VEARM
En hjälpande hand

Saša Kojić

2015

LUND
UNIVERSITY

Examensarbete i

Biomedicinsk Teknik

Institutionen för Biomedicinsk Teknik
Lunds Tekniska Högskola

Handledare: Christian Antfolk

Abstract

During this project a virtual human model was created and placed in a virtual room with a surrounding environment. These models were created in the 3D modeling software Autodesk 3ds Max, in combination with the digital sculpting and painting software Autodesk Mudbox. The chosen engine to run the simulation was the game engine Unity 5 with the reason to provide an increased realism over classic simulation programs such as Matlab and LabVIEW. The surrounding environment was also created in this engine.

To control the human model sensors can be directly implemented into the game engine, which was done with the sensor system Myo from Thalmic. An alternative is by acquiring the sensors data in Matlab and sending the corresponding rotation data to Unity, which was done with the sensor system CyberGlove I.

In order to send information between Matlab and Unity a TCP/IP link was implemented, where Matlab acts as the server while Unity acts as a client who connects to the Matlab server. This was done locally on the same computer, but it is also possible to control the model from an external system.

The end result was a well-functioning human model in which the hands and legs look realistic in motion. The limiting factor of how well the model follows the real motion is how well the sensors can detect these.

Some problems where noted when the elbows were rotated in several directions simultaneously, which could have been prevented by a more realistic set of bones. This would however lead to a higher performance demand on the running system and it would increase the complexity of the tool. Movements in the spine and shoulders suffer from the same effect, but these parts are not visible during the simulation. A greater possibility to customize the human model would also create a better experience for the user, where gender, skin tone, height and build could be variables. This tool only has one human model of a Caucasian male included.

The implemented applications provide the ability for prosthetics training and mirror therapy, but with minor modifications the tool can be customized to fit most applications. The biggest limiting factor in this process would be the developer's imagination.

Tags

Virtual Reality Rehabilitation, Virtual Environment Rehabilitation, Prosthesis Training, Prosthesis R&D, Stroke Rehabilitation, Mirror Therapy, Serious Games

Sammanfattning

I detta arbete skapades en virtuell människomodell som placerades i ett virtuellt rum med omgivande miljö. Dessa modeller skapades i 3D-modelleringsprogrammet Autodesk 3ds Max i kombination med det digitala skulpterings- och målningsprogrammet Autodesk Mudbox. Motorn som valdes för att driva simuleringen var Unity 5. Det är en spelmotor som valdes för att ge en ökad realism över klassiska simuleringsprogram såsom Matlab och LabVIEW. Det var i denna spelmotor som den omgivande miljön skapades.

För att styra människomodellen kan antingen sensorer implementeras direkt i spelmotorn, vilket gjordes med sensoruppsättningen Myo från Thalmic, eller från rotationsinformation sänd från Matlab, vilket gjordes med sensoruppsättningen CyberGlove I.

För att kunna skicka information mellan Matlab och Unity implementerades en TCP/IP-länk, där Matlab agerar server medan Unity agerar klient och kopplar upp sig mot Matlab-servern. Detta skedde lokalt på samma dator men styrningen hade även kunnat ske från ett externt system.

Slutresultatet blev en väl fungerande modell där händer och ben ser realistiska ut i rörelse. Den begränsande faktorn för hur väl människomodellen följer de verkliga rörelserna är hur väl sensoruppsättningen detekterar dessa.

Vissa problem uppstod vid armbågarna då underarmen roterades runt flera axlar samtidigt vilket hade kunnat förbättras med en bättre modellering av de styrande benen. Detta hade dock inneburit en högre prestandabelastning och en ökad styrningskomplexitet. Rörelser i ryggrad och axlar drabbas av samma effekt men detta syns inte under simuleringen. Större möjlighet till att skräddarsy människomodellen hade även kunnat implementeras där kön, hudton, höjd och kroppsbyggnad varit variabler. Detta verktyg stöder endast en människomodell av en kaukasisk man.

De användningsområden som implementerats ger möjlighet till protesträning och spegelterapi, men med mindre modifikationer kan användningsområdena utökas mycket. Den största begränsande faktorn är utvecklarens fantasi.

Förord

Detta examensarbete är gjort som ett avslut av civilingenjörsutbildningen teknisk fysik på Lunds universitet. Målet med arbetet har varit att skapa ett verktyg där användaren kan styra en människomodell med hjälp av de sensoruppsättningar som terapeuten eller utvecklare vill använda sig av. Fokus lades på att hålla verktyget så generellt som möjligt för att tillåta styrning av alla kroppsdelar. Arbetet utfördes på Lunds tekniska högskolas institution för biomedicinsk teknik. Arbetet utfördes under vårterminen 2015 och motsvarar 30 högskolepoäng.

Innehållsförteckning

Introduktion

Bakgrund

I dagsläget är utvecklingsprocessen av proteser lång och dyr vilket leder till en dyr slutprodukt. Då en del kalibreringar av den färdiga produkten ofta behövs, till exempel för att avgöra vilka värden på sensorerna som motsvarar en knuten näve, hade en virtuell miljö där man kunnat utföra dessa kalibreringar underlättat för både patient och läkare. Patienten hade då även kunnat testa olika sensoruppsättningar och träna på att styra protesen, vilket har visat sig fungera väl i Virtual Reality (VR) då dessa kunskaper kan överföras till verkligheten (Resnik, Etter, Lieberman Klinger, & Kambe, 2011). Sjukhusen hade även kunnat ha ett flertal uppställningar med datorer och sensorer till en låg kostnad vilket inneburit att flera patienter kunnat träna samtidigt med en gemensam läkare/terapeut som lett träningen. En lägre kostnad per träningstillfälle leder till en större mängd träningstillfällen eftersom den begränsande faktorn för antalet träningstillfällen i dagsläget ofta är sjukhusets budget per patient. Möjligheten till att testa olika mätuppsättningar hade även förenklat specialanpassningar som patienten kunnat ha ett större inflytande över (Simmons, Arthanat, & Macri, 2014) (Rizzo & Kim, 2005).

Då proteser med känselåterkoppling är i forskningsstadiet hade en träningsplattform där patienten kunnat utforska denna funktion och kalibrera känselåterkopplingens styrka varit fördelaktig. Ett projekt inom proteser med känselåterkoppling är projektet SmartHand som utfördes på Lunds universitet (Sebelius, 2010).

Fantomsmärtor är ett annat problem som patienter i behov av proteser kan lida av. Detta är smärtor som patienten upplever i en kroppsdel som denne inte längre har kvar. De kan uppstå om patienten till exempel varit med om en olycka och förlorat en hand. Fantomsmärtorna upplevs då där den förlorade handen skulle ha suttit (Mickelsson, 2004). En vanlig åkomma är att fantomsmärtan sitter i den saknade handen som upplevs vara krampartat hopknuten. I dagsläget kan detta behandlas genom att patienten lägger in stumpen bakom en spegel som reflekterar den andra handen vilket visas i Figur 5. Patienten utför då olika rörelser som innefattar att knyta ihop sin befintliga hand och sedan sakta slappna av i den. Detta kan med tiden helt bota smärtorna, men behandlingen är inte alltid framgångsrik och ställer krav på att patienten har kvar en hand (Ezendam, Bongers, & Jannink,

2009). En virtuell miljö som styrs med hjälp av sensorer hade gjort denna behandling möjlig oberoende av kroppsdel och rehabiliteringen hade även kunnat göras roligare. Även realismen hade ökat då patienten medvetet försökt röra sin fantomkroppsdel vilket återspeglats i att modellens arm hade förflyttat sig på samma sätt. Detta leder till att hjärnan lättare accepterar den artificiella kroppsdelen som sin egen (Perez-Marcos, Sanchez-Vives, & Slater, 2011). Under den konventionella spegelterapin rör patienten sin motsatta kroppsdel vilket resulterar i samma rörelse i den artificiella som visas i spegeln, vilket inte är en lika övertygande upplevelse. En stor fördel är att kravet på att ha en fungerande motsatt kroppsdel inte längre funnits.

Ett liknande projekt som tog fram en virtuell arm som styrdes av sensorer gjordes på Chalmers. Detta utfördes på en patient som efter 48 år inte botats av den traditionella spegelterapin. Med den virtuella armen blev däremot resultatet lyckat under den tid terapin fortlöpte och styrningen tillämpades även i ett bilspel vilket uppskattades av patienten (Ortiz-Catalan, Sander, Kristoffersen, Håkansson, & Brånemark, 2014).

Syfte

Syftet med detta arbete är att ta fram ett verktyg som går att använda till en mängd olika sensoruppsättningar och ändamål. De två primära användningsområdena är protesträning och behandling av fantomsmärtor. Möjligheten att göra de olika rehabiliteringsprogrammen roligare är även en viktig komponent i verktyget.

För att hålla verktyget generellt krävs en människomodell där alla kroppsdelar går att styra. Dessa kroppsdelar ska även vara lätta att styra, och då Matlab är väl etablerat i den akademiska världen kommer det att användas för att ta emot signaler, behandla dem och slutligen styra modellen. Versionen av Matlab som används är R2015a. För att göra modellen och dess omgivande miljö realistisk används spelmotorn Unity 5 under simuleringarna.

De två kompletta sensoruppsättningar CyberGlove I från CyberGlove systems, som visas i Figur 1, och Myo från Thalmic, som visas i Figur 2, implementeras i verktyget. Myo:n används för att mäta underarmens position och rotation och CyberGlove:n används för att mäta handen och fingrarnas positioner.

Figur 1: Figuren visar CyberGlove version ett, samma version som implementerats i verktyget.

Figur 2: Bilden visar en Myo från utvecklaren Thalmic.

Möjligheten till känselåterkoppling är implementerad och en enklare variant i form av olika vibrationslängder av Myo:n beroende på vilket objekt modellen rör vid är inkluderad.

Projektet delades upp i följande tre faser:

- Skapandet och riggningen av människomodellen i 3ds Max 2015 och Mudbox 2015 från Autodesk.
- Skapandet av spelvärlden i Unity som människomodellen kommer att befinna sig i.
- Kommunikationslänken och styrningen mellan Matlab och Unity.

Avgränsningar

De områden som implementeras är styrningen av underarmen och handen. Denna rörelse kan kopplas till modellens vänstra och högra arm eller endast kopplas till modellens högra arm.

Människomodellen begränsas till en modell av en kaukasisk man då grundmodellen för en man finns i Mudbox.

De sensoruppsättningar som implementeras är CyberGlove I med 18 sensorer och Myo. CyberGlove:n avläses via Matlab som sedan styr människomodellen via en TCP/IP-länk medan Myo:n är direkt kopplad mot Unity för att visa de olika styrningsmetoder som finns i verktyget.

Känselåterkopplingen är begränsad till två situationer: att den högra handen antingen rör vid en citron eller en lejonstatyett placerade på bordet framför människomodellen. Myo:n vibrerar i fallet av citronen med en medellång vibration och i fallet med statyetten med en lång vibration för att symbolisera hur hårt objektets yta är.

Teori

Proteser

Marknaden för proteser är i dagsläget väldigt stor och i USA finns det mellan 400 000 och 1 000 000 personer som använder proteser (UXL Encyclopedia of Science, 2002). För en benprotes ligger det amerikanska priset mellan 10 000 och 70 000 dollar beroende på komplexitet, vilket motsvarar ca 84 000 till 590 000 svenska kronor om en dollar motsvarar 8,4 svenska kronor (enligt kursen den 3/6 2015). En undersökning av den totala protesrelaterade livstidskostnaden för amerikanska krigsveteraner med enbart ett ben beräknades ligga på över 1,4 miljoner dollar per person, vilket motsvarar ca 12 miljoner svenska kronor (CostHelper Prosthetic Leg, 2014). I Sverige ligger antalet benamputationer på cirka 2 000 - 3 000 personer per år och majoriteten väljer att använda protes (Akademiska sjukhuset, 2013). För en armprotes ligger det amerikanska priset mellan 10 000 till 100 000 dollar, motsvarande ca 84 000 till 840 000 svenska kronor, för en aktiv funktionell protes medan en passiv kan kosta mindre än 5 000 dollar, motsvarande ca 42 000 svenska kronor. Den totala protesrelaterade livstidskostnaden beräknad på amerikanska krigsveteraner ligger på ca 823 000 dollar per person, motsvarande 6,9 miljoner svenska kronor (CostHelper Prosthetic Arm, 2014). Antalet armamputationer i Sverige ligger på runt 50 per år (Personskadeförbundet RTP). I Sverige bekostas proteser av landstinget (Personskadeförbundet RTP, u.d.).

Proteser finns huvudsakligen i två olika utföranden: interna proteser som opereras in i patienten, t.ex. höftledsproteser, och externa som patienten själv kan fästa och ta av. De externa proteserna kan även delas upp i aktiva och passiva proteser, där de aktiva går att styra i olika hög grad och de passiva antingen används av estetiska skäl eller för att kunna stödja sig på och kila fast objekt med. Två klassiska exempel på passiva externa proteser man använde förr i tiden är en krok som ersatte en förlorad hand och ett träben som ersatte ett förlorat ben.

Detta arbete kommer att fokusera på de externa aktiva proteserna då det främst är dessa som använder sig av sensorer för styrning och återkoppling. När termen protes används i rapporten menas hädanefter en aktiv extern protes. I Figur 3 visas den avancerade handprotesen bebionic3 och i Figur 4 visas en benprotes. Båda dessa externa proteser är aktiva och går att styra.

Figur 3: Bilden visar den aktiva handprotesen bebionic3.

Figur 4: Bilden visar en aktiv benprotes.

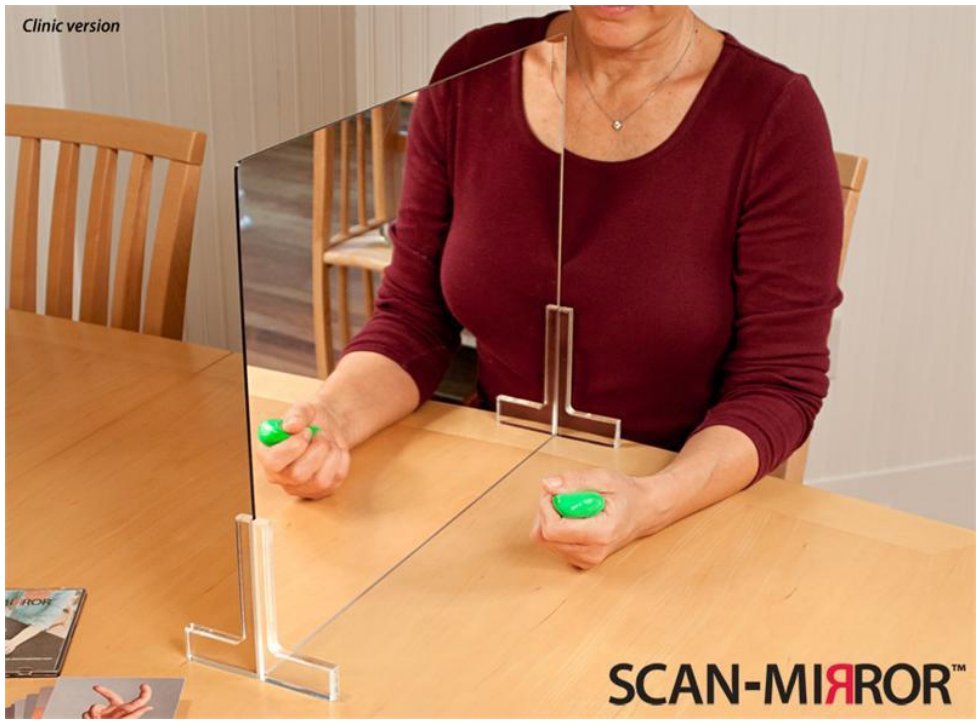
När patienten valt ut en passande protes krävs det träning för att kunna optimera användningen. Träningen bör därför påbörjas direkt vilket ofta inte är möjligt då det kan dröja innan protesen levereras till sjukhuset. Detta innebär att det kan dröja innan patienten får möjlighet att lära sig använda protesen vilket leder till att det tar längre tid innan protesen går att använda i vardagslivet. Tiden från det att en protes valts ut till dess att patienten kan börja använda den i sin vardag bör vara den kortast möjliga vilket ett virtuellt träningsprogram hade bidragit till. En studie som undersökte denna metod lät en patient träna på att använda sin protes i en virtuell miljö (Virtual Environment – VE), och träningen ledde till att patienten kunde använda sin protes på ett bättre sätt (Resnik, Etter, Lieberman Klinger, & Kambe, 2011).

