta oko tog vektora. Ako definiramo da je θ kut rotacije oko vektora smjera tada vrijedi sljedeći zapis kvaterniona:

$q = \cos(\frac{\theta}{2}) + (x\mathbf{i} + y\mathbf{j} + z\mathbf{k})\sin(\frac{\theta}{2})$

Ovakav kvaternion se naziva rotacijski kvaternion. Prednost rotacijskih kvaterniona u odnosu na zapis rotacije Eulerovim kutovima je nemogućnost pojave blokade kardana (engl. gimbal lock) što zapravo znači gubitak jednog stupnja slobode kod rotacija ako se prethodno napravi neka nezgodna rotacija.

Kvaternioni su vrlo složeno područje pa ih tako nećemo detaljnije objašnjavati jer se u radu već ionako koriste postojeće implementacije.

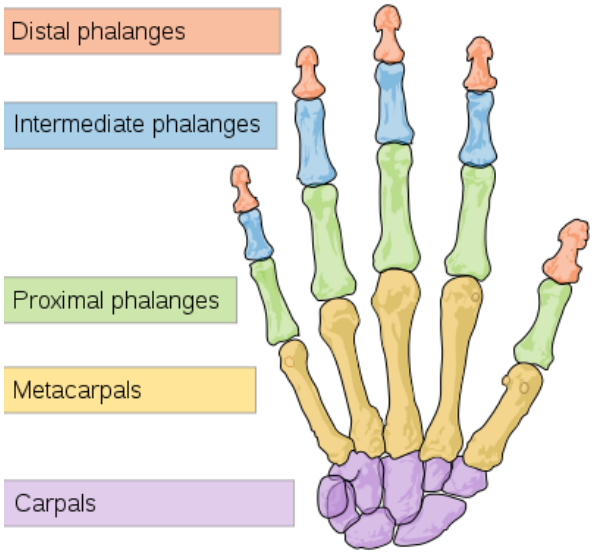
Potrebni podaci za prikaz ruke

U ovome radu zbog jednostavnosti te zbog činjenice da je rukavica za virtualnu stvarnost napravljena da radi samo za desnu ruku ćemo se baviti prikazom samo desne ruke. Na slici 2.5 vidimo sve kosti jedne ruke. Možemo primijetiti da svaki prst osim palca ima tri kosti. Svi prsti imaju tri pomične kosti u odnosu na karpalne kosti: kod svih prstiju osim palca su to proksimalni, srednji te distalni članak dok su kod palca to proksimalni i distalni članak ali i pripadajuća metakarpalna kost, koja je kod drugih prstiju fiksna u odnosu na karpalne kosti.

Iz ovoga možemo zaključiti da su nam potrebne sljedeće vrijednosti, za vrijednosti položaja koristi se 3D vektor a za opis rotacije koristi se kvaternion:

- Položaj sredine šake
- Rotacija cijele šake
- Rotacija proksimalnih članaka svih prstiju
- Rotacija srednjih članaka svih prstiju (osim palca)
- Rotacija metakarpalne kosti palca

S obzirom da namjeravamo koristiti dvije različite tehnologije za dobivanje podataka dobra je ideja napraviti zajedničko programsko sučelje za oba rješenja kako bismo ih mogli lako izmjenjivati po potrebi.



Slika 2.5: Kosti ruke i prstiju.

```
interface IHandInfoProvider {
     void UpdateData();

    bool IsPalmPositionAvailable { get; }
    Vector3 PalmPosition { get; }

    bool IsPalmRotationAvailable { get; }
    Quaternion PalmRotation { get; }

    bool IsThumbAvailable { get; }
    Quaternion[] Thumb { get; }

    bool IsIndexAvailable { get; }
    Quaternion[] Index{ get; }

    bool IsMiddleAvailable { get; }
    Quaternion[] Middle { get; }

    bool IsRingAvailable { get; }
    Quaternion[] Ring { get; }

    bool IsPinkyAvailable { get; }
```

```
    Quaternion[] Pinky { get; }
}
```

Metoda *UpdateData()* dohvaća podatke koji su važeći u trenutku pozivanja metoda s izvora (LeapMotiona ili rukavice za virtualnu stvarnost) i posprema ih u svojstva *PalmPosition*, *PalmRotation* itd. Polja *Thumb*, *Index*, itd. su uvijek veličine 4 jer toliko svaki prst ima kostiju (osim palca, za njega se uzima da je prvi član u polju "prazna", nepostojeća kost). Na indeksu 0 je rotacija zgloba koji je najbliži karpalnim kostima.

Boolean varijable *IsPalmPositionAvailable*, *IsPalmRotationAvailable*, itd. govore korisniku sučelja da li su ti podaci na izvoru dostupni tek korisnik može, ako nisu, koristiti neke svoje pretpostavljene vrijednosti.

2.2.1. Programsko sučelje uređaja Leap Motion

Leap Motion pruža elegantno programsko sučelje za dobivanje podataka o položaju i orijentaciji obje ruke. Pošto radimo u jeziku C# opisujemo korištenje Leap Motiona u istome.

Prvo moramo uključiti Leap biblioteku u naš kod:

```
using Leap;
```

Prije korištenja podataka iz Leap uređaja moramo instancirati novi *Controller* objekt:

```
var controller = new Controller();
```

Sada možemo iz *Frame* objekta izvlačiti podatke o položaju i orijentaciji ruku i prstiju. Idući primjer prikazuje kako dobiti položaj dlana svake ruke i ispisati ga na ekran:

```
var frame = controller.Frame();
foreach (var h in frame.Hands)
{
  Console.WriteLine(h.PalmPosition.x, h.PalmPosition.y,
                    h.PalmPosition.z);
}
```

Sada možemo izgraditi metodu *UpdateData()*:

```
public void UpdateData()
{
    _dataAvailable = false;
```

```
if (!_controller.IsConnected)
{
    return;
}

foreach (var h in _controller.Frame().Hands)
{
    if (h.IsRight)
    {
        _dataAvailable = true;
        _palmPosition = h.PalmPosition.ToUnityScaled();
        _palmRotation = h.Basis.Rotation() * _reorient;

        foreach (var f in h.Fingers)
        {
            if (f.Type() == Finger.FingerType.TYPE_THUMB)
            {
                _SetBoneRotations(_thumb, f);
            }
            else if (f.Type() == Finger.FingerType.TYPE_INDEX)
            {
                _SetBoneRotations(_index, f);
            }
            else if (f.Type() == Finger.FingerType.TYPE_MIDDLE)
            {
                _SetBoneRotations(_middle, f);
            }
            else if (f.Type() == Finger.FingerType.TYPE_RING)
            {
                _SetBoneRotations(_ring, f);
            }
            else if (f.Type() == Finger.FingerType.TYPE_PINKY)
            {
                _SetBoneRotations(_pinky, f);
            }
        }
```

```
        }
    }
}
```

Boolean varijabla *_dataAvailable* je zapravo vrijednost svake boolean varijable koje sadrži sučelje *IHandInfoProvider*