Det slutgiltiga målet med proteser är att kunna ersätta den förlorade kroppsdelen med ett lika väl eller till och med bättre fungerande alternativ. För att uppnå detta resultat underlättar en kortare och billigare utvecklingsperiod, vilket simuleringen i detta arbete förväntas kunna bidra till.

Fantomsmärtor

Fantomsmärtor är smärtor som patienten upplever sig ha i en bortamputerad kroppsdel. Dessa smärtor brukar främst upplevas i armar eller ben, men även i interna organ. Av de patienter som amputeras upplever upp till 80 % fantomsmärtor (Mickelsson, 2004). Anledningen till att smärtan uppstår kan bero på att delar av nerverna till den amputerade kroppsdelen finns kvar och skickar signaler till det centrala nervsystemet.

En behandling som reducerar denna smärta och i bästa fall även botar den är spegelterapi. Denna terapi utförs som tidigare nämnt genom att patienten placerar sin stump bakom en spegel som reflekterar den friska kroppsdelen. Av denna anledning lämpar sig terapin enbart för de yttre extremiteterna. Detta resulterar i en illusion av att den amputerade kroppsdelen finns kvar och en exempeluppställning av detta visas i Figur 5. Genom att sedan fokusera på den reflekterade kroppsdelen under rörelse och avslappning reduceras fantomsmärtan. Anledningen till att detta fungerar är att hjärnan uppfattar den artificiella kroppsdelen som kroppens egen om position och rörelse stämmer överens med den som förväntas av hjärnan (Perez-Marcos, Sanchez-Vives, & Slater, 2011).



Figur 5: Bilden visar en exempeluppsättning för spegelterapi.

Den största smärtreducerande effekten upplevs under själva behandlingen. Det krävs flera behandlingstillfällen för att slutligen förhoppningsvis bota smärtan helt. Tyvärr brukar den återkomma efter en viss tid. Terapin har enligt patienter och terapeuter en positiv effekt, men en omfattande vetenskaplig studie som bekräftar detta har inte utförts (Lowe, 2015) (Ezendam, Bongers, & Jannink, 2009) (MacLachlan, McDonald, & Waloch, 2004). Denna behandling är inte bara effektiv vid fantomsmärtor utan har även påvisats ha effekt vid brännskador. En studie undersökte en patient med omfattande brännskador med syfte att reducera dennes smärta. Patienten fick då använda en VR-hjälm från Oculus Rift som visade en virtuell representation av patienten när armarna var friska. Under studien upplevde patienten sig känna mindre smärta från sina armar (Hoffman, o.a., 2014).

Terapi och behandlingar i Virtual Reality

Ett område som börjar växa inom medicinsk terapi är behandlingar utförda i Virtual Reality. Dessa blir mer och mer realistiska i takt med att nöjesindustrin utvecklar spel och filmer mot en verkligare återgivning, vilket i sin tur har blivit möjligt i takt med att beräkningskraften hos datorer ökar.

Historiskt sett har militärer, parallellt med nöjesindustrin, varit drivande inom VR och de har använt sig av VR i olika hög grad under soldatträning då scenarion har varit svåra att återskapa i verkligheten (Laff, 2007). Dessutom innebär träningen i VR en låg risk för användaren (Rizzo & Kim, 2005). Användandet började då datorer blev kraftfulla nog att ersätta de brädspel som tidigare använts för att träna på taktiker. I början av denna konvertering var det främst brädspelen som direkt återskapades för att underlätta beräkningar och manipulation för användarna, men detta utvecklades till hela världar i takt med att grafikmöjligheterna förbättrades. För att visa hur långt tekniken inom spelindustrin har kommit visas nedan i Figur 6 en skärmbild från det kommande spelet ”Tom Clancy's The Division”. Tekniken som används inom spelindustrin kan även användas inom rehabilitering då båda sker i realtid.

Figur 6: En skärmbild från spelet "Tom Clancy's The Division".

I dagsläget finns det många simulatorer som efterliknar verkligheten i en hög grad. Användaren ges därmed förutsättningar till förberedelse inför att utföra uppgiften i verkligheten. Dessa simulatorer och applikationer som

visuellt liknar spel, men vars syfte skiljer sig åt, och istället för att endast underhålla användaren även fokuserar på att lära eller träna användaren inom ett område kallas för ”Serious Games”. Exempel på dessa kan vara flygplanssimulatorer, intervjusimulatorer och medicinsk träning inför operationer (Smith, 2010).

När Serious Games började nå den medicinska världen, som har ett högre krav på bevisade effekter än många andra industrier, ökade mängden akademiska studier inom området. De undersökningar som gjorts är dock relativt spridda och hade behövt bli mer koordinerade sinsemellan för att stärka den teoretiska grunden, varför flera större och mer omfattande studier krävs för att etablera VR-rehabilitering inom den medicinska världen. Då patienters förutsättningar varierar gällande faktorer som ålder, rörelseförmåga och teknisk kunskap behöver även ett större fokus läggas på patienterna som deltar i undersökningarna (Ortiz-Catalan, o.a., 2014).

Att kunna individualisera rehabiliteringsövningarna efter patientens personliga förutsättningar är något som förbättrar rehabiliteringens resultat, vilket är ett område som VR underlättar avsevärt. Rehabiliteringsprogrammet kan utformas till att ha flera olika nivåer som kan bestämmas utifrån patientens prestation. Dessa nivåer kan då ge variation och ökad komplexitet. Patienten kan dessutom få en känsla av utveckling och tillfredsställelse av att klara av uppgifterna. Möjligheten att koppla styrningen mot kommersiella spel kan även leda till en roligare rehabilitering, speciellt för barn. Fokus kan då skifta från en tråkig rehabiliteringsövning till ett spel som patienten tycker är roligt, vilket ökar motivationen (Ortiz-Catalan, o.a., 2014) (Rizzo & Kim, 2005) (Wrzesien & Raya, 2010). Patientens inställning till sin möjlighet att förbättras påverkar även resultatet. En patient som tror på möjligheten till förbättring söker sig till större utmaningar vilket är centralt för självinlärningen och leder till en mer lyckad rehabilitering (Lee, Heeter, Magerko, & Medler, 2012). Serious Games har även visat sig leda till en bättre inlärning än vad konventionell träning gjort, bland annat vid hjärt-lungräddning och brädspel (Knighta, o.a., 2010) (Buttussia, o.a., 2013).

Ett stort problem med rehabilitering är att det stora kravet på antalet repetitioner av rehabiliteringsövningar leder till att patienten behöver utföra dem utanför sjukhuset, vilket ofta leder till ett sämre genomförande av övningarna och därmed ett sämre resultat. Detta beror även på att

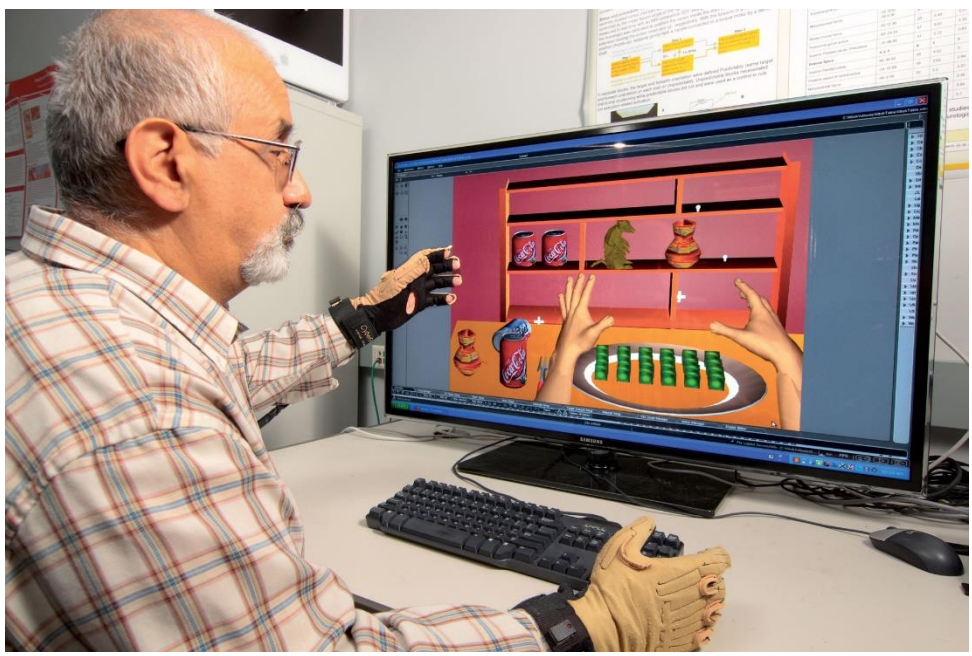
övningarna snabbt blir enformiga och patienten tappar motivation. För att öka patientens motivation kan Serious Games användas för att lägga till möjligheten att tävla med eller mot andra patienter. En studie visade att olika patienter föredrog olika spelsätt, vilket ytterligare stärker vikten av att fokusera mer på patienternas individuella förutsättningar (Novak, Nagle, Keller, & Riener, 2014). Ett annat problem är att patienten inte alltid vet om övningen utförts tillräckligt väl, vilket visar på återkopplingens betydelse. Även detta är något VR lämpar sig väl för, där återkoppling kan ges och erhållas konsekvent. Rehabiliteringsprogram blir även lätta att återskapa och kan dupliceras obegränsat (Rizzo & Kim, 2005).

Då dessa rehabiliteringsövningar är viktiga för ett lyckat rehabiliteringsresultat hade övningar som motiverade patienten till att anstränga sig mer och som gör att patienten vet om övningen utförts rätt ökat chanserna för en lyckad rehabilitering. I takt med att patienten blir bättre på att utföra sina övningar bör svårighetsgraden öka, vilket är lättare att tillämpa i VR än i verkligheten. Att programmet dessutom kan spara statistik, såsom antalet korrekt utförda övningar, gör det möjligt för terapeuten att förändra och diskutera upplägget av övningarna med patienten. I dagsläget är det enda terapeuten får reda på angående hur väl övningarna genomförts det patienten berättar, vilket inte alltid stämmer överens med verkligheten (Ortiz-Catalan, o.a., 2014) (Rizzo & Kim, 2005).

Det har även genomförts undersökningar på hur väl kunskaper och erfarenheter som användaren lärt sig i VR kan översättas till verkligheten. Motorisk inlärning visade sig vara möjlig och rörelser som patienten lärt sig i VR kunde användas i motsvarande uppgifter i verkligheten. I vissa fall kunde rörelserna generaliseras till andra uppgifter som patienten inte upplevt i VR-träningen (Ortiz-Catalan, o.a., 2014).

Det har även utförts många studier på VR-rehabilitering för strokepatienter där fokus legat på att den nedsatta rörelseförmågan i den påverkade sidans hand. Rehabiliteringens mål är att hjälpa patienten träna upp sin rörelseförmåga till högsta möjliga grad. Denna sorts rehabilitering i VR har fått positiva resultat där greppstyrka och rörelseförmåga har ökat, och ett exempel på en rehabiliteringsmiljö visas i Figur 7 (Ortiz-Catalan, o.a., 2014) (Merians, o.a., 2011) (Jack, o.a., 2001) (Mónica da Silva Cameirão, Duarte, & Verschure, 2011) (Burdea, Cioi, Martin, Fensterheim, &

Holenski, 2010). En jämförelse mellan klassisk rehabilitering och rehabilitering utförd i VR gav resultatet att VR-rehabiliteringen gav signifikant bättre resultat (Turolla, o.a., 2013). En annan studie undersökte om Serious Games med mål att träna upp rörelsemöjligheterna i axel, armbåge och underarm för patienter med hjärnskador orsakade av stroke eller olyckor hade en positiv effekt. Detta visade sig vara fallet. Priset för rehabiliteringen visade sig även vara lägre då VR användes (Simmons, Arthanat, & Macri, 2014). Det visade sig även vara viktigt att övningarna utformades så att patienten blev tvungen att utmana sin rörelseförmåga, då en studie som bland annat hade en virtuell version av air hockey visade att många patienter valde att spela defensivt eftersom de då inte behövde röra sina armar lika mycket (Acosta, Dewald, & Dewald, 2012).



Figur 7: Bilden visar en virtuell träningsmiljö för strokepatienter där CyberGloves används.

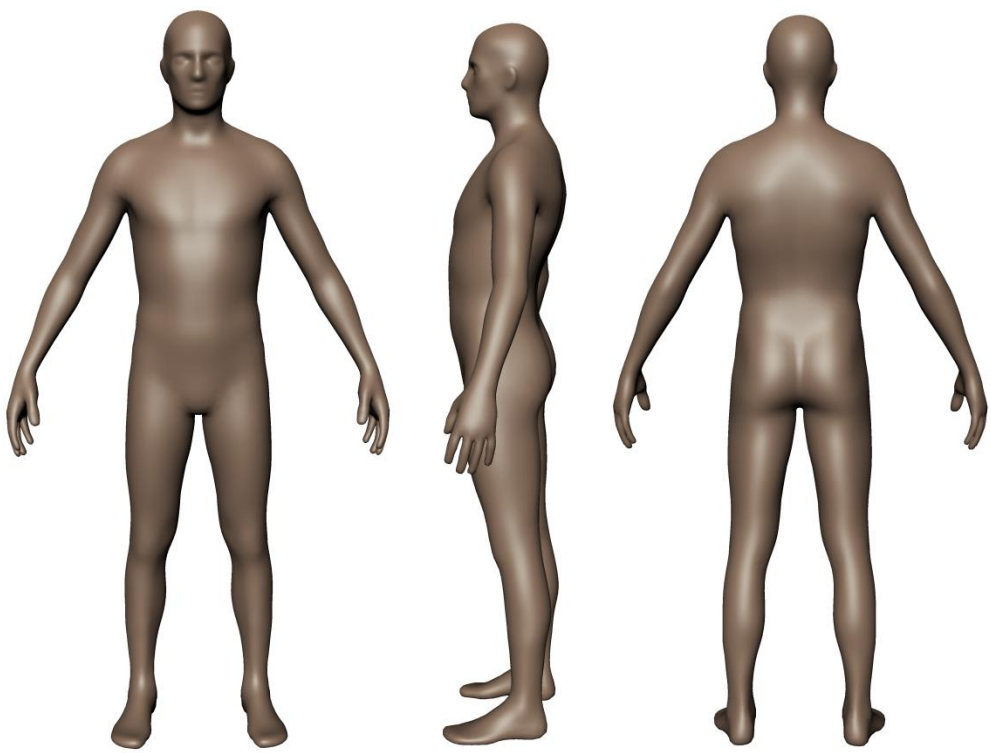
Ett par nackdelar med VR inom rehabilitering är att det fortfarande är oetablerat, att tekniken inte har en standardisering som underlättar inkopplingsenheter och hårdvarukrav, saknar kompabilitet mellan olika plattformar och att ett fåtal patienter kan uppleva illamående. Ett standardiserat ramverk för VR inom medicin hade underlättat spridningen avsevärt, men att detta saknas i dagsläget är inte oväntat då området fortfarande är nytt. Bieffekter som illamående antas minska i takt med att grafikmöjligheterna ökar (Ortiz-Catalan, o.a., 2014) (Rizzo & Kim, 2005).