Metoda *_SetBoneRotations(...)* je vrlo jednostavna, samo postavlja orijentacije kostiju u oblik koji pogoduje sučelju *IHandInfoProvider*.

```
private void _SetBoneRotations(Quaternion[] bones, Finger finger)
{
    bones[0] = finger.Bone(Bone.BoneType.TYPE_METACARPAL)
     .Basis.Rotation() * _reorient;

    bones[1] = finger.Bone(Bone.BoneType.TYPE_PROXIMAL)
     .Basis.Rotation() * _reorient;

    bones[2] = finger.Bone(Bone.BoneType.TYPE_INTERMEDIATE)
     .Basis.Rotation() * _reorient;

    bones[3] = finger.Bone(Bone.BoneType.TYPE_DISTAL)
     .Basis.Rotation() * _reorient;
}
```

Kvaternion *_reorient* definiran je na sljedeći način i koristi se da bi se rotacije dobivene iz Leap Motion APIa bile pravilno usmjerene u koordinatnom sustavu koji koristi Unity:

```
private Quaternion _reorient = Quaternion.Inverse(
  Quaternion.LookRotation(Vector3.left, -Vector3.up));
```

Leap Motion pruža za alat Unity modele ruku te neke skripte za upravljanje istima što znatno olakšava prikaz podataka dobivenih iz APIja u nekoj Unity sceni.

Leap Motion API pruža još neke funkcionalnosti poput dohvaćanje same slike dobivene iz kamera te vrlo ograničeno prepoznavanje uzoraka, npr. da li su prsti skvrčeni u šaku ili ne. Za potrebe ovog rada načinit će se vlastita implementacija prepoznavanja uzoraka pa ove dodatne mogućnosti nećemo koristiti.

2.2.2. Programsko sučelje rukavice za virtualnu stvarnost

Povezivanje rukavice za virtualnu stvarnost sa alatom Unity je bio znatno veći posao od povezivanja uređaja Leap Motion zbog raznovrsnosti tehnologija koje su korištene za izradu.

Dobivanje pozicije pomoću uređaja Microsoft Kinect

Microsoft pruža programsko sučelje za dobivanje podataka iz uređaja preko programskog okruženja .NET. Unity ne podržava verziju .NET okruženja koju koristi sučelje za uređaj međutim na sreću Microsoft je pružio omotač (engl. wrapper) tog sučelja za Unity.

Programsko sučelje vrši detekciju položaja ljudi i njihovih udova koji se nalaze ispred uređaja te naravno daje korisniku pristup tim podacima. U ovome radu koristit ćemo samo informaciju o položaju desne šake.

Uređaj prvo moramo inicijalizirati na sljedeći način:

```
KinectSensor ks = Kinect.KinectSensor.GetDefault();
ks.Open();

BodyFrameReader bfr = ks.BodyFrameSource.OpenReader();
```

Klasa *BodyFrameReader* izvlači podatke o lokaciji i orijentaciji prepoznatih ljudskih oblika.

Sada se moramo pretplatiti na događaj koji pruža objekt *bfr* u kojem će biti sadržana informacija o lokaciji i orijentaciji prepoznatih ljudskih oblika.

```
bfr.FrameArrived += _KinectFrameReceived;
```

Na kraju u metodi *_KinectFrameReceived* koja obrađuje događaj *FrameArrived* izvlačimo podatke o položaju desne ruke.

```
private void _KinectFrameReceived(object sender, Kinect.BodyFrameArri
{

    using (Kinect.BodyFrame bf = e.FrameReference.AcquireFrame())
    {
        if (bf != null)
        {
            var bodies = new Kinect.Body[bf.BodyCount];
            bf.GetAndRefreshBodyData(bodies);
```

```
                if (bodies.Length > 0)
                {
                    foreach (var b in bodies)
                    {
                        if (b.IsTracked)
                        {
                            _palmPositionDataAvailable = true;
                            Dictionary<Kinect.JointType, Kinect.Joint>
                              joints = b.Joints;
                            Kinect.CameraSpacePoint csp =
                              joints[Kinect.JointType.HandRight]
                               .Position;
                            _palmPosition =
                              new Vector3(csp.X, csp.Y, csp.Z);
                        }
                    }
                }
            }
        }
    }
```

Dobivanje podataka iz mikrokontrolera Arduino

Arduino s računalom komunicira preko serijskog porta međutim pošto današnja računala sve rjeđe imaju serijski port koristimo USB adapter koji emulira serijski port. Unity u teoriji podržava direktno čitanje sa serijskog porta međutim s obzirom da Unity radi s programskim okruženjem Mono (koji je otvorena implementacija .NET okruženja) što znači da povremeno neke stvari ne rade kako bi trebale, među njima je i podrška za serijsku komunikaciju.

Kao rješenje smo u pravom .NETu implementirali UDP server koji prenosi podatke s Arduina preko mreže (u ovom slučaju se to događa lokalno unutar istog računala preko virtualnog "loopback" priključka koji šalje mrežne pakete unutar istog računala).

Server je vrlo jednostavan, cijelo vrijeme dok radi prikuplja najnovije podatke s Arduina preko serijskog porta a kada primi bilo kakav podatak od aplikacije šalje trenutno aktualne podatke preko mreže aplikaciji. Podaci dolaze u obliku niza ASCII

znakova u sljedećem formatu:

```
A3B3DC55, 43216DAD, BE2D2344, DE2451A9, 5, 55, 45, 67, 21
```

Primijetimo da ima sveukupno 9 niza znakova odvojenih zarezom. Prva četiri su bajtovi brojeva (zapisanih u heksadecimalnom formatu) s pomičnim zarezom zapisanih u IEEE 754 formatu (32 bitna varijanta) te predstavljaju *x*, *y*, *z* i *w* vrijednosti kvaterniona rotacije šake. Idućih 5 brojeva je "količina savinutosti" svakog od prstiju, počevši od palca, te se u teoriji kreće između 0 i 100, međutim kako prst ne može saviti bend senzor toliko da dođe do maksimalne savijenosti vrijednost se za sve prste osim palca popne do otprilike 75, a za palac do 65.

Nakon što smo dobivene podatke pretvorili iz tekstualnog oblika u oblik razumljiv računalu možemo implementirati sučelje *IHandInfoProvider*.

Kvaternion rotacije kojeg dobijemo iz IMU senzora ne možemo izravno postaviti kao kvaternion rotacije virtualne šake jer se koordinatni sustavi senzora i Unitya ne poklapaju. Kvaternion rotacija dobiven iz Arduina se mora s desna pomnožiti s kvaternionom koji predstavlja rotaciju za 90°oko x-osi.

S obzirom da IMU senzor koristi magnetometar početna rotacija kada se uređaj u pali ovisit će o tome gdje je smjer sjevera, a pošto naravno računala nisu uvijek okrenuta prema sjeveru moramo imati neki način da kažemo aplikaciji da je trenutna orijentacija rukavice ona koja gleda prema monitoru.

Zato smo implementirali da pritiskom na tipku "H" te orijentiranje ruke da pokazuje prema monitoru neposredno nakon paljenja aplikacije se postavlja referentni kvaternion rotacije na trenutnu vrijednost iz Arduina te i taj kvaternion naravno pomnožimo s kvaternionom koji predstavlja rotaciju za 90°po x-osi.