Metod

Människomodell

Det första steget i arbetet var att skapa en realistisk människomodell. För att lyckas med detta krävdes en programvara som var kraftfull nog att göra bra modeller till ett rimligt pris. De två programvaror som jämfördes var Autodesk Maya, som är populärt inom filmindustrin, och Autodesk 3ds Max som är populärt inom spelindustrin. Autodesk låter dessutom studenter använda deras programvaror gratis. Båda programmen ansågs kunna utföra samma uppgifter med ett liknande resultat, där Maya verkade ge ett något bättre resultat till priset av en svårare inlärning. Då 3ds Max redan använts i en annan kurs och då det ansågs vara något lättare utan att märkbart försämra slutresultatet skapades modellen i detta.

För att lägga till detaljer och måla modellen valdes Autodesk Mudbox då det är avsett att användas i kombination med 3ds Max och Maya. I Mudbox ingår det en människomodell av en man som valdes som bas för projektets modell. Utseendet på denna visas i Figur 8 nedan.



Figur 8: Bilden visar grundmodellen av människan som finns tillgänglig i Autodesk Mudbox.

Denna grundmodell består av 8272 polygoner, där fler polygoner generellt ökar prestandakraven på datorn som ska rendera dem. Fler polygoner innebär dock en högre detaljrikedom, vilket resulterar i att antalet polygoner får balanseras för att ge en realistisk modell som inte kräver för mycket prestanda. Då antal polygoner endast är en av faktorerna på hur prestandakrävande det slutgiltiga programmet blir finns det inte några specifika riktlinjer på hur många polygoner en modell bör ha. Eftersom endast en komplex modell ska manipuleras i det slutgiltiga verktyget sattes kravet att modellen skulle vara uppbyggd av maximalt 20 000 polygoner, men målet var 10 000 polygoner.

Grundmodellen importerades direkt till 3ds Max där den omarbetades för att ha kläder och andra grova detaljer såsom muskler. Då ansiktet inte kommer att synas lades ingen tid på att förändra detta och fokus låg på resten av modellen. Modellen blev klädd med en t-shirt, jeans och ett par gymnastikskor. Anledningen till att dessa neutrala kläder valdes var att många skulle kunna känna att de hade kunnat göra samma klädval och därför lättare kunna relatera till modellen. Användaren antas även i de flesta fall ha armarna fria från kläder då olika sorters sensorer är påsatta, vilket var anledningen till att en kortärmad t-shirt valdes. Då grundmodellen föreställde en man förblev modellen vara detta. Användaren kommer dock endast att se armarna och i vissa användningsområden benen, vilket minskar vikten av modellens kön då modellens armar och ben inte skiljer sig mycket från en kvinnas. Resultatet av ommodelleringen visas i Figur 15, där renderingen är gjord i 3ds Max.

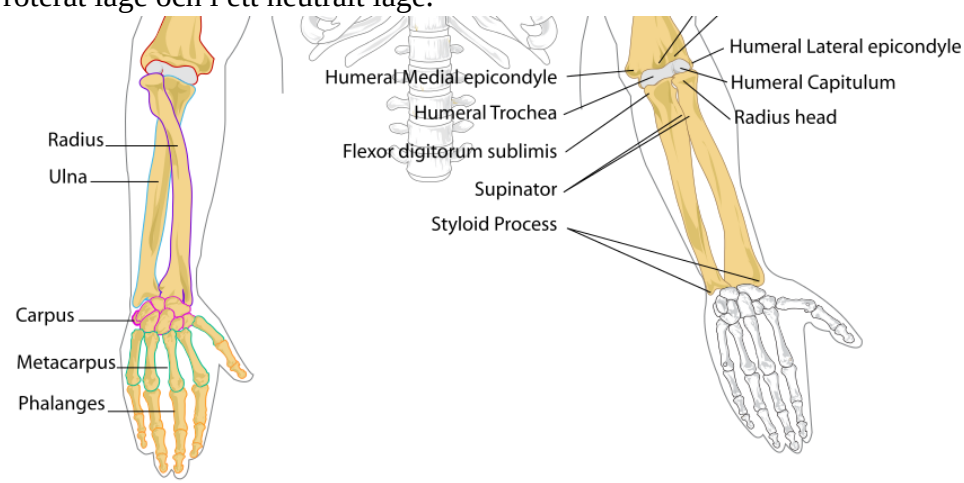
Modellen exporterades därefter till Mudbox för att lägga till detaljer och färger. För att kunna addera detaljer krävs dock ett större antal polygoner, vilket går att öka direkt i Mudbox. Modeller brukar därför ha ett stort antal polygoner då detaljarbete utförs i Mudbox, vilket sedan projiceras på en basmodell med lägre antal polygoner med hjälp av till exempel höjddata på polygonernas normaler.

De detaljer som lades till var främst konturer, muskler, leder och veck på armar och händer, och neutrala färger valdes. Modellens hudfärg valdes efter modellerarens egen. Resultatet av detta arbete visas i Figur 16.

De detaljer som lagts till överfördes inte helt som förväntat, vilket syntes tydligast på modellens händer. Detta hade kunnat optimeras och en ny basmodell med större antal polygoner runt områden med mer detaljer hade

skapas, men då denna process är väldigt tidskrävande hade det skiftat projektets fokus. I Figur 17 visas en närbild på modellens vänstra hand innan och efter detaljerna projicerats till en grundmodell med ökat antal polygoner jämfört med den första importerade modellen.

Det sista steget i modelleringen var att rigga modellen för att kunna röra den, vilket görs med modeller av ben som kopplas samman. Enligt konventionen är ben på vänster sida av modellen blåa, röda i mitten och gröna på höger sida. Dessa ben renderas aldrig, och syns därmed endast vid modelleringen. Figur 10 och Figur 11 visar hur dessa ben ser ut och hur de är placerade i modellen. För att hålla prestandakraven nere modelleras varje kroppsdel som ska gå att röra med ett ben, vilket blir mest märkbart då underarmen rör sig. En mänsklig underarm har två ben, armbågsbenet (latin: ulna) och strålbenet (latin: radius), som tillsammans ökar rotationsmöjligheten av underarmen. I Figur 9 visas dessa två ben i ett roterat läge och i ett neutralt läge.

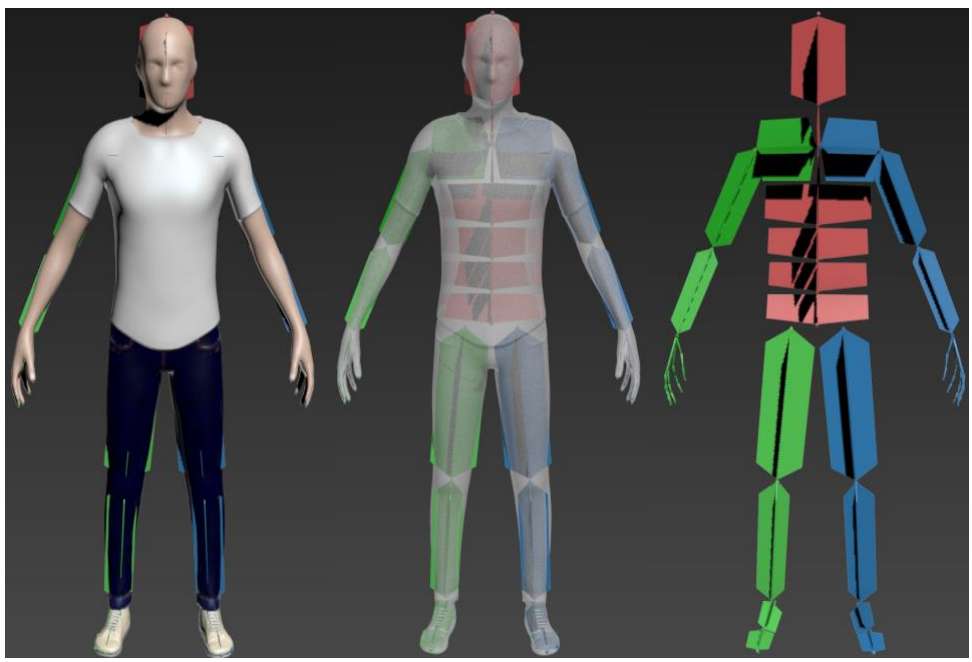


Figur 9: Bilden visar de ben som finns i underarmen med namnen på latin.

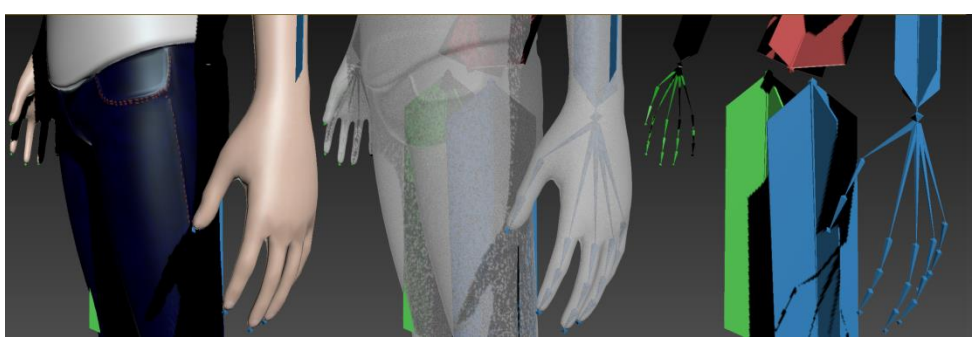
När benen placerats ut i modellen behöver de kopplas mot modellens yta. Detta görs i en process där varje polygon kopplas med en viss fördelning mot de olika benen. En polygon som är placerad i mitten av ett ben får alltså 100 % av sin förflyttning från detta ben, medan en polygon som är placerad mellan två ben kan få lika stora delar från båda benen.

Arbetsflödet för modelleringen visas i Figur 12. Resultatet av varje steg och vilket program det gjordes i är utmarkerat.

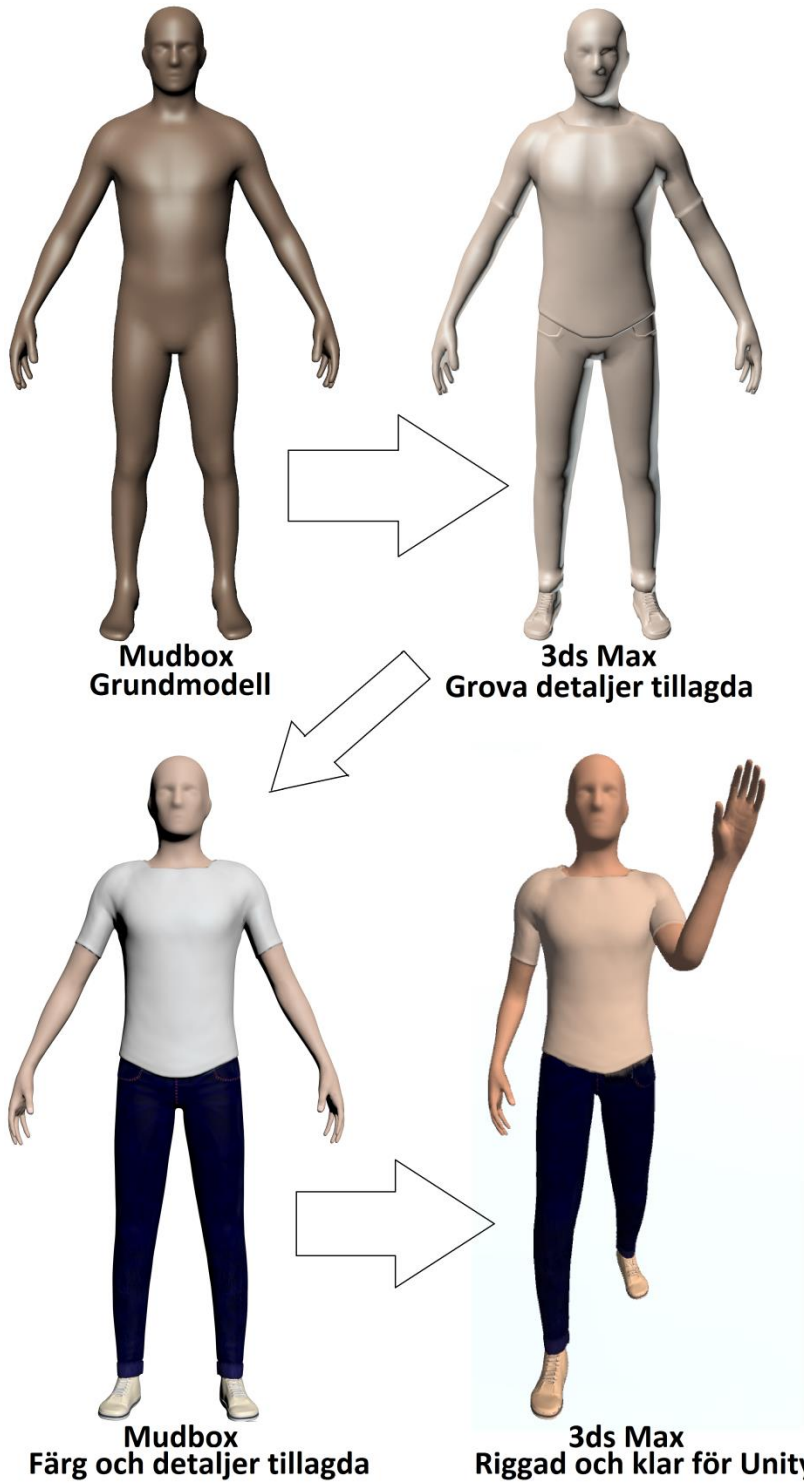
Den slutgiltiga modellen hade kunnat optimeras polygonmässigt och göras mer realistisk om detta varit projektets enskilda ändamål och mer tid därmed kunnat läggas på det. Antalet polygoner blev slutligen 17 376 då en projicering av detaljerna på den första basmodellen inte blev bra nog. Då erfarenhet och kunskap inom detta inte fanns sedan tidigare var målet att göra en modell som ansågs vara realistisk nog och som gick att använda i projektet utan att ställa alltför höga krav på prestandan.



Figur 10: Bilden visar de ben som styr modellen.



Figur 11: Bilden visar en närbild på de ben som styr handen.



Figur 12: Bilden visar arbetsflödet för modelleringen.

Spelvärld

Nästa steg var att skapa en värd att placera människomodellen i. Först skapades ett rum i 3ds Max. Detta blev sedan inrett med ett antal modeller hämtade från TurboSquid vilka visas i Tabell 1.

Tabell 1: Tabellen visar skaparen av och namnen på de objekt som används för att inreda rummet.

Namn	**Skapare**	**Objekt**
Fireplace Boley	Kupfer	Eldstad
Lemon Whole	Softwavez	Citron
Dali Fazon F5	M600maxx	Högtalare
Contemporary Corner Sofa	Astudio bd	Soffa
Coffe Table	Wydler Studios	Bord
Crane sculpture	Ericrevollo	Skulptur av en trana
Slim Bedroom Lamp	Woodbyte	Golvlampa
Wunderlicht 61669-8 Maxwel	Oodindd	Taklampa

Ett antal texturer användes även för att ge rummet ett mer realistiskt intryck. Dessa redovisas i Tabell 2.

Tabell 2: Tabellen visar skaparen av och namnet på de texturer som använts i rummet.

Namn	**Skapare**	**Objekt**
Free Furniture Props	Vertex Studio	Golvtextur
99 seamless wallpapers pack	Seetvael	Väggtextur

När rummet modellerats klart exporterades det till Unity. Skärmbilder från det färdiga rummet, då citronen och lejonstatyetten är dolda, renderade i 3ds Max visas i Figur 18 och Figur 19.

För att skapa omgivningen användes Unitys ”Terrain editor”. Tanken med miljön var att den skulle vara lugn och stillsam för att få användaren att känna sig trygg och avslappnad. Motivet blev därför en äng med kullar. Framför rummets panoramafönster placerades en liten damm där vattnet sakta rör sig och en tårpil vars grenar hänger ut över vattnet. En skärmbild tagen inifrån rummet visas i Figur 20 och en skärmbild tagen strax utanför rummet visas i Figur 21. Dessa bilder finns på sida 27.

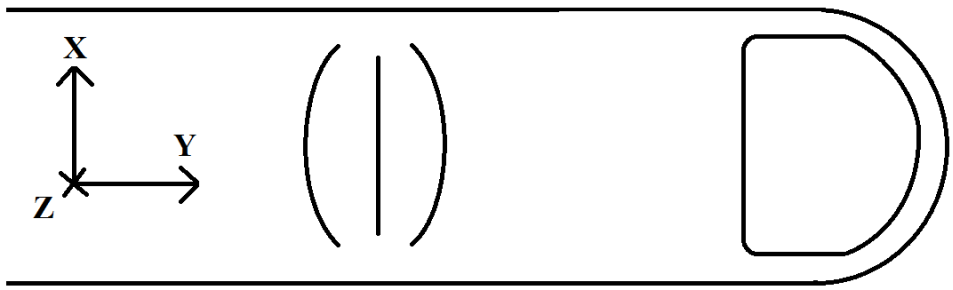
Kommunikationslänk och styrning

För att slutligen kunna styra människomodellen skrevs skript i Matlab och kod i Unity på språket C#. I Unity implementerades först Myo:n genom att utgå från den exempelkod som var inkluderat i det ”Software Development Kit” (SDK) som Thalmic gett ut. Koden byggdes vidare till att låta användaren styra höger underarm och spegla denna rörelse till den vänstra armen. Möjligheten att kunna återställa armarnas position till dess ursprungsläge lades till, och utförs då knappen ”R” trycks ner. Myo:n har funktionen att avläsa fem olika handgester, varav en knuten näve och spretande fingrar resulterar i att modellens hand knyts respektive öppnas vilket bekräftas med en kort vibration i Myo:n. Denna kod redovisas i Appendix 4 under kapitlet ”Myokod”.

Därefter lades två objekt att interagera med till, en citron och en lejonstatyett. Då användaren rör vid objekten med modellens högerhand vibrerar Myo:n i olika lång tid. En vidröring av citronen resulterar i en medellång vibration, medan en vidröring av lejonstatyetten resulterar i en lång vibration för att symbolisera hur hårt objektets yta är. Denna kod finns under kapitlet vibrationskod i Appendix 4.

Därefter utvecklades möjligheten att kommunicera med Matlab via en TCP/IP-länk. Denna kod vidareutvecklades från ett grundläggande kommunikationsexempel där ett meddelande kunde skickas via en TCP/IP-länk. Kodexemplet hämtades från bloggen ”GirlsCanCode” under posten ”Update: Unity and TCP/IP Socket Connections” från den förste februari 2015. Denna kod finns under kapitlen ”Socketkod” och ”TCPkod” i Appendix 4. Länken aktiveras då användaren har CyberGlove:n aktiverad och klickar på en knapp med texten ”Connect” uppe i vänstra hörnet av programmet.

Koden programmerades om till att ta emot en textsträng som formaterats på ett specifikt sätt. Denna textsträng innehåller rotationsdata i alla led för alla modellens ben i handen. Koordinatsystemet som används i strängen antas vara positivt orienterat där y-vektorn pekar utmed fingret och en positiv rotation i x-led leder till en böjning av fingret till en knuten näven då alla fingrar böjs positivt i x-led. Figur 13 visar detta koordinatsystem.



Figur 13: Bilden visar koordinatsystemet som används för mottagen rotationsdata i Unity.

Den mottagna informationen behöver sedan konverteras eftersom koordinatsystemen för modellens ben inte stämmer överens med det koordinatsystem som används i Matlab. Därefter utförs rotationerna.

En funktion för att kunna återställa modellens fingrar till dess initiala position lades även till, vilket aktiveras då knappen ”T” trycks ner. Fingrarnas position avläses då programmet startas, vilket innebär att användaren kan ändra denna referenskonfiguration genom att rotera dem i Unitys editor.

Slutligen lades kontrollkod till för att underlätta ändring av variabler, vilket redovisas under kapitlet ”Kontrollkod” i Appendix 4.

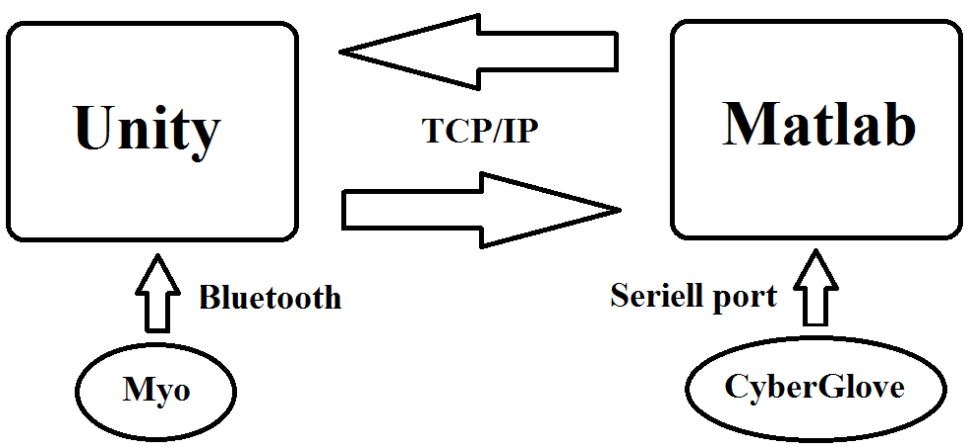
Nästa steg var att utveckla skriptet i Matlab. Först skapades ett skript för att avläsa CyberGlove:n. Skriptet redovisas i Appendix 3, där även underfunktioner finns med. En av underfunktionerna står för konverteringen av rotationsmatrisen, som har bredden tre och höjden 18, till en korrekt formaterad textsträng. Den andra underfunktionen används vid kalibreringen av handsken, som då detekterar om alla sensorer går att avläsa.

Skriptet inleds med en karta som visar modellens ben i handen och deras indexnumrering. Därefter bestämmer användaren hur mycket de olika benen i handen ska gå att röra, vilket anges i grader. Detta följs av en kalibrering där handsken läses av då användaren böjer handleden uppåt och utåt samtidigt som fingrarna spänns ut maximalt och bildar en öppen hand, vilket representerar max-värdena. Därefter registreras minimi-värdena då användaren ombeds knyta ihop handen, rotera handleden inåt och nedåt. Dessa värden skalas sedan om med hjälp av rörelseomfånget som angivits och resulterar i en konverteringsvektor. Det finns även en sparad kalibrering som användaren kan välja att använda, alternativt kan en

personlig kalibrering sparas undan för att ta bort kalibreringskravet vid varje session.

När detta är färdigt är handsken kalibrerad och skriptdelen som upprätthåller TCP/IP-länken kan aktiveras. I praktiken är detta en loop som, med en uppdateringsfrekvens som synkroniseras mot det angivna värdet i kontrollkoden i Unity, konstant körs och i varje iteration avläser sensorerna. Loopen avbryts då Unity skickar ett meddelande som innehåller ”Disconnect”. Detta meddelande blir skickat då användaren trycker på knappen ”Disconnect” som ersätter knappen ”Connect” då länken blivit upprättad. Denna kommunikation sker lokalt på datorn, men kan även ske via ett nätverk för att styra modellen på avstånd eller från andra maskiner och system med specialanpassad hårdvara.

Varje gång handsken blir avläst konverteras värdena till grader med hjälp av konverteringsmatrisen från kalibreringen. Därefter placeras denna information in på korrekt plats i rotationsmatrisen. Den rotationsinformation som skickas är skillnaden från den aktuella loopiterationens rotationsmatris till den föregående loopiterationens rotationsmatris. En systembild över kommunikationen visas i Figur 14.

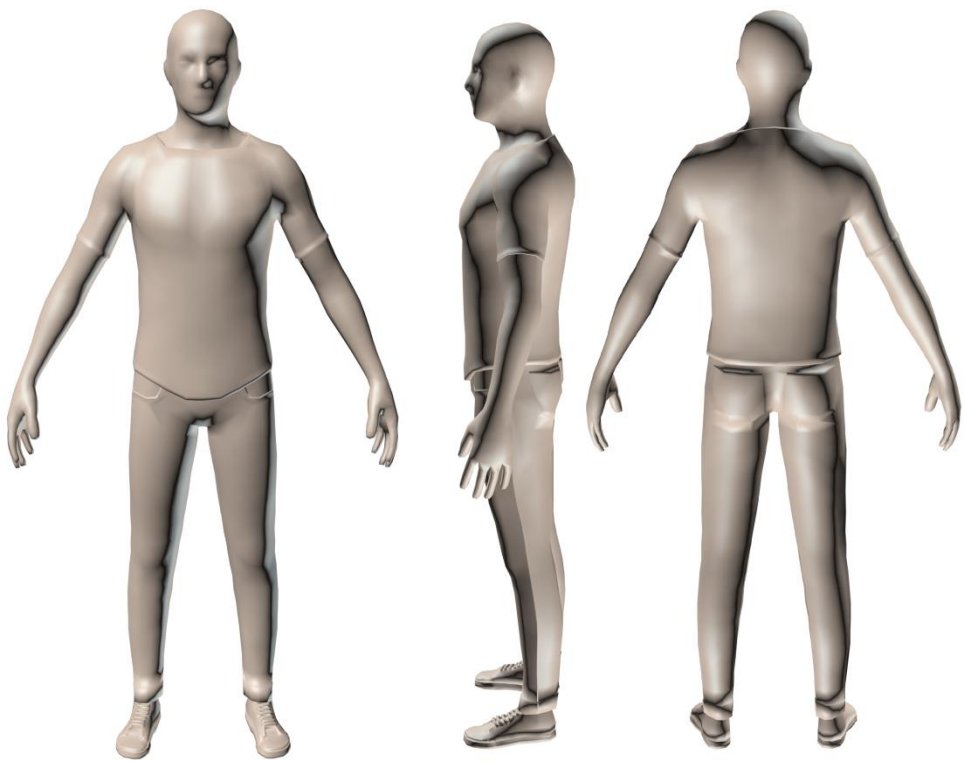


Figur 14: Bilden visar en systembild på kommunikationslänken.

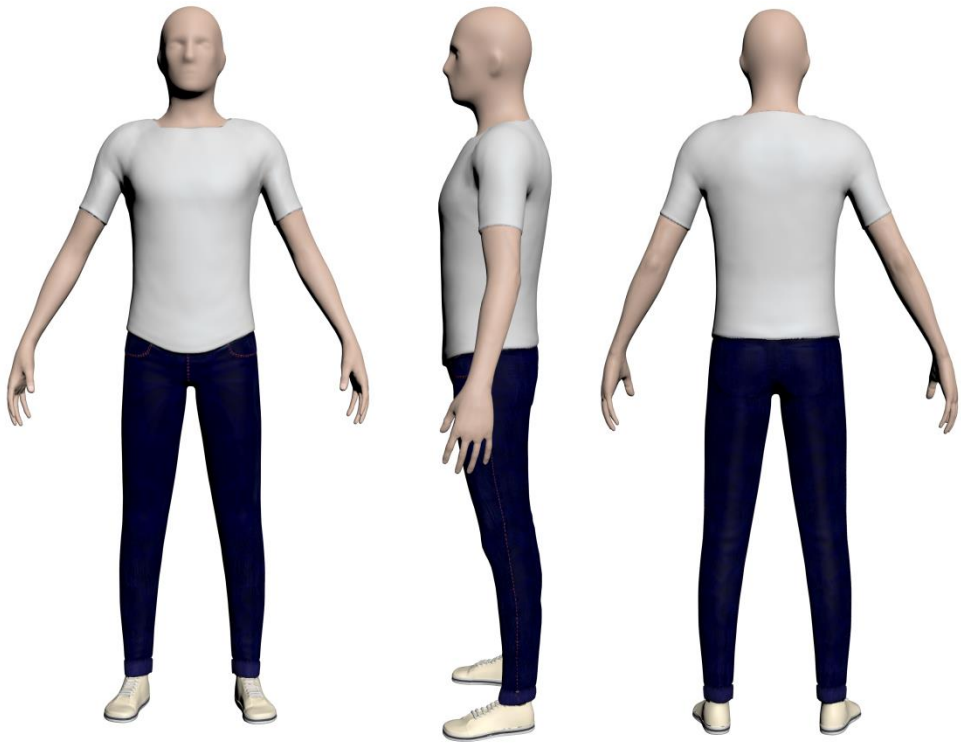
Resultat

Människomodell

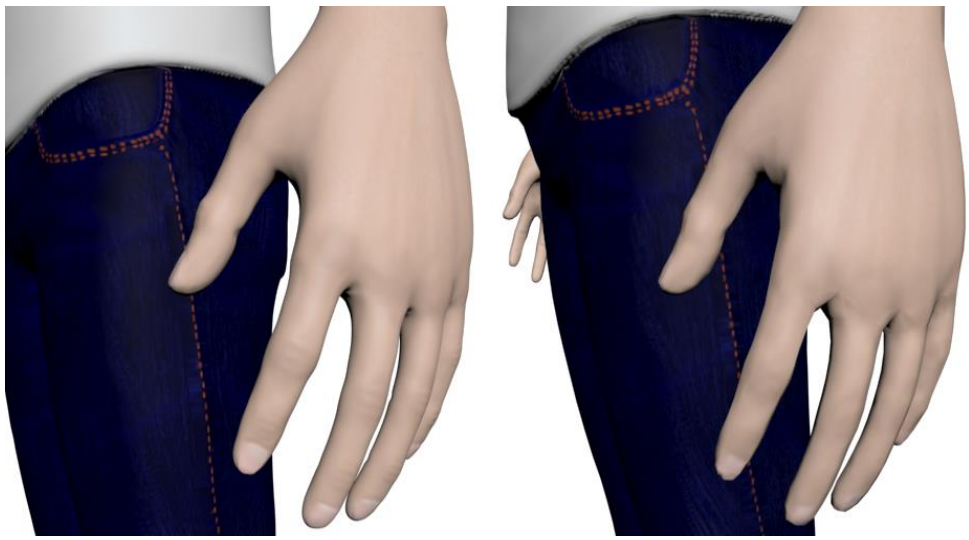
Resultatet från den första grövre ommodelleringen gjord i 3ds Max visas i Figur 15, där kläder, skor och större former har lagts till. Denna modell vidareutvecklades sedan i Mudbox, där färg och detaljer lades till. Resultatet av detta visas i Figur 16. Slutligen projicerades de finare detaljerna ner på en basmodell med färre antal polygoner, där skillnaden visas i Figur 17. Denna modell med projicerade detaljer riggades därefter för att kunna röras.