Na kraju imamo konačnu formulu za dobivanje rotacije rukavice u Unityu:

```
_palmRotation = _referentRotation *
    (new Quaternion(_arduinoData.Data[0], _arduinoData.Data[1]
       ,_arduinoData.Data[2],_arduinoData.Data[3]))
    * Quaternion.Euler(new Vector3(90,0,0))
```

gdje je *_referentRotation* referentni kvaternion dobiven pravilnim usmjerivanjem rukavice i pritiskom na tipku "H" a *_arduinoData.Data[0..3]* su redom *x*, *y*, *z*, *w* vrijednosti kvaterniona dobivenog iz Arduina.

S obzirom da iz svakog bend senzora dobijemo samo koliko je pojedini prst savinut a ne točne rotacije svakog zgloba kao u sučelju za uređaj Leap Motion moramo napraviti neke pretpostavke o tome kako izgleda savijen prst. Ako savijemo prst do

kraja vidjet ćemo da je kut svakog zgloba otprilike 90°. Već smo prije spomenuli da vrijednost savinutosti za većinu senzora je otprilike 75 kada je prst maksimalno savijen. Zato ćemo prvo skalirati tu vrijednost na interval $[0, 1]$ te onda pomnožiti sa 90°. To će biti vrijednost kuta svakog zgloba u tom prstu.

Rotaciju zglobova vršimo uzastopnim množenjem kvaterniona. Rotacija prvog zgloba će biti umnožak rotacijskog kvaterniona šake i kvaterniona koji predstavlja rotaciju za onoliko stupnjeva koliko smo izračunali u prethodnom koraku po z-osi. Rotacija idućeg zgloba će biti umnožak rotacije prijašnjeg i opet kvaterniona koji predstavlja rotaciju po z-osi za istu količinu stupnjeva. Isto naravno ponavljamo i za treći zglob. Za palac malo mijenjamo postupak jer kao što smo već rekli savinutost kod palca ne prelazi 65 te palac je u početku drukčije orijentiran u odnosu na šaku od ostatka prstiju pa moramo njegovu rotaciju na početku postaviti na neku smislenu vrijednost.

Kod za sve prste osim palca:

```
private Quaternion[] _FingerBendInterpolation(float bend)
{
    float ang = bend/75.0f * 90;

    // prvi zglob
    Quaternion f1 = _palmRotation * Quaternion.Euler(0, 0, -ang);

    // drugi zglob
    Quaternion f2 = f1 * Quaternion.Euler(0, 0, -ang);

    // treci zglob
    Quaternion f3 = f2 * Quaternion.Euler(0, 0, -ang);

    return new Quaternion[]
    {
        f1,
        f2,
        f3
    };
}
```

Koristimo negativne vrijednosti kuta jer je koordinatni sustav prstiju tako postavljen.

Za palac je kod vrlo sličan. Početna rotacija je odabrana na način da izgleda čim prirodnije:

```
private Quaternion[] _FingerBendInterpolationThumb(float bend)
{
    float ang = bend/60.0f * 90;

    Quaternion f1 = _palmRotation * Quaternion.Euler(new Vector3(298,
    Quaternion f2 = f1 * Quaternion.Euler(0, 0, -ang);
    Quaternion f3 = f2 * Quaternion.Euler(0, 0, -ang);

    return new Quaternion[]
    {
        f1,
        f2,
        f3
    };
}
```

2.3. Interakcija s uređajem

Nakon dobivanja podataka o ruci i prstima moramo iskoristiti te podatke za interakciju s virtualnim prostorom, u ovom konkretnom primjeru s uređajem Micro Soot.

Upravljanje uređajem će se realizirati kombinacijom gesti prstiju te geste pokretima ruke.

2.3.1. Geste prstiju

U ovome radu gestu prstiju definiramo kao skup položaja prstiju u odnosu na dlan. Najočitiji primjeri su kada korisnik skvrči sve prse u šaku ili kada potpuno izravna sve prste. Tako na primjer možemo omogućiti korisniku da rotira uređaj oko svoje osi kako bi ga mogao gledati iz drugog kuta tako da skvrči prste u šaku i pomiče šaku lijevo ili desno.

Očito je da je potrebno osmisliti klasifikator koji bi odredio točno o kojoj se gesti radi. Za tu svrhu koristit ćemo neuronske mreže.

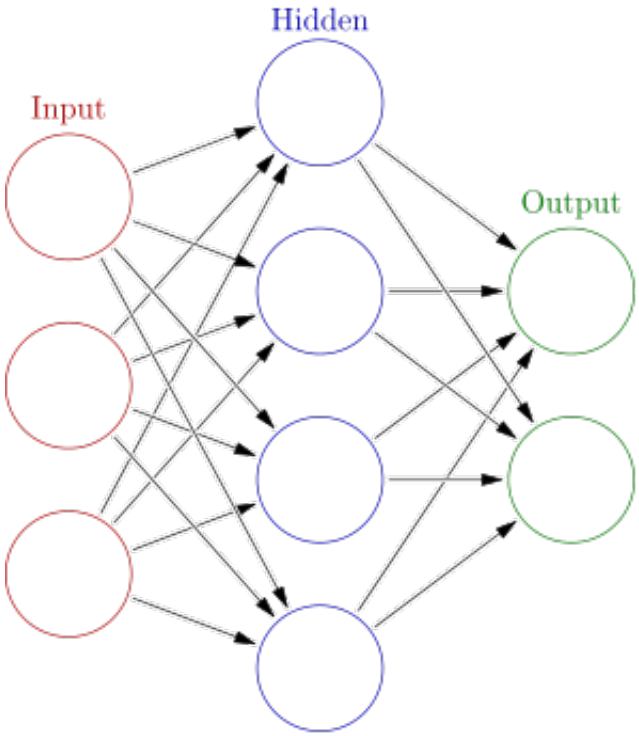
Neuronske mreže

Neuronske mreže su inspirirane pravim neuronima u živim bićima. Koriste se kako bi se aproksimirale funkcije s mnogo ulaza a koje nisu na jednostavan način izračunljive.

Neuronske mreže sastoje se od više slojeva neurona:

- Ulazni sloj, sastoji se od onoliko neurona koliko ima ulaznih vrijednosti
- Jedan ili više skrivenih slojeva, sastoji se od proizvoljnog broja neurona
- Izlazni sloj, također može imati proizvoljan broj izlaza ovisno o izvedbi

Svaki neuron se sastoji od više ulaza te jednog izlaza. Svaki ulazni neuron ima samo jedan ulaz te obično mu je zadatak samo da prenese taj ulaz neuronima idućeg sloja. U svakom idućem sloju svaki neuron ima onoliko ulaza koliko prethodni sloj ima neurona.