Figur 15: Bilden visar människomodellen i 3ds Max innan detaljer och färg lagts till.



Figur 16: Bilden visar den färdigmodellerade människomodellen renderad i 3ds Max.



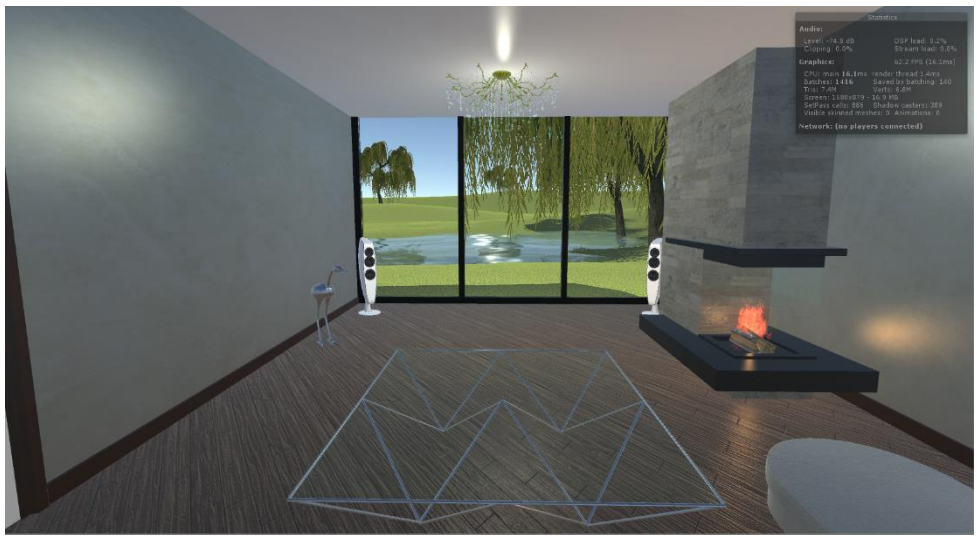
Figur 17: Bilden visar en jämförelse mellan detaljerna innan projicering (vänster) och efter de har projicerats på basmodellen.

Spelvärlden

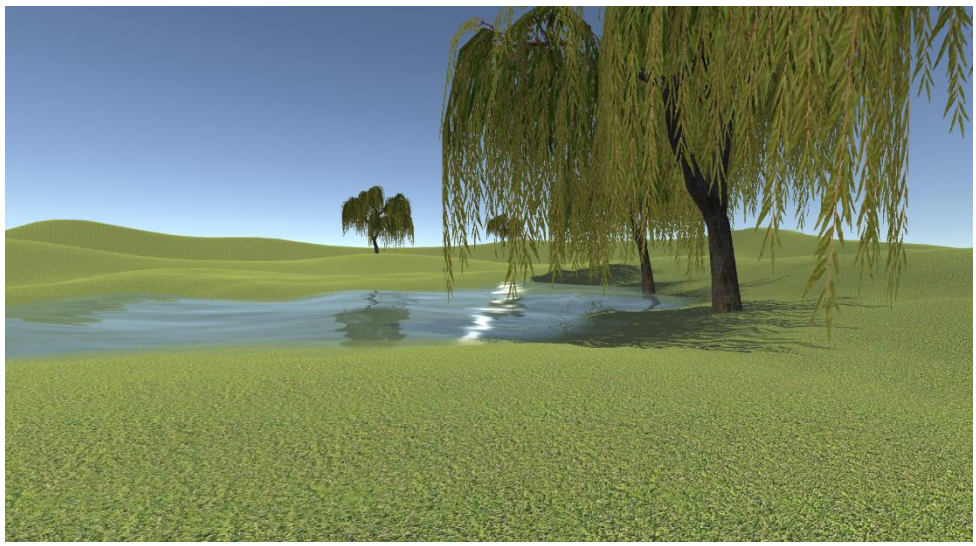
Rummet som modellerades i 3ds Max visas inifrån i Figur 18 och utifrån i Figur 19. Detta rum placerades sedan i en miljö som skapades i Unity, vilket visas i Figur 20 och Figur 21.

Figur 18: Bilden visar en rendering av rummet inifrån utförd i 3ds Max.

Figur 19: Bilden visar en rendering av rummet utifrån utförd i 3ds Max.



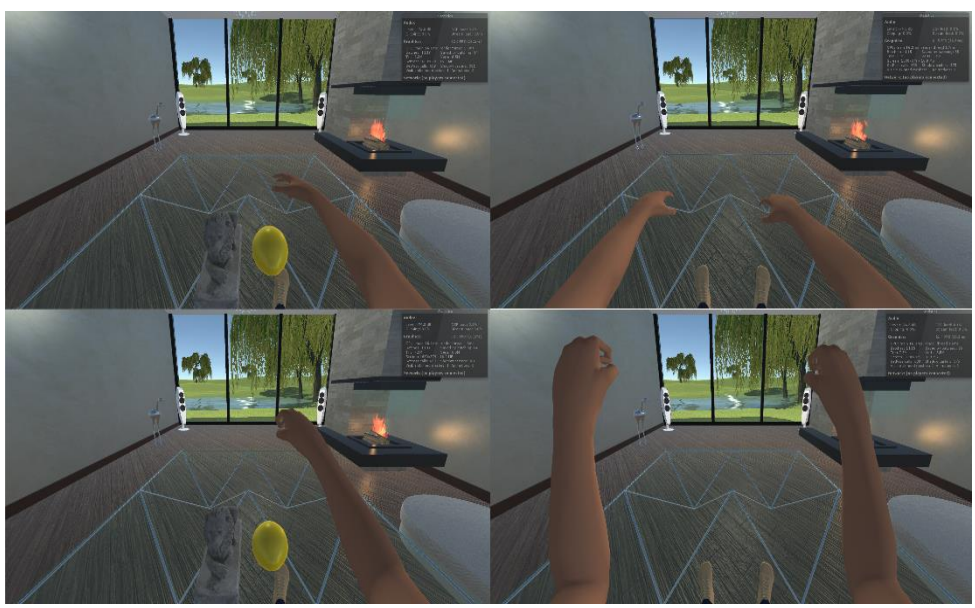
Figur 20: En skärmbild tagen inuti spelmotorn inifrån rummet med en partikeleffekt för brasan tillagd. Rutan uppe i högra hörnet visar renderingsinformation och syns ej då verktyget används.



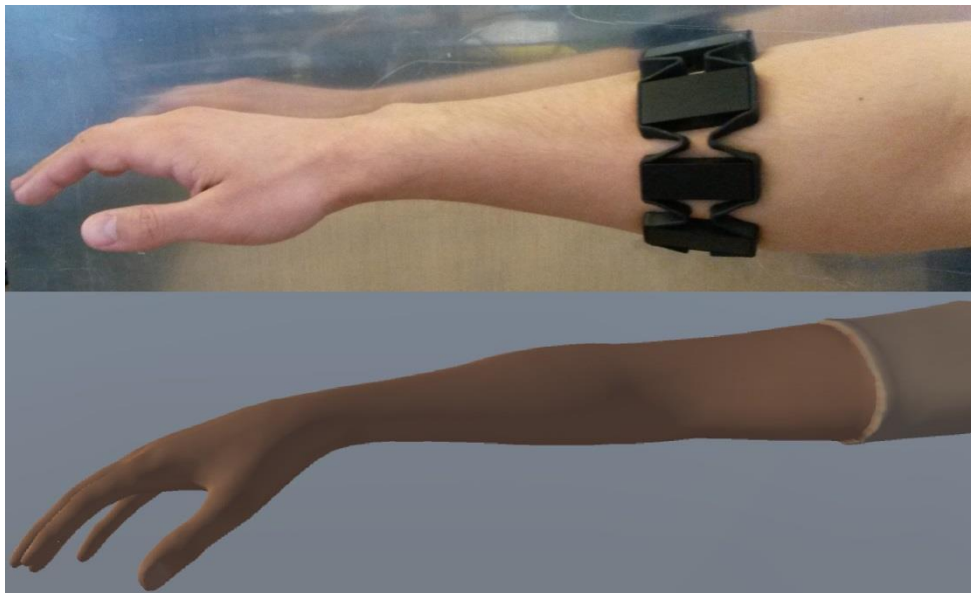
Figur 21: En skärmbild tagen inuti spelmotorn på miljön. Kameran är placerad precis utanför rummet.

Kommunikationslänken och styrning

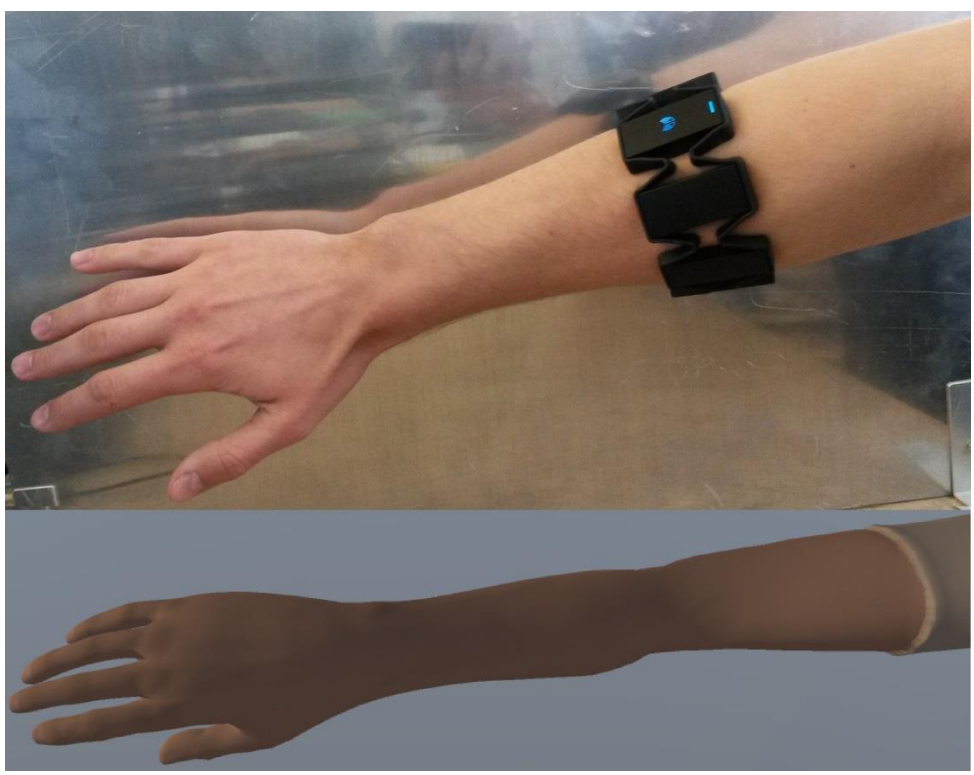
De skript och den kod som används för att sköta kommunikationen mellan Matlab och Unity redovisas i Appendix 3 och 4. Figur 22 visar skärmbilder från det färdiga programmet då endast Myo:n används för att styra modellen. I de två vänstra bilderna visas interaktionsobjekten medan de är dolda i de två högra. I de två högra bilderna är även speglingen aktiverad. De två övre bilderna visar öppna händer medan de två undre bilderna visar knutna händer som styrs med hjälp av inbyggda gester i Myo:n. Figur 23, Figur 24 och Figur 25 visar den verkliga armens position i kombination med vad simuleringen visar då endast Myo:n används för att styra modellen. Figur 26, Figur 27 och Figur 28 visar den verkliga handens position i kombination med modellens resulterande position då endast CyberGlove:n används för att styra simuleringen. I alla bilder där den verkliga armen/handen även visas har modellens arm/hand isolerats för att tydligare visa resultatet. Slutligen visar Figur 29 då både CyberGlove:n och Myo:n används i kombination.



Figur 22: Bilden visar hur verktyget ser ut under drift då endast Myo:n används för att styra modellen. Informationsrutan i det övre högra hörnet visas inte under simuleringarna.

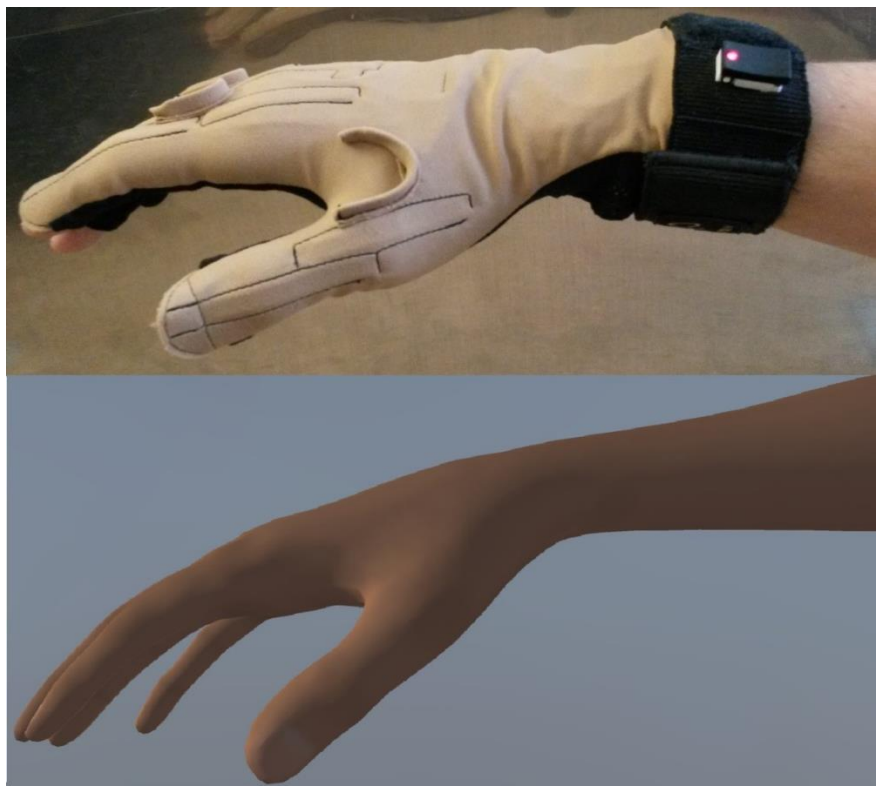


Figur 23: Den övre bilden visar den verkliga armens position medan den undre bilden visar den resulterande positionen av modellens arm då endast Myo:n används för att styra simuleringen.



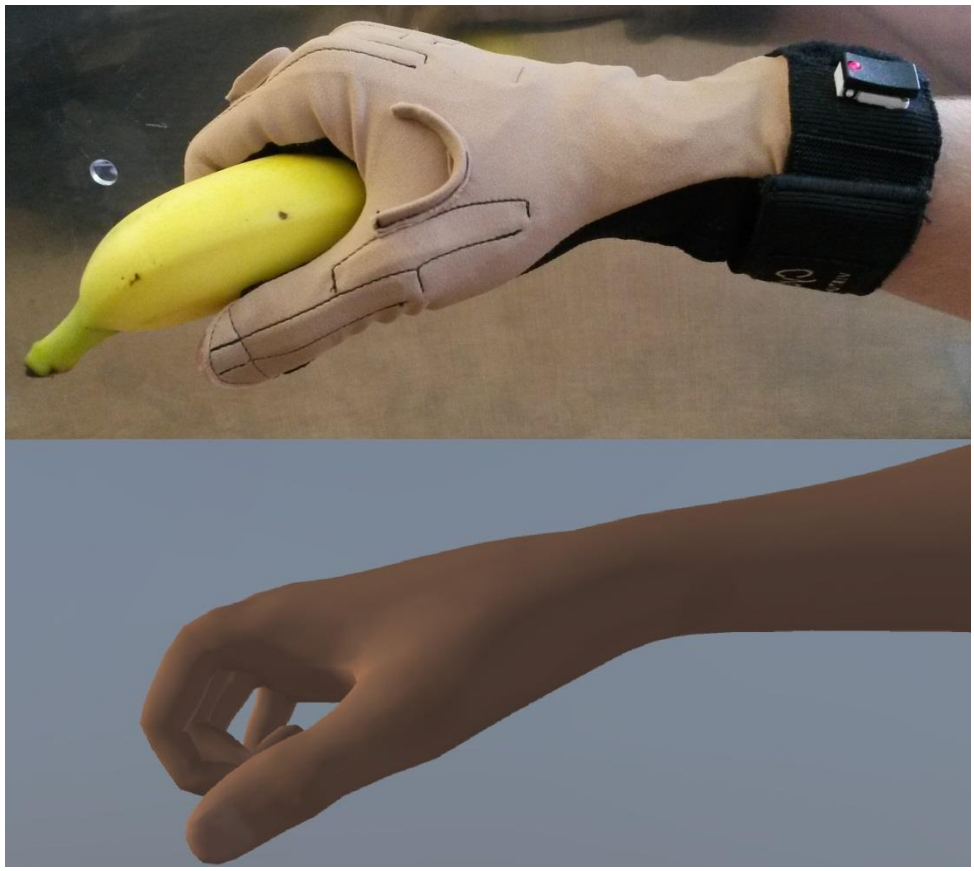
Figur 24 Den övre bilden visar den verkliga armens position och den undre bilden visar simuleringen då endast Myo:n används för att styra simuleringen.

Figur 25: Den övre bilden visar den verkliga armens position och den undre bilden visar simuleringen då endast Myo:n används för att styra simuleringen.

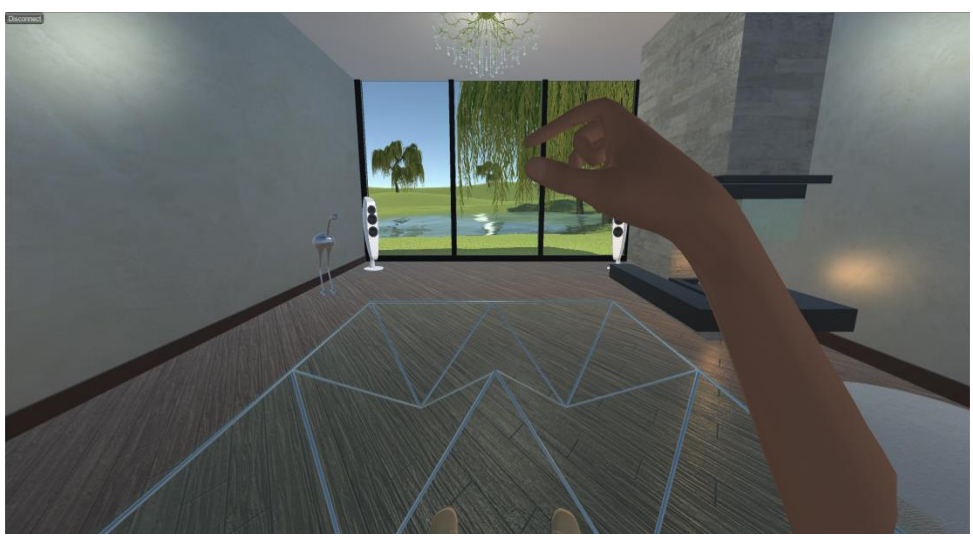


Figur 26: Den övre bilden visar den verkliga handens position och den undre bilden visar simuleringen då endast CyberGlove:n används för att styra simuleringen.

Figur 27: Den övre bilden visar den verkliga handens position och den undre bilden visar simuleringen då endast CyberGlove:n används för att styra simuleringen.



Figur 28: Den övre bilden visar den verkliga handens position och den undre bilden visar simuleringen då endast CyberGlove:n används för att styra simuleringen.



Figur 29: Bilden visar verktyget då både Myo:n och CyberGlove:n används för att styra simuleringen.

Diskussion och slutsatser

Den slutgiltiga människomodellen uppfyllde de krav som sattes. Då endast armar och ben kan ses av användaren var det här störst fokus lades. Dessa kroppsdelar rör sig korrekt med undantag av armbågarna då underarmen roteras runt mer än en axel samtidigt. Myo:n är utformad för att placeras nära armbågen på underarmen vilket resulterar i att rotationer av underarmen inte korrekt roterar människomodellens underarm. Effekten blir istället något lik rörelsen av en stelopererad underarm. Detta hade kunnat gå att åtgärda genom att ha en sensor som placeras närmare handen, vilket lett till en större rotation av underarmen nära armbågen. Detta hade ökat förvrängningen vid armbågen, vilket är välkänt inom modelleringsindustrin och kallas för ”candy wrapping effect”. En bättre lösning hade varit att lägga till två ben i underarmen för att återskapa hur underarmens ben ser ut i verkligheten. Detta hade dock ökat prestandakraven och komplicerat styrningen av modellen. Ett alternativ hade varit att skapa olika grafikinställningar som resulterat i olika prestandakrav för verktyget, vilket hade varit ett naturligt nästa steg om projektet vidareutvecklats.

Spelvärlden blev tillräckligt realistisk för sammanhanget och uppfyllde därmed de mål som sattes på den. Den skulle kunna utökas till att innehålla fler miljöer med olika aktiviteter för protesträning. Dessa nya miljöer skulle även kunna bestå av olika svårighetsnivåer. Ett ytterligare tillägg som ökat realismen hade varit ljud från den omgivande miljön och ljudåterkoppling som synkroniserats med känseláterkopplingen.

Överlag blev projektet lyckat där modellens rörelser väl stämmer överens med de rörelser användaren gör om en korrekt kalibrering skett. Det enda fall då rörelsen inte är helt korrekt är vid rotation av underarmen vilket främst beror på modellens utformning men även sensorns placering. Det som begränsar hur väl modellen rör sig i övrigt är hur väl de sensorer som används kan avläsa vilken position den verkliga kroppsdelen har. Verktyget i sig kan följa den verkliga kroppens rörelser väldigt väl om sensorerna ger en korrekt information om kroppsdelarnas positioner. Den enda inverkan som verktyget har är på grund av människomodellens proportioner mellan de olika kroppsdelarna. En användare som till exempel har väldigt långa fingrar jämfört med modellen kommer märka att modellens fingrar roteras mer än användarens egna. Detta hade kunnat lösas med hjälp av en modell som utvecklaren kunnat förändra

proportionerna på. Nackdelen med detta är främst att modellen hade blivit avsevärt svårare att skapa, men även att verktyget blivit svårare att använda.

Rotationer på alla ben kan ske med en precision på 10^{-5} grader från Matlab till Unity. Den valda precisionen är med en decimals noggrannhet och den största rörelsen som kan ske på ett kommando via TCP/IP-länken är $\pm$999,9 grader. Denna maximala rörelse förväntas aldrig ske, och en typisk rörelsestorlek förväntas vara i storleksordning av ett fåtal grader. Då prestandakravet inte påverkas av hur stor den övre gränsen är valdes en som garanterat inte kommer överskridas. Även detta kan modifieras av användaren men detta intervall bör fungera väl för de flesta användningsområden.

Kommunikationen mellan Matlab och Unity fungerade väldigt väl och hastigheten kunde hållas hög. Den hastighet som valdes under testningen av verktyget var att ett meddelande skickades var tionde millisekund. Vid denna hastighet upplevdes ingen fördröjning. En högre hastighet innebär ett högre krav på prestanda. Denna hastighet kan ökas och sänkas beroende på användarens krav. En utökning för att styra hela kroppen skulle kunna implementeras men då detta tagit extra tid utan att visa på nya funktioner valdes verktyget att hållas mindre komplicerat. Fokus lades istället på andra områden såsom känselåterkoppling. Användaren kan med enbart grundläggande programmeringskunskaper ändra vilken kroppsdel som styrs av den mottagna rotationsinformationen från Matlab, eller välja att utöka styrningen till hela kroppen.

Något som saknas i detta arbete är en utvärdering där patienter, terapeuter och utvecklare fått möjlighet att testa verktyget. Synpunkter och tips från dem vore värdefulla. Detta hade prioriterats vid en vidareutveckling av projektet. Då hade även framtida utvecklingsområden kunnat diskuteras med slutanvändaren och de skulle kunna vara delaktiga i denna process. Tyvärr vore det en alltför omfattande process för att kunna inkludera i detta arbete.

Överlag visade sig modelleringen vara det svåraste området. Där behövdes modellens realism ständigt vägas mot den prestanda som krävs för att kunna driva modellen under simuleringens gång utan att alltför mycket dra ner på uppdateringsfrekvensen. Projektets genomgående arbetsflöde var att

först skapa en grundläggande lösning som sedan utökades steg för steg för att sedan resultera i ett komplett verktyg.

Det utfördes inget omfattande test på modellens förmåga att följa användarens rörelse eftersom begränsningen snarare ligger i hur väl sensorerna kan avläsa rörelsen.

Referenser

Acosta, A. M., Dewald, H. A., & Dewald, J. P. (2012). Pilot study to test effectiveness of video game on reaching performance in stroke. *Journal of Rehabilitation Research & Development*, 431-444.

Akademiska sjukhuset. (den 27 Augusti 2013). *Allt fler äldre går med protes efter amputation.* Hämtat från Akademiska sjukhuset: http://www.akademiska.se/sv/Pressrum_old/Pressmeddelanden/Allt-fler-aldre-lar-sig-ga-efter-amputation/

Burdea, G. C., Cioi, D., Martin, J., Fensterheim, D., & Holenski, M. (2010). The Rutgers Arm II Rehabilitation System - A Feasibility Study. *Transactions on Neural Systems & Rehabilitation Engineering*, 505-514.

Buttussia, F., Pellisb, T., Vidania, A. C., Pauslerc, D., Carchiettic, E., & Chittaroa, L. (2013). Evaluation of a 3D serious game for advanced life support retraining. *International Journal of Medical Informatics*, 798-809.

CostHelper Prosthetic Arm. (2014). *Prosthetic Arm Cost.* Hämtat från CostHelper health: http://health.costhelper.com/prosthetic-arms.html

CostHelper Prosthetic Leg. (2014). *Prosthetic Leg Cost.* Hämtat från CostHelper health: http://health.costhelper.com/prosthetic-legs.html

Ezendam, D., Bongers, R. M., & Jannink, M. J. (2009). Systematic review of the effectiveness of mirror therapy in upper extremity function. *Disability and Rehabilitation*, 2135-2149.

Hoffman, H. G., Meyer, W. J., Ramirez, M., Roberts, L., Seibel, E. J., Atzori, B., . . . Patterson, D. R. (2014). Feasibility of Articulated Arm Mounted Oculus Rift Virtual Reality Goggles for Adjunctive Pain Control During Occupational Therapy in Pediatric Burn Patients. *Cyberpsychology, Behavior, and Social Networking*, 397-401.

Jack, D., Boian, R., Merians, A. S., Tremaine, M., Burdea, G. C., Adamovich, S. V., . . . Poizner, H. (2001). Virtual Reality-

Enhanced Stroke Rehabilitation. *Neural Systems and Rehabilitation Engineering*, 308-318.

Knighta, J. F., Carleyb, S., Tregunnac, B., Jarvis, S., Smithiesd, R., Freitas, S. d., . . . Mackway-Jonesb, K. (2010). Serious gaming technology in major incident triage training: A pragmatic controlled trial. *Resuscitation*, 1175-1179.

Laff, M. (2007). *Serious Gaming: The Trainer's New Best Friend.* American Society for Training & Development.

Lee, Y.-H., Heeter, C., Magerko, B., & Medler, B. (2012). Gaming Mindsets: Implicit Theories in Serious Game Learning. *Cyberpsychology, Behavior and Social Networking*, 190-194.

Lowe, R. (den 7 Mars 2015). *Mirror Therapy*. Hämtat från Physiopedia: http://www.physio-pedia.com/Mirror_Therapy

MacLachlan, M., McDonald, D., & Waloch, J. (2004). Mirror treatment of lower limb phantom pain: A case study. *Disability and Rehabilitation*, 901-904.

Merians, A. S., Fluet, G. G., Qiu, Q., Saleh, S., Lafond, I., Davidow, A., & Adamovich, S. V. (2011). Robotically facilitated virtual rehabilitation of arm transport integrated with finger movement in persons with hemiparesis. *Journal of NeuroEngineering and Rehabilitation*.

Mickelsson, M. (2004). *Fantomsmärta & Psykosyntes.* Stockholm: HumaNova – Europeiska Psykosyntesuniversitetet.

Mónica da Silva Cameirão, S. B., Duarte, E., & Verschure, P. F. (2011). Virtual reality based rehabilitation speeds up functional recovery of the upper extremities after stroke: A randomized controlled pilot study in the acute phase of stroke using the Rehabilitation Gaming System. *Restorative Neurology and Neuroscience*, 287-298.

Novak, D., Nagle, A., Keller, U., & Riener, R. (2014). *Increasing motivation in robot-aided arm rehabilitation with competitive and cooperative gameplay.* Zurich: Journal of NeuroEngineering and Rehabilitation.

Ortiz-Catalan, M., Nijenhuis, S., Ambrosch, K., Bovend'Eerdt, T., Koenig, S., & Lange, B. (2014). Virtual Reality. i J. L. Pons, & D. Torricelli, *Emerging Therapies in Neurorehabilitation* (ss. 249-265). Göteborg: Springer-Verlag Berlin Heidelberg.