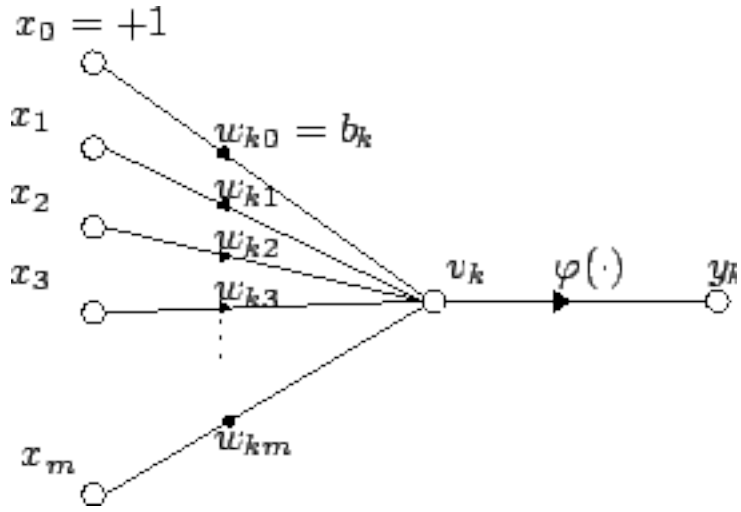
Slika 2.6: Skica neuronske mreže.

Svaki neuron je modeliran na sljedeći način:

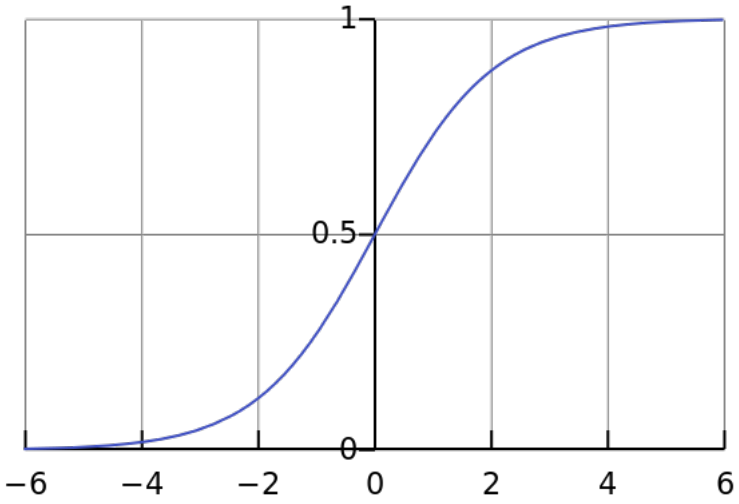
Ako neuron ima n ulaza kao što je prikazano na slici 2.7 tada se izlaz neurona računa po sljedećoj formuli:

$y_k = \varphi\left(\sum_{j=0}^{m} w_{kj}x_j\right)$

Gdje je $\varphi()$ takozvana aktivacijska funkcija neurona. U našem slučaju to će biti funkcija sigmoide $\varphi(t) = \frac{1}{1+e^{-t}}$ kako bi se izlazne vrijednosti zadržale između 0 i 1.



Slika 2.7: Neuron.



Slika 2.8: Funkcija sigmoide.

Vrijednosti $x_1..x_m$ (vrijednost x_0 je uvijek 1 te postoji samo zbog jednostavnosti zapisa izlaza) su ulazi u neuron a vrijednosti $w_{k0}..w_{kj}$ su težinske vrijednosti neurona.

Ako želimo da neuronska mreža radi nešto korisno moramo ju naučiti podacima za učenje. Podaci za učenje čine skup parova ulaza u mrežu i očekivanih izlaza iz mreže. Mreža mora imati onoliko ulaza i izlaza koliko ih imaju i podaci za učenje. Skup ulaza u neuronsku mrežu još u kontekstu strojnog učenja nazivamo vektorom značajki.

Neuronska mreža se može učiti na puno načina, najpopularniji su algoritam backpropagation te genetski algoritam. Algoritam backpropagation radi na način da za svaki ulaz svakog podatka za učenje izračuna izlaz te ga usporedi s očekivanim rezultatom i prema tome korigira težinske vrijednosti svih neurona da rezultat bude "malo

točniji" nego prije. To se radi u puno iteracija (10000) te se na kraju dobiva neuronska mreža koja daje točne rezultate za podatke za učenje.

Genetski algoritam funkcionira na način da generira nasumično više neuronskih mreža s nasumičnim težinskim vrijednostima te onda ocjenjuje koliko svaka generirana mreža daje točne izlaze, te onda "križa" i "mutira" najbolje mreže. Nakon dovoljnog broja iteracija također dobivamo mrežu koja daje dobre izlaze.

Prepoznavanje gesti prstiju

U ovome radu koristimo backpropagation algoritam. Kao ulaze koristimo kvaternione koji predstavljaju rotacije pokretnih zglobova svih prstiju ruke, to je dakle četiri vrijednosti za tri pokretna zgloba u svih pet prstiju, sveukupno $5 \times 3 \times 4 = 60$. Mreža će imati onoliko izlaza koliko gesti želimo prepoznati, dakle ako želimo prepoznati samo geste ispruženih prstiju i skvrčenih u šaku imat ćemo samo dva izlaza.

Svakom izlazu ćemo dakle pridijeliti jednu klasu, te će svi primjeri za učenje koji predstavljaju tu klasu imati 1 za taj izlaz a 0 za sve ostale.

Kada ćemo čitati izlaze iz mreže za prave primjere oni će za svaku klasu davati neku vrijednost između 0 i 1. Time dobivamo mogućnost da prihvatimo samo one klasifikacije u koje je mreža jako sigurna (npr. ako je izlaz za gestu "skvrčeni prsti" veći od 0.98 tek tada prihvaćamo klasifikaciju te geste). Tako ćemo izbjegavati klasifikaciju nekih gesti koje su na granici gesti koje želimo prepoznati (npr. ako mreža vrati vrijednost 0.55 za "skvrčeni prsti" gestu a 0.45 za "ispruženi prsti" gestu).

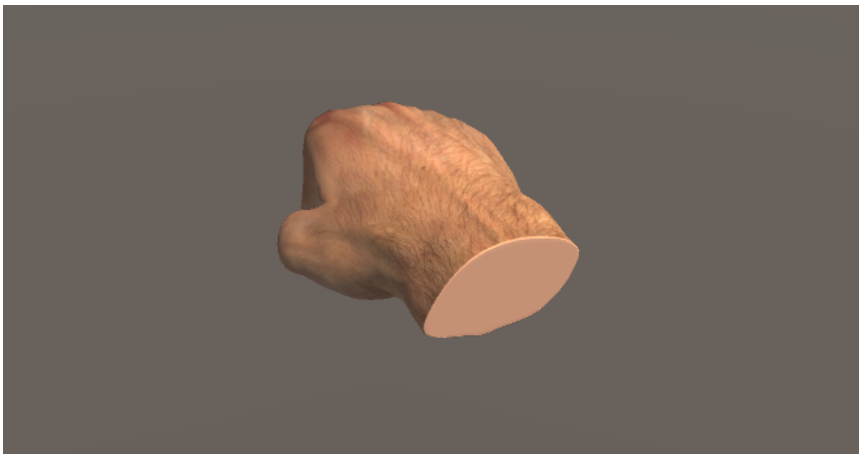
U ovome radu snimali smo uzorke za tri različite geste:

- Ispružena ruka (Slika 2.9)
- Skvrčena ruka (Slika 2.10)
- Upiranje prstom (Slika 2.11)

Za svaku klasu snimljeno je oko 5 uzoraka. Uzorci za pojedinu klasu su međusobno vrlo slični ali opet dovoljno različiti da neuronska mreža za pojedinu klasu prepoznaje širi spektar gesti jer korisnik kada pokušava napraviti gestu ne pomakne uvijek sve prste u isti položaj kako to zahtijeva "savršeni" primjer geste.