Ortiz-Catalan, M., Sander, N., Kristoffersen, M. B., Håkansson, B., & Brånemark, R. (2014). *Treatment of phantom limb pain (PLP) based on augmented reality and gaming controlled by myoelectric pattern recognition: a case study of a chronic PLP patient.* Göteborg: Frontiers in Neuroscience.

Perez-Marcos, D., Sanchez-Vives, M. V., & Slater, M. (2011). *Is my hand connected to my body? The impact of body continuity.* Barcelona: Springer Science+Business Media B.V.

Personskadeförbundet RTP. (u.d.). *Amputation.* Hämtat från Personskadeförbundet RTP: http://www.rtps.se/index.php?id=164

Personskadeförbundet RTP. (u.d.). *Personskadeförbundet RTP - Benamputerade*. Hämtat från Protesen och protesutprovning: http://www.rtp.se/stockholm/index.php?id=261

Personskadeförbundet RTP. (u.d.). *Protesen och protesutprovning - Armamputerade*. Hämtat från Personskadeförbundet RTP: http://www.rtps.se/index.php?id=165

Resnik, L., Etter, K., Lieberman Klinger, S., & Kambe, C. (2011). Using virtual reality environment to facilitate training with advanced upper-limb prosthesis. *Journal of Rehabilitation Research & Development*, 707-718.

Rizzo, A. "., & Kim, G. J. (2005). A SWOT Analysis of the Field of Virtual Reality Rehabilitation and Therapy. *Presence*, 119-146.

Sebelius, F. (2010). *Smarthand Newsletter 2*. Hämtat från Smarthand Newsletter 2: http://www.elmat.lth.se/~smarthand/smarthand_newsletter_2.pdf

Simmons, C. D., Arthanat, S., & Macri, V. J. (2014). Pilot study: Computer-based virtual anatomical interactivity for rehabilitation of individuals with chronic acquired brain injury. *Journal of Rehabilitation Research & Development*, 377-399.

Smith, R. (2010). The Long History of Gaming in Military Training. *Simulation & Gaming*, 6-19.

Turolla, A., Dam, M., Ventura, L., Tonin, P., Agostini, M., Zucconi, C., . . . Piron, L. (2013). Virtual reality for the rehabilitation of the upper limb motor function after stroke: a prospective controlled trial. *Journal of NeuroEngineering and Rehabilitation*.

UXL Encyclopedia of Science. (2002). *Encyclopedia.com*. Hämtat från Protheses and implants: http://www.encyclopedia.com/topic/Prostheses_and_implants.aspx

Wrzesien, M., & Raya, M. A. (2010). *Learning in serious virtual worlds: Evaluation of learning effectiveness and appeal to students in the E-Junior project.* Valencia: Elsevier.

Bildkällor

Figur 1: Plusea. Hämtad den 1 juni 2015 från http://www.plusea.at/?p=3779.

Figur 2: Nick Summers. Publicerad den 10 juni 2014. Hämtad den 1 juni 2015 från http://thenextweb.com/gadgets/2014/06/10/thalmic-labs-unveils-final-design-myo-armband-minority-report-style-tech-control/.

Figur 3: SPS. Hämtad den 25 maj 2015 från http://www.spsco.com/bebionic-v3-hand.html/.

Figur 4: Toby Hudson. Publicerad den 8 november 2012. Hämtad den 25 maj 2015 från http://en.wikipedia.org/wiki/Prosthesis.

Figur 5: DanMic Global. Hämtad den 23 maj 2015 från http://www.danmicglobal.com/Scan-Mirror-Therapy.aspx.

Figur 6: Riot Pixels. Publicerad den 9 december 2013. Hämtad den 23 maj 2015 från http://en.riotpixels.com/games/tom-clancys-the-division/screenshots/.

Figur 7: Stacywang2014. Publicerad den 5 mars 2014. Hämtad den 23 maj 2015 från http://mag.rehabfun.com/tag/vr-stroke/.

Figur 8: Författarens egen.

Figur 9: Mariana Ruiz Villarreal. Publicerad den 3 januari 2007. Hämtad den 3 juni 2015 från http://en.wikipedia.org/wiki/Arm.

Figur 10: Författarens egen.

Figur 11: Författarens egen.

Figur 12: Författarens egen.

Figur 13: Författarens egen.

Figur 14: Författarens egen.

Figur 15: Författarens egen.

Figur 16: Författarens egen.

Figur 17: Författarens egen.

Figur 18: Författarens egen.

Figur 19: Författarens egen.

Figur 20: Författarens egen.

Figur 21: Författarens egen.

Figur 22: Författarens egen.

Figur 23: Författarens egen.

Figur 24: Författarens egen.

Figur 25: Författarens egen.

Figur 26: Författarens egen.

Figur 27: Författarens egen.

Figur 28: Författarens egen.

Figur 29: Författarens egen.

Appendix I – Användarguide

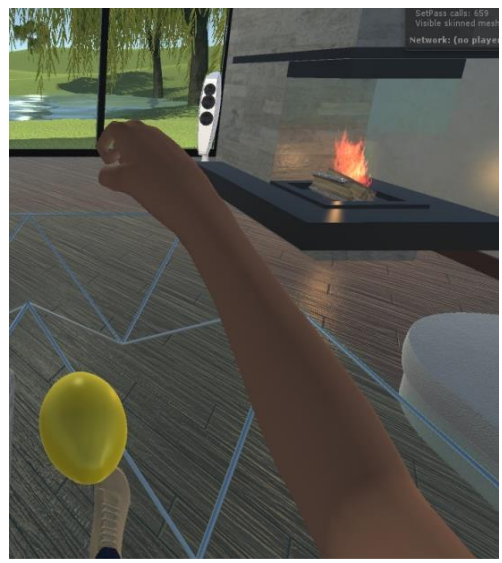
1. Installera följande programvaror:
   - Matlab r2015a
   - Unity 5
   - Drivrutin för Myo
2. Koppla in Bluetooth-sändaren för Myo:n om denna planeras användas.
3. Koppla in CyberGlove:n om denna planeras användas.
   a. Gå in enhetshanteraren och kontrollera att den port som CyberGlove:n är inkopplad på har samma avläsningshastighet angiven som kontrollenheten till CyberGlove.
   b. Notera portens namn, t.ex. COM4
4. Starta Matlab.
   a. Kontrollera att de två funktionerna ”checkConnection.m” och ”mat2string.m” finns i den aktuella mappen.
   b. Ändra porten som används för att skapa kommunikationslänken för CyberGlove:n till den som noterats tidigare.
   c. Kör kalibreringsskriptet eller ladda in den undansparade kalibreringen.
5. Starta Unity.
   a. Starta projektet och välj den scen du vill använda.
   b. Ställ in de inställningar du vill använda då du använder programmet i kontrollkoden. Detta är bifogat till människomodellen och kan manipuleras genom att markera människomodellen i Unitys editor. Val som spegling och sensoruppsättning finns här.
   c. Om TCP/IP-länken har ändrats i Matlab behöver den även ändras i skriptet ”TCPConnection.cs”.
   d. Om paustiden förändrats i Matlab behöver det även ändras till samma värde i kontrollskriptet.
6. Starta Unity-scenen genom att trycka på play-knappen.
7. Om de sensoruppsättningar som används är direkt implementerade i Unity kan steg 8-11 hoppas över.
8. Starta Matlab-skriptet.
9. Tryck på ”Connect”-knappen i Unitys övre vänstra hörn.
10. Länken är nu skapad och Matlab skickar data till Unity.
11. Knappen ”T” används för att återställa fingrarnas position till den ursprungliga och knappen ”R” används för att återställa underarmens position till den ursprungliga.

12. Tryck på ”Disconnect”-knappen då simuleringen ska avslutas för att stänga TCP/IP-länken.
13. Avsluta sedan Unity-scenen genom att trycka på stoppikonen.
14. Båda programmen kan nu stängas ner.

Appendix II – Populärvetenskaplig sammanfattning

Ett virtuellt rehabiliteringsprogram för proteträning och behandling av fantomsmärtor.

VEARM är en virtuell människomodell som går att styra med sensorer fästa på användarens verkliga kropp. Verktyget är bland annat framtaget för att underlätta protesutveckling, då detta i dagsläget är en väldigt lång och dyr process. Dessutom lämpar sig även verktyget för rehabilitering av fantomsmärtor.



Ovan visas verktyget under drift

VEARM (Virtual Environment Arm) är ett verktyg som går att använda för många olika saker. Det är främst framtaget för att användas vid proteträning, där patienter kan kalibrera och lära sig styra sina proteser innan de blivit levererade. Verktyget gör det även möjligt för utvecklare och terapeuter att tillsammans med patienten testa olika placeringar av protesens sensorer och snabbt testa hur väl de fungerar. Detta underlättar skräddarsydda sensoruppsättningar för varje patient eftersom alla placeringar kan testas och bestämmas utan att prototyper av protesen behöver beställas. Att kunna skräddarsy varje patients protes är ett mål som i dagsläget inte går att uppfylla. Anledningen till att alla inte kan använda samma protes likaväl grundar sig på den stora variation som finns bland patienters individuella förutsättningar och mål. Faktorer påverkar detta är till exempel under vilka aktiviteter de vill använda sin protes och hur mycket tid de är beredda att lägga på att lära sig styra protesen väl. En ung aktiv patient som spelar gitarr behöver till exempel en mycket mer avancerad protes än en äldre patient som är nöjd med att kunna stödja sig mot sin protes och vill kunna använda den direkt.

Ett annat område som är på frammarsch när det gäller proteser är känselåterkoppling som ger patienten möjlighet att kunna känna vad protesen rör vid. Även här kan verktyget användas för att testa hur många

olika kanaler som användaren vill ha känselåterkoppling på. Vill patienten till exempel kunna känna om varje enskilt finger rör vid något eller räcker det med en signal som visar om handen rör vid något över huvud taget? Hur stark ska denna återkoppling vara? Hur känslig och hur stor variation ska återkopplingen kunna erbjuda? Detta är frågor som verktyget kan underlätta att besvara.

Ytterligare ett användningsområde är rehabilitering av fantomsmärtor. Detta är smärtor som ofta uppstår i förlorade kroppsdelar, där patienten till exempel upplever sig ha ont i en förlorad hand. En metod som används i dagsläget är att patienten placerar sin stump bakom en spegel som reflekterar den friska kroppsdelen. Då ges illusionen av att spegelbilden är en riktig kroppsdel och genom att utföra olika övningar och avslappningar dämpas fantomsmärtan. I bästa fall försvinner fantomsmärtorna för gott efter ett flertal rehabiliteringstillfällen, men detta är tyvärr inte alltid fallet. Denna rehabilitering ställer även krav på att den motsatta sidans kroppsdel ska vara frisk. Ett nytt sätt att behandla detta på är att avläsa muskelsignalerna i stumpen och koppla dem mot en virtuell arm som dessa rehabiliteringsövningar utförs med. Eftersom rätt muskler då används för att styra den virtuella armen blir rehabiliteringen mer verklighetstrogen och illusionen av att armen tillhör patienten starkare. Rehabiliteringen kan även göras roligare genom att koppla styrningen mot spel.

Slutligen kan verktyget även användas för att träna upp rörligheten i armar och händer hos strokepatienter. Fördelen med ett virtuellt träningsverktyg är en roligare rehabilitering där patienten får konsekvent återkoppling på hur väl övningarna utförts. Det är nämligen ett problem i dagsläget då patienten behöver göra många övningar på egen hand, vilket snabbt leder till en lägre motivation och sämre utförda övningar. Svårighetsgraden på övningarna är också lättare att öka allteftersom.

För att hålla verktyget generellt går människomodellen att styra via det väletablerade beräkningsprogrammet Matlab. Till detta kan alla möjliga sensorer och återkopplingssystem kopplas, och signalen kan behandlas och manipuleras av utvecklare och terapeuter.

Läs hela rapporten ”VEARM – En hjälpande hand” skriven av Saša Kojić på LUP.