Eksperimentiranjem se pokazalo da za potrebe ovoga rada nije nužno imati više od jednog skrivenog sloja s onoliko neurona koliko ima klasa (tri u našem slučaju) u neuronskoj mreži jer su svi uzorci u istoj klasi vrlo bliski jedan drugome u prostoru vektora značajki. Takva mreža je već nakon 10000 iteracija vrlo dobro istrenirana dok mrežama s većim brojem neurona treba znatno više te naravno vrijeme izračuna

Slika 2.9: Gesta ispružene ruke.



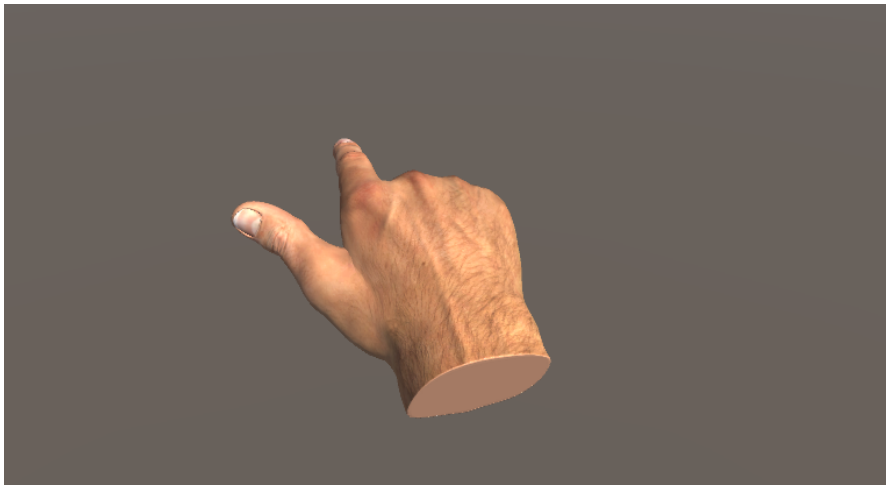
Slika 2.10: Gesta skvrčene ruke.

pojedine iteracije je znatno povećano zbog većeg broja neurona te veza između neurona koje se povećavaju kvadratno s brojem neurona.

2.3.2. Pomicanje uređaja

Pomicanje uređaja je izvedeno kombinacijom gesti prstiju te pokreta ruke. Ako korisnik napravi gestu skvrčene ruke te pomiče ruku lijevo ili desno uređaj će se okretati u tu stranu. To nam omogućuje intuitivno pregledavanje uređaja iz svih kutova.

Rotacija uređaja se vrši tako da, ako korisnik trenutno izvodi gestu skvrčene šake, prilikom svakog osvježavanja ekrana vrši se projekcija točke pozicije šake na ekran. Tada se oduzima razlika između projekcije od prošlog osvježavanja ekrana po x-osi i y-osi. Tada se gleda na kojoj osi je apsolutna vrijednost razlike veća te se onda rotira po onoj drugoj. Dakle ako je razlika veća na x-osi tada će se objekt rotirati oko y-osi i obrnuto.



Slika 2.11: Gesta upiranja prstom.

Slika 2.12: Poza ruke u mirovanju.

Rotaciju uvijek vršimo samo oko jedne osi u isto vrijeme jer inače korisnik može rotirati objekt na način da bude "ukoso", odnosno da bude rotiran i oko z-osi što nema smisla za potrebe ovog rada.

2.3.3. Geste pokreta

Geste pokreta su geste koje korisnik izvodi pomicanjem ruke po prostoru. Ako želimo prepoznavati takav oblik gesti, moramo prvo osmisliti način kako prikazati jednu gestu u računalu te naravno kako ju klasificirati.

Za klasificiranje ćemo koristiti algoritam $P.

Računalni prikaz geste

Gestu ćemo u računalu prikazati kao skup točaka u 2D prostoru.

Da bi izveo gestu korisnik mora prvo napraviti gestu prstiju ispružene ruke te držeći tu gestu napraviti željenu gestu pokreta. Dok traje gesta ispružene ruke prilikom svakog osvježavanja ekrana snima se položaj šake na ekranu na isti način kako je objašnjeno u podpoglavlju "Pomicanje uređaja". Tako izmjerene točke se spremaju u listu točaka geste. S obzirom da se ekran obično osvježava s relativno velikom frekvencijom (obično preko 30 puta u sekundi) da ne bi imali previše točaka u listi točaka geste dodajemo novu točku u listu samo ako je od zadnje točke u listi udaljena više od neke male vrijednosti (npr. 5 piksela).

Sada kada imamo snimljenu gestu preostaje nam klasifikacija.

$P algoritam

$P algoritam razvijen je kolaboracijom sveučilišta u Marylandu, sveučilišta u Washingtonu, te sveučilišta Stefan cel Mare u Rumunjskoj. Dizajniran je da bude jednostavan i brz a opet moćan algoritam za prepoznavanje gesti[2].

Algoritam kao ulaz prima skup 2D točaka te sadrži prije definiranu listu skupova točaka koji služe kao predlošci za razne geste te pokušava odrediti koji predložak je najbliži gesti.

Prije nego što se algoritam može izvesti potrebno je napraviti nekoliko transformacija nad ulaznim skupom točaka te na listi predložaka.

Za algoritam je nužno da dva skupa točaka koji se provjeravaju imaju jednak broj točaka. Naravno kada rukom radimo gestu po ekranu to ne možemo nikako garantirati. Zato se provodi postupak ponovnog uzorkovanja kako bi se ujednačio broj točaka. Za potrebe ovog rada sve veličine svih skupova točaka ćemo svesti na 32 što je dovoljan broj da algoritam ne izgubi na preciznosti a opet da se algoritam izvodi dovoljno brzo na računalu.

Prvi korak postupka ujednačavanja broja točaka je izračunati ukupnu duljinu putanje po skupu točaka. Algoritam za ovo je prilično jednostavan, jednostavno iteriramo po listi točaka te zbrajamo udaljenosti između susjednih točaka:

```
public static double PathLength(List<Point2D> gesture)
{
    double d = 0.0;
    for (int i = 1; i < gesture.Count; i++)
    {
        d += gesture[i - 1].Distance(gesture[i]);
    }
```

```
    return d;
}
```

Metoda *Distance* izračunava euklidsku udaljenost između dvije točke:
$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$.

Nakon što smo izračunali duljinu putanje možemo početi s uzorkovanjem skupa točaka. Prvo izračunamo željenu udaljenost između dvije točke u novo uzorkovanom skupu, nazovimo tu vrijednost *I*. To radimo tako da jednostavno podijelimo ukupnu duljinu putanje sa $N - 1$ Gdje je *N* u našem slučaju 32 kao što je već spomenuto. Nakon toga kao i u prethodnom postupku iteriramo po susjednim točkama i zbrajamo udaljenosti između njih sve dok ne prijeđemo duljinu *I*, tada popravimo poziciju zadnje točke da sveukupna udaljenost ne prijeđe vrijednost *I*. Ako se desi udaljenost između dvije susjedne točke više puta prijeđe duljinu *I* (što se obično dešava kada ima manje točaka na početnom skupu točaka nego u ciljanom skupu) tada ćemo istu točku morati više puta "popravljati". Cijeli algoritam uzorkovanja prikazan je u nastavku:

```
public static List<Point2D> Resample(List<Point2D> gesture, int n)
{
    double I = PathLength(gesture) / (n - 1);
    double D = 0.0;

    List<Point2D> newPoints = new List<Point2D>
      (new Point2D[]{new Point2D(gesture[0])});
    Point2D prevPoint = new Point2D(gesture[0]);
    for (int i = 1; i < gesture.Count; i++)
    {
        double d = gesture[i - 1].Distance(gesture[i]);
        if ((D + d) >= I)
        {
            prevPoint = gesture[i - 1];
            while ((D + d) >= I)
            {
                Point2D q = new Point2D();
                double t = Math.Min(
                            Math.Max((I - D) / d, 0.0f), 1.0f);
                if (Double.IsNaN(t))
                {
```

```
                    t = 0.5;
                }
                q.x = (1.0 - t) * prevPoint.x + t * (gesture[i].x);
                q.y = (1.0 - t) * prevPoint.y + t * (gesture[i].y);
                newPoints.Add(q);
                d = D + d - I;
                D = 0;
                prevPoint = new Point2D(q);
            }
            D = d;
        }
        else
        {
            D += d;
        }
    }

    // Nekada se desi zbog gresaka u
    // zaokruzivanju da se zadnja tocka ne doda
    // pa u tom slucaju moramo kompenzirati
    if (newPoints.Count == n - 1)
    {
        newPoints.Add(new Point2D(gesture.Last()));
    }

    return newPoints;
}
```

Nakon uzorkovanja slijedi skaliranje skupa točaka. To znači svođenje svih točaka u 2D prostor $[-0.5, 0.5] \times [-0.5, 0.5]$. To radimo jer bi inače neke geste pokrivale veću površinu jer se aplikacija može pokretati u mnogo rezolucija. Postupak je prilično jednostavan. Prvo moramo pronaći maksimalne i minimalne *x* i *y* vrijednosti u skupu točaka. Tada računamo razlike između maksimalnih vrijednosti po pojedinoj osi te uzimamo veću od njih i podijelimo sve x i y vrijednosti svih točaka s tim brojem. Implementacija je vrlo jednostavna:

```
public static List<Point2D> Scale(List<Point2D> gesture)
```

```
{
    double xmin = gesture.Min(p => p.x);
    double ymin = gesture.Min(p => p.y);
    double xmax = gesture.Max(p => p.x);
    double ymax = gesture.Max(p => p.y);

    double scale = Math.Max(xmax - xmin, ymax - ymin);

    return gesture.Select
      (p => new Point2D(p.x / scale, p.y / scale)).ToList();
}
```

Sada imamo gestu koja stane u pravokutnik s duljinom i širinom ne većom od 1.

Na kraju moramo gestu centrirati kako bi se stvarno nalazila u prostoru $[-0.5, 0.5] \times [-0.5, 0.5]$. Postupak je jednostavan, prvo izračunamo centar geste na način da napravimo aritmetički prosjek nad svim točkama po x i y osi. Tada jednostavno oduzmemo x i y vrijednosti centra od x i y vrijednosti svake točke u gesti.

```
public static List<Point2D> TranslateToOrigin(List<Point2D> gesture)
{
    Point2D centroid =new Point2D();
    foreach (var p in gesture)
    {
        centroid.x += p.x;
        centroid.y += p.y;
    }

    centroid.x /= gesture.Count;
    centroid.y /= gesture.Count;

    return gesture.Select(p =>
      new Point2D(p.x - centroid.x, p.y - centroid.y)).ToList();
}
```

Sada kada je gesta normalizirana može se početi izvoditi algoritam.

Prvo treba objasniti način na koji algoritam određuje "udaljenost" između dvije geste. To radi na način da uzmemo neke dvije geste, u ovom slučaju gesta koja se pokušava klasificirati i jedna gesta iz skupa predložaka. Za jednu gestu odaberemo

neku početnu točku u listi njenih točaka te za tu točku nalazimo najbližu točku iz druge geste, te tu drugu točku označimo kao "sparenu" i više ju ne razmatramo te sumi udaljenosti pridružimo udaljenost između tih točaka.

Postupak ponavljamo za ostale točke ali kada pribrajamo sumu koristimo težišnu vrijednost koja se smanjuje od 1 prema 0 za svako sljedeće pribrajanje. Ovu heuristiku radimo zato jer s obzirom da sparene točke više ne razmatramo, smanjuje se sigurnost da je upravo ta udaljenost najmanja moguća za tu točku, to možemo tvrditi samo za prvu točku koju smo razmatrali.

```
public static double CloudDistance(List<Point2D> gesture,
   List<Point2D> template, int start)
{
    double sum = 0.0;
    bool[] matched = new bool[template.Count];
    Array.Clear(matched, 0, matched.Length);
    int i = start;
    do
    {
        double min = Double.PositiveInfinity;
        double weight = 0.0;
        int index = -1;
        for (int j = 0; j < template.Count; j++)
        {
            if (matched[j])
            {
                continue;
            }

            double d = gesture[i].Distance(template[j]);
            if (d < min)
            {
                min = d;
                index = j;
            }
        }
        matched[index] = true;
```

```
            weight = 1.0 - (((i - start) + gesture.Count)
               % gesture.Count) / (double)gesture.Count;
            sum += weight * min;

            i = (i + 1) % gesture.Count;

        } while (i != start);

        return sum;
    }
```

Kao što možemo vidjeti složenost ovog algoritma je $O(n^2)$ gdje je n broj točaka u gesti.

Ostatak algoritma se u suštini svodi na računanje udaljenosti promatrane geste sa svim predlošcima ali s različitom početnom točkom, pa je tako ukupna složenost nalaženja minimalne udaljenosti između dvije geste $O(n^3)$. Kao najbolja udaljenost uzima se naravno ona najmanja.

Kao ubrzanje algoritma možemo umjesto da provjeravamo udaljenosti dvije geste za svaku početnu točku možemo za svaku drugu ili treću, ta metoda ne garantira najbolje rješenje ali smanjuje složenost algoritma međutim s obzirom da radimo s gestama od samo 32 točke današnja računala su dovoljno jaka da čak i s maksimalnom složenosti provedu algoritam u stvarnom vremenu.

Odsječak koda u nastavku prikazuje uspoređivanje jedne geste i predloška te vraća najmanju nađenu udaljenost.

```
public static double GreedyCloudMatch(List<Point2D> gesture,
    List<Point2D> template, int n)
{
    double e = 1.0;
    int step = (int)Math.Pow(n, 1.0 - e);
    double min = Double.PositiveInfinity;

    for (int i = 0; i < n; i += step)
    {
        double d1 = CloudDistance(gesture, template, i);
        double d2 = CloudDistance(template, gesture, i);
        min = new double[] { d1, d2, min }.Min();
```

```
    }

    return min;
}
```

Varijabla *e* (zapravo predstavlja grčko slovo ϵ) služi za podešavanje gore spomenute aproksimacije, na primjer ako je $e = 0.5$ provjeravat će se samo udaljenost za $n^{0.5}$ odnosno $\sqrt{n}$ početnih točaka, što smanjuje složenost algoritma sa $O(n^3)$ na $O(n^{2.5})$, odnosno u općenitom slučaju $O(n^{n-\epsilon})$.

Kako bi algoritam bio potpun potrebno je još jedino primijeniti gore opisan postupak između promatrane geste i svakog od predložaka te naravno odabrati onaj predložak koji ima najmanju udaljenost od promatrane geste.

```
public static Result PRecognizer(List<Point2D> gesture,
   List<GestureTemplate> templates, int n)
{
    gesture = Normalize(gesture, n);
    GestureTemplate best = null;
    double dist = Double.PositiveInfinity;
    foreach (var t in templates)
    {
        double newDist = GreedyCloudMatch(gesture, t.gesture, n);
        if (newDist < dist)
        {
            dist = newDist;
            best = t;
        }

    }
    return new Result { score = dist, template = best };
}
```

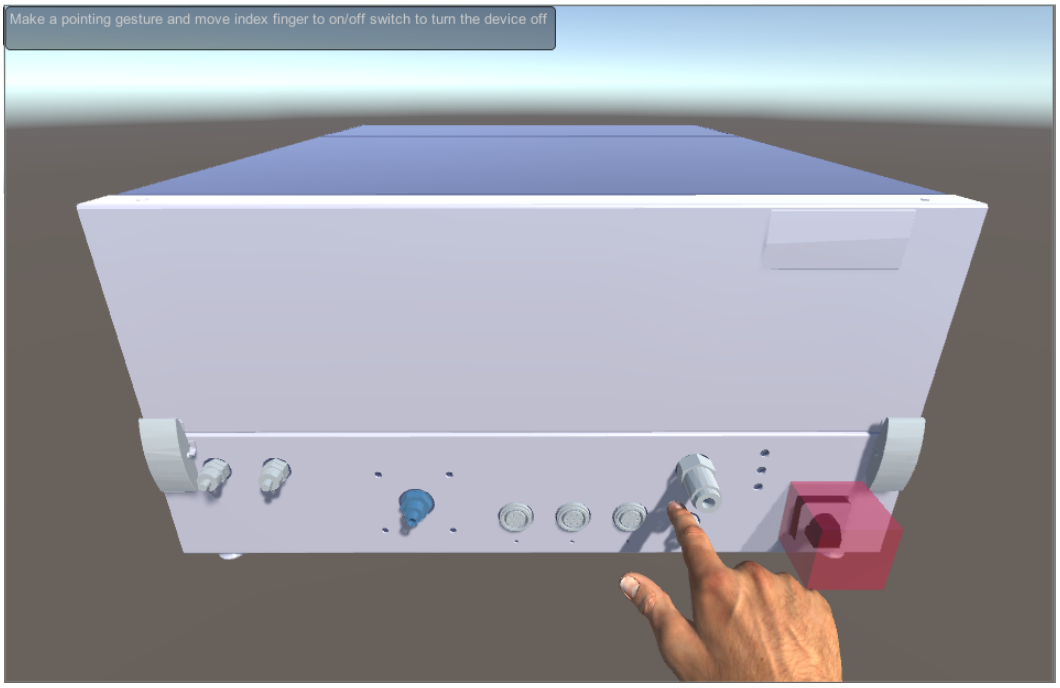
Cijeli pseudokod algoritma možete pronaći u literaturi[3].

2.3.4. Upravljanje uređajem

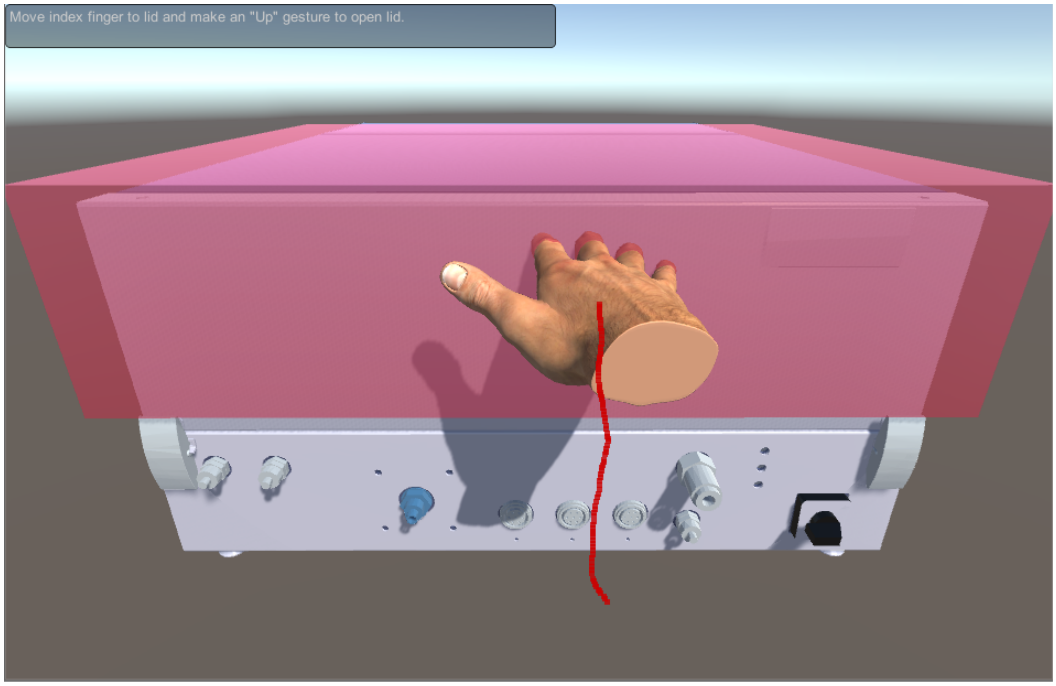
Sada kada imamo implementirane sve potrebne algoritme možemo implementirati primjer održavanja uređaja.

Upravljanje se svodi na to da korisnik pomakne virtualnu ruku na neki dio uređaja te napravi određenu gestu prstiju ili ruke. Na primjer na početku korisnik mora napraviti gestu upiranja prstom te dovesti kažiprst do prekidača za paljenje i gašenje uređaja kako bi ugasio uređaj. Nakon toga treba dovesti ruku do poklopca uređaja te napraviti gestu pokreta prema gora kako bi otvorio uređaj. Korisnik izvodi geste pokreta tako da prvo napravi gestu ispružene ruke te onda pomiče ruku da napravi gestu.

Na sličan način se izvode sve akcije koje korisnik može obaviti nad uređajem.



Slika 2.13: Korisnik izvodi gestu upiranja prstom i pomiče ruku prema označenom području.



Slika 2.14: Korisnik izvodi gestu pomicanja ruke prema gore u označenom području.

2.4. Usporedba korištenih tehnologija

Dvije korištene tehnologije (uređaj Leap Motion i rukavica za virtualnu stvarnost) su vrlo različito izvedene pa tako imaju znatno drukčija svojstva.

2.4.1. Instalacija i postavljanje

Uređaj Leap Motion

Uređaj Leap Motion je bio relativno jednostavan za instalirati. Jedini potrebni koraci su bili instalacija službenog programskog paketa, instalacija Unity dodatka te naravno spajanje uređaja s računalom preko USB porta.

Rukavica za virtualnu stvarnost

Rukavica je bila nešto složenija. Potrebno je bilo instalirati programski paket za Kinect, programski paket za Arduino te upravljačke programe (engl. driver) za emulator serijskog porta preko USB porta.

2.4.2. Korištenje

Uređaj Leap Motion

Leap Motion je brzo prepoznao ruku nakon pokretanja. Poziciju je prepoznavao prilično precizno sve dok ruka nije ulazila u rubne dijelove radnog prostora uređaja (hemisferno područje oko jedan metar iznad uređaja).

Poziciju i orijentaciju šake i prstiju prepoznaje nešto lošije pogotovo ako korisnik radi nagle pokrete, na primjer ako iz ispružene šake pokuša naglo skvrčiti šaku prsti će na ekranu biti u vrlo čudnoj poziciji. S obzirom da se uređaj sastoji od samo jedne komponente sustav je prilično robusan i jednostavan za korištenje.

Uređaj radi zadovoljavajuće dobro za većinu potreba.

Rukavica za virtualnu stvarnost

Kod rukavice je stvar drugačija.

Kinect ima problema s pozicioniranjem šake ako se korisnik nalazi preblizu, tada pozicija zna skakati brzo iz jedne pozicije u drugu znatno udaljenu. Također ima malih problema kod prepoznavanje šake kada korisnik nosi rukavicu jer iz rukavice strši dosta žica međutim to bi se moglo lako riješiti tako da se skrate žice.

IMU senzor radi prilično brzo i precizno no ima i nedostatke. Kao što je već prije spomenuto mora se pri pokretanju kalibrirati da se rotacija poklapa sa stanjem na ekranu te ima problem sa tzv. "driftom", ako ga se pusti u stanju mirovanja može se primijetiti da se ruka na ekranu sama od sebe okreće. To se događa zbog izvedbe magnetometra te je čest problem u takvim uređajima. Bend senzori rade prilično dobro međutim nekada se zna desiti da im vrijednost počne znatno skakati u kratkom vremenskom roku pa se na ekranu to manifestira tako da prsti na virtualnoj ruci brzo titraju. Ovi problemi bi vjerojatno bili umanjeni da su se koristili skuplji senzori.

S obzirom da se rukavica sastoji od mnogo dijelova postavljanja i korištenje je malo otežano u odnosu na uređaj Leap Motion, što je i za očekivati s obzirom da se ne radi o gotovom komercijalnom proizvodu kao što je Leap Motion.

U konačnici ako izuzmemo napor pri instalaciji i postavljanju jer se radi o prototipu rukavica radi prilično dobro unatoč manjim smetnjama.

3. Zaključak

U ovome radu predstavljene su dvije tehnološki vrlo različite izvedbe upravljanja virtualnim uređajem pomoći gesti ruke. Jedna izvedba je pomoću već postojeće tehnologije, uređaja Leap Motion, a druga pomoću rukavice za virtualnu stvarnost razvijene na Fakultetu elektrotehnike i računarstva.

Predstavljene su tehnologije koje su se koristile te su definirani podaci potrebni za prikaz ruke u virtualnom svijetu te način na koji se oni dobivaju i dohvaćaju iz dviju izvedbi.

Razrađena su dva algoritma za prepoznavanje dviju vrsti gesti ruke: geste prstiju te geste pokreta. Za geste prstiju je korištena neuronska mreža a za geste pokreta algoritam $P.

Opisane su sve tehničke poteškoće suočene pri pretvorbi i prikazu podataka iz različitih tehnologija u alatu Unity poput različitih koordinatnih sustava senzora rukavice i alata Unity, nedostaci u implementaciji programskog okruženja Mono itd.

Rad je bio zanimljiv jer ima poveznica s mnogo znanstvenih i inženjerskih disciplina: automatikom, elektronikom i računarskom znanošću. Posebno je zanimljiv bio rad s rukavicom za virtualnu stvarnost jer se radi o uređaju koji je napravljen skoro pa "od nule" koji nije već gotovi komercijalni proizvod.

Na kraju se pokazalo da su rezultati zadovoljavajući za obje izvedbe te da ovaj način upravljanja virtualnim svijetom ima potencijala postati popularniji, iako vjerojatno neće u skorije vrijeme zamijeniti tipkovnicu i miš.

4. Literatura

1. Križan, Petra. Izrada rukavice za virtualnu stvarnost. Diplomski rad. Fakultet Elektrotehnike i Računarstva, Zagreb, 2015.

2. Radu-Daniel Vatavu, Lisa Anthony, Jacob O. Wobbrock, 28.8.2012, icmi-12.pdf, *Gestures as Point Clouds: A $P Recognizer for User Interface Prototypes*, http://faculty.washington.edu/wobbrock/pubs/icmi-12.pdf, 20.5.2015

3. Radu-Daniel Vatavu, Lisa Anthony, Jacob O. Wobbrock, 30.10.2012, pdollar.pdf, *$P Point-Cloud Gesture Recognizer Pseudocode*, https://depts.washington.edu/aimgroup/proj/dollar/pdollar.pdf, 21.5.2015

Upravljanje gestama virtualnim uređajem MicroSoot

Sažetak

U radu se implementirao način upravljanja virtualnim uređajem Micro Soot pomoću gesti ruke. Korištene se dvije različite izvedbe prikaza virtualne ruke na računalu, uređaj Leap Motion te rukavica za virtualnu stvarnost. Implementirana su dva algoritma za dva različita tipa gesti ruke, jedan je izveden pomoći neuronske mreže a drugi je algoritam $P. Napravljena je usporedba dvije korištene izvedbe.

Ključne riječi: Upravljanje gestama, geste, LeapMotion, Micro Soot, virtualna stvarnost

Controlling the virtual Micro Soot device with gestures

Abstract

In this paper we have implemented a way to controll the virtual Micro Soot device with hand gestures. Two different technologies for representing the human hand in virtual space are used, the Leap Motion device and the virtual reality glove. Two algorithms for two types of gestures were implemented, one based on neural networks and the other is the $P algorithm. A comparison between the two used technologies was made.

Keywords: Gesture, Gesture recognition, Virtual reality glove, Micro Soot, LeapMotion