Appendix III – MATLAB-kod

Huvudskript

```
%% Mapping charts
% CyberGlove I data mapping (18 sensors edition was used in this
case):
% Number of sensors:                                  (18)   (22)
% CyberGlove1Sensor01 == ThumbRoll                    roll | roll
% CyberGlove1Sensor02 == ThumbInnerJoint           flexure | flexure
% CyberGlove1Sensor03 == ThumbOuterJoint           flexure | flexure
% CyberGlove1Sensor04 == ThumbIndexAbduction     abduction | abduction
% CyberGlove1Sensor05 == IndexInnerJoint           flexure | flexure
% CyberGlove1Sensor06 == IndexMiddleJoint          flexure | flexure
% CyberGlove1Sensor07 == IndexOuterJoint               N/A | flexure
% CyberGlove1Sensor08 == EMPTY                         N/A | N/A
% CyberGlove1Sensor09 == MiddleInnerJoint          flexure | flexure
% CyberGlove1Sensor10 == MiddleMiddleJoint         flexure | flexure
% CyberGlove1Sensor11 == MiddleOuterJoint              N/A | flexure
% CyberGlove1Sensor12 == MiddleIndexAbduction    abduction | abduction
% CyberGlove1Sensor13 == RingInnerJoint            flexure | flexure
% CyberGlove1Sensor14 == RingMiddleJoint           flexure | flexure
% CyberGlove1Sensor15 == RingOuterJoint                N/A | flexure
% CyberGlove1Sensor16 == RingMiddleAbduction     abduction | abduction
% CyberGlove1Sensor17 == PinkyInnerJoint           flexure | flexure
% CyberGlove1Sensor18 == PinkyMiddleJoint          flexure | flexure
% CyberGlove1Sensor19 == PinkyOuterJoint               N/A | flexure
% CyberGlove1Sensor20 == PinkyRingAbduction      abduction | abduction
% CyberGlove1Sensor21 == PalmArch                      arch | arch
% CyberGlove1Sensor22 == WristFlexion              flexure | flexure
% CyberGlove1Sensor23 == WristAbduction          abduction | abduction
% Rotation matrix (rotMat) mapping, each row contains the x-value,
% y-value & z-value. The numbers describe the rows used in the
% matrix.
% Each line represents a bone in the hand
% The picture below shows the left hand, palm down starting from
% the wrist
%
%   |20  |16  |12  |8           |         Coordinates
%                               |  (Capital letters for axis)
%   |19  |15  |11  |7      |4   |              ^ Y
%                               |              |
%   |18  |14  |10  |6     /3    |       X      |
%                               |       < - -  x  Z
%   \17  |13  |9   /5   /2       |
%
%          1|

%% Choose the range of motion in degrees
rom_max = [10;    % ThumbRoll
           90;    % ThumbInnerJoint
           90;    % ThumbOuterJoint
           20;    % ThumbIndexAbduction
           90;    % IndexInnerJoint
           90;    % IndexMiddleJoint
           90;    % MiddleInnerJoint
```

```
            90;    % MiddleMiddleJoint
            15;    % MiddleIndexAbduction
            90;    % RingInnerJoint
            90;    % RingMiddleJoint
            15;    % RingMiddleAbduction
            90;    % PinkyInnerJoint
            90;    % PinkyMiddleJoint
            15;    % PinkyRingAbduction
            15;    % PalmArch
            90;    % WristFlexion
            90];   % WristAbduction

rom_min = [0;    % ThumbRoll
           0;    % ThumbInnerJoint
           0;    % ThumbOuterJoint
           0;    % ThumbIndexAbduction
           0;    % IndexInnerJoint
           0;    % IndexMiddleJoint
           0;    % MiddleInnerJoint
           0;    % MiddleMiddleJoint
           -5;   % MiddleIndexAbduction
           0;    % RingInnerJoint
           0;    % RingMiddleJoint
           -5;   % RingMiddleAbduction
           0;    % PinkyInnerJoint
           0;    % PinkyMiddleJoint
           -5;   % PinkyRingAbduction
           0;    % PalmArch
           0;    % WristFlexion
           0];   % WristAbduction
%% Calibration - Run first
% Change port accordingly
s = serial('COM6', 'BaudRate', 115200, 'InputBufferSize', 20);
rotMat = zeros(20,3);
rotMatOld = zeros(20,3);
disp('Flex your hand and tilt your wrist to the left and upwards in
the opposite direction of your palm');
pause(3);
fopen(s);
fwrite(s,'G');
data = fread(s);
calib_min = data(2:19);
disp('Data gathered');
pause(1);
disp('Make a fist and tilt your wrist to the right and inwards in
the direction of your palm');
pause(3);
fwrite(s,'G');
data = fread(s);
calib_max = data(2:19);
disp('Data gathered');
% Calculating the conversion vector between read values and
rotation in degrees
conversionVec = (rom_max - rom_min)./(calib_max - calib_min);
% Checks connection for each sensor, if the same value is returned
for the
```

```
% maximum and minimum value of the calibration the conversion
vector is set
% to 0 for that index
conversionVec = checkConnection( conversionVec, rom_min, rom_max );
disp('Calibration completed');
fclose(s);
%% Saved calibration, run this if you want to use it
conversionVec = [0.1;       % ThumbRoll
                 1.0;       % ThumbInnerJoint
                 1.8;       % ThumbOuterJoint
                 0.2;       % ThumbIndexAbduction
                 0.9;       % IndexInnerJoint
                 0.7;       % IndexMiddleJoint
                 1.0;       % MiddleInnerJoint
                 0.9;       % MiddleMiddleJoint
                 0.2;       % MiddleIndexAbduction
                 0.95;      % RingInnerJoint
                 0.8;       % RingMiddleJoint
                 0.4;       % RingMiddleAbduction
                 0.9;       % PinkyInnerJoint
                 0.9;       % PinkyMiddleJoint
                 0.5;       % PinkyRingAbduction
                 0.2;       % PalmArch
                 1.6;       % WristFlexion
                 0.9];      % WristAbduction
calib_max = [176;        % ThumbRoll
             140;        % ThumbInnerJoint
             150;        % ThumbOuterJoint
             180;        % ThumbIndexAbduction
             180;        % IndexInnerJoint
             180;        % IndexMiddleJoint
             180;        % MiddleInnerJoint
             180;        % MiddleMiddleJoint
             180;        % MiddleIndexAbduction
             170;        % RingInnerJoint
             170;        % RingMiddleJoint
             180;        % RingMiddleAbduction
             190;        % PinkyInnerJoint
             150;        % PinkyMiddleJoint
             150;        % PinkyRingAbduction
             150;        % PalmArch
             145;        % WristFlexion
             150];       % WristAbduction
calib_min = [60;         % ThumbRoll
             90;         % ThumbInnerJoint
             60;         % ThumbOuterJoint
             60;         % ThumbIndexAbduction
             40;         % IndexInnerJoint
             30;         % IndexMiddleJoint
             80;         % MiddleInnerJoint
             60;         % MiddleMiddleJoint
             80;         % MiddleIndexAbduction
             30;         % RingInnerJoint
             40;         % RingMiddleJoint
             30;         % RingMiddleAbduction
             40;         % PinkyInnerJoint
```

```
            50;         % PinkyMiddleJoint
            150;        % PinkyRingAbduction
            50;         % PalmArch
            90;         % WristFlexion
            50];        % WristAbduction
rotMat = zeros(20,3);
rotMatOld = zeros(20,3);
%% Accept a connection from any machine on port 27015.
t = tcpip('127.0.0.1', 125, 'NetworkRole', 'server');
% Accept a connection from any machine on port 27015.
%t = tcpip('0.0.0.0', 125, 'NetworkRole', 'server');
% Open a connection. This will not return until a connection is
received.
s = serial('COM6', 'BaudRate', 115200, 'InputBufferSize', 20);
fopen([s t]);
fwrite(t, 'Contact Established');
fwrite(s, 'G');
data = fread(s);
data = data(2:19);
pause(1)
MsgFromClient = '';
% First read (to prevent a rotation being sent in the first
message)
fwrite(s, 'G');
data = fread(s);
data = data(2:19);
% Converting the data to degrees
dataDeg = conversionVec.*data;
% Mapping the data
rotMat(1,1) = dataDeg(17);                % Wrist Flexion
rotMat(1,3) = -dataDeg(18);               % Wrist Abduction
rotMat(2,3) = -dataDeg(4);                % Thumb Index Abduction
rotMat(3,1) = dataDeg(2);                 % Thumb Inner Joint
rotMat(4,1) = dataDeg(3);                 % Thumb Outer Joint
rotMat(6,1) = dataDeg(5);                 % Index Inner Joint
rotMat(6,3) = -dataDeg(9);                % Middle Index Abduction
rotMat(7,1) = dataDeg(6);                 % Index Middle Joint
rotMat(8,1) = rotMat(7,1);                % Index Outer Joint
rotMat(10,1) = dataDeg(7);                % Middle Inner Joint
rotMat(10,3) = -0.5*dataDeg(12);          % Ring Middle Abduction
rotMat(11,1) = dataDeg(8);                % Middle Middle Joint
rotMat(12,1) = rotMat(11,1);              % Middle Outer Joint
rotMat(14,1) = dataDeg(10);               % Ring Inner Joint
rotMat(14,3) = 0.5*dataDeg(12);           % Ring Middle Abduction
rotMat(15,1) = dataDeg(11);               % Ring Middle Joint
rotMat(16,1) = rotMat(15,1);              % Ring Outer Joint
rotMat(18,1) = dataDeg(13);               % Pinky Inner Joint
rotMat(18,3) = dataDeg(12)+dataDeg(15);   % Pinky Ring Abduction
rotMat(19,1) = dataDeg(14);               % Pinky Middle Joint
rotMat(20,1) = rotMat(19,1);              % Pinky Outer Joint
rotMatOld = rotMat;
% Starting the loop
while(length(strfind(MsgFromClient, 'Disconnect')) < 1)
MsgFromClient = '';
% Match the pause to the "Send to Server" time in Unity
pause(0.1)
```

```
% Reading the message sent from the server
if(t.BytesAvailable > 0)
  recievedMsg = fread(t, t.BytesAvailable);
  for j=1:length(recievedMsg)-2
    MsgFromClient = [MsgFromClient (char(recievedMsg(j)))];
  end
end
% Reading the data from the glove
fwrite(s, 'G');
data = fread(s);
data = data(2:19)
% Making sure that the values are within the calibration
for i = 1:length(data)
  if(data(i) > calib_max(i))
    data(i) = calib_max(i);
  end
  if(data(i) < calib_min(i))
    data(i) = calib_min(i);
  end
end
% Converting the data to degrees
dataDeg = conversionVec.*data;
% Mapping the data
rotMat(1,1) = dataDeg(17);                  % Wrist Flexion
rotMat(1,3) = -dataDeg(18);                  % Wrist Abduction
rotMat(2,3) = -dataDeg(4);                   % Thumb Index
Abduction
rotMat(3,1) = dataDeg(2);                   % Thumb Inner Joint
rotMat(4,1) = dataDeg(3);                   % Thumb Outer Joint
rotMat(6,1) = dataDeg(5);                   % Index Inner Joint
rotMat(6,3) = -dataDeg(9);                   % Middle Index
Abduction
rotMat(7,1) = dataDeg(6);                   % Index Middle Joint
rotMat(8,1) = rotMat(7,1);                  % Index Outer Joint
rotMat(10,1) = dataDeg(7);                  % Middle Inner Joint
rotMat(10,3) = -0.5*dataDeg(12);                % Ring Middle
Abduction
rotMat(11,1) = dataDeg(8);                  % Middle Middle Joint
rotMat(12,1) = rotMat(11,1);                % Middle Outer Joint
rotMat(14,1) = dataDeg(10);                 % Ring Inner Joint
rotMat(14,3) = 0.5*dataDeg(12);                % Ring Middle
Abduction
rotMat(15,1) = dataDeg(11);                 % Ring Middle Joint
rotMat(16,1) = rotMat(15,1);                 % Ring Outer Joint
rotMat(18,1) = dataDeg(13);                 % Pinky Inner Joint
rotMat(18,3) = dataDeg(12)+dataDeg(15);    % Pinky Ring Abduction
rotMat(19,1) = dataDeg(14);                  % Pinky Middle Joint
rotMat(20,1) = rotMat(19,1);                 % Pinky Outer Joint
% Converting the rotation matrix to a string
rotMatDiff = rotMat-rotMatOld;
transmission = mat2string(rotMatDiff);
rotMatOld = rotMat;
fwrite(t, transmission);
fclose(s);
fclose(t);
delete(t);
```

```
delete(s);
```

Funktioner

```
function [output] = mat2string(input)
%MAT2STRING A function which transforms a matrix to a string
representation
%   Input is a matrix of size NxM where each entry is
%   outputted as a three-digit value, intended to be used
%   with Unity to send rotation data in degrees.
%   Ex input: [-1 0 30; 3 -145.2 5]
%   Output: "-001.0 +000.0 +030.0 +003 -145.2 +005.0 "
[height, ~] = size(input);
output_string = '';
inputRounded = round(10*input);
inputFloored = 10*fix(inputRounded/10);
decimals = inputRounded - inputFloored;
inputFloored = fix(input);
for i=1:height
  for k=1:3
    temp = abs(inputFloored(i,k));
    % Adding the sign
    if(input(i,k)<0)
      output_string = [output_string '-'];
    else
      output_string = [output_string '+'];
    end
    % Adding the integers
    if(temp<1)
      output_string = [output_string '000'];
    elseif(temp<10)
      output_string = [output_string '00' int2str(temp)];
    elseif(temp<100)
      output_string = [output_string '0' int2str(temp)];
    elseif(temp<1000)
      output_string = [output_string int2str(temp)];
    else
      output_string = [output_string '000'];
    end
    % Adding the decimal
    if(temp<1000)
      output_string = [output_string '.'
      int2str(abs(decimals(i,k))) ' '];
    else
      output_string = [output_string '.0 '];
    end
  end
end
output = output_string;
end
```

```
function [ output ] = checkConnection(conversionVec,
rom_min, rom_max )
%CHECKCONNECTION Checks connection for each sensor
% If the same value is returned for the maximum and minimum
% value of the calibration the conversion vector is set to 0
% for that index
[height, ~] = size(conversionVec);
string = '';
noContact = false;
for i=1:height
  if(abs(rom_max(i)-rom_min(i))<10)
    conversionVec(i) = 0;
    if(noContact)
      string = num2str(i);
    else
      string = [string ' ' num2str(i)];
    end
    noContact = true;
  end
end
if(noContact)
  disp(' ');
  disp('No difference between the max and min value was
  detected for the following sensors:')
  disp(string);
  disp('Possible detected values for these sensors will be
  ignored until a new calibration has been made');
  disp(' ');
end
output = conversionVec;
end
```

Appendix IV – Unitykod

Kontrollkod

Kod som är lagd på människomodellen för att underlätta variabeländringar.

```
using UnityEngine;
using System.Collections;
public class Controls : MonoBehaviour {
public bool EnableMyo = true;
public bool EnableCyberGlove = true;
public bool EnableMirroring = true;
public float fingerRotx, fingerRotz = 0.0f;
public float fingerRoty = 70.0f;
public string serverIP = "127.0.0.1";
public int serverPort = 27015;
public float sendToServer = 0.1f;
  public bool returnEnableMyo(){
    return EnableMyo;
  }
  public bool returnEnableCyberGlove(){
    return EnableCyberGlove;
  }
  public bool returnEnableMirroring(){
    return EnableMirroring;
  }
  public float returnfingerRotx(){
    return fingerRotx;
  }
  public float returnfingerRoty(){
    return fingerRoty;
  }
  public float returnfingerRotz(){
    return fingerRotz;
  }
  public string returnserverIP(){
    return serverIP;
  }
  public int returnserverPort(){
    return serverPort;
  }
  public float returnsendToServer(){
    return sendToServer;
  }
}
```

Socketskod

Denna kod sköter kommunikationen med Matlab och mappar om det sända meddelandet för att sedan rotera modellens ben korrekt. Koden är baserad på exempelkod hämtad från https://girlscancode.wordpress.com/2015/02/01/update-unity-and-tcpip-socket-connections/.

```
using UnityEngine;
```

```
using System.Collections;
using System.Collections.Generic;
using System.Text;
using System;
public class socketScript : MonoBehaviour {
  // Variables
  public TCPConnection myTCP;
  private string serverMsg;
  // Bones
  public GameObject humanModel = null;
  public GameObject lWrist = null;
  public GameObject rWrist = null;
  public GameObject lIndex01 = null;
  public GameObject lMiddle01 = null;
  public GameObject lRing01 = null;
  public GameObject lLittle01 = null;
  public GameObject lThumb01 = null;
  public GameObject rIndex01 = null;
  public GameObject rMiddle01 = null;
  public GameObject rRing01 = null;
  public GameObject rLittle01 = null;
  public GameObject rThumb01 = null;
  // Allocating variables for the rest of the bones
  // Left
  private GameObject lIndex02, lIndex03, lIndex04, lMiddle02,
lMiddle03,
  lMiddle04, lRing02, lRing03, lRing04, lLittle02, lLittle03,
lLittle04,
  lThumb02, lThumb03;
  private Vector3 lWristvec, lIndex01vec, lIndex02vec, lIndex03vec,
  lIndex04vec, lMiddle01vec, lMiddle02vec, lMiddle03vec,
lMiddle04vec,
  lRing01vec, lRing02vec, lRing03vec, lRing04vec, lLittle01vec,
  lLittle02vec, lLittle03vec, lLittle04vec, lThumb01vec,
lThumb02vec,
  lThumb03vec;
  private Vector3 lWristInitvec, lIndex01Initvec, lIndex02Initvec,
  lIndex03Initvec, lIndex04Initvec, lMiddle01Initvec,
lMiddle02Initvec,
  lMiddle03Initvec, lMiddle04Initvec, lRing01Initvec,
lRing02Initvec,
  lRing03Initvec, lRing04Initvec, lLittle01Initvec,
lLittle02Initvec,
  lLittle03Initvec, lLittle04Initvec, lThumb01Initvec,
lThumb02Initvec,
  lThumb03Initvec;
  // Right
  private GameObject rIndex02, rIndex03, rIndex04, rMiddle02,
rMiddle03,
  rMiddle04, rRing02, rRing03, rRing04, rLittle02, rLittle03,
rLittle04,
  rThumb02, rThumb03;
  private Vector3 rWristvec, rIndex01vec, rIndex02vec, rIndex03vec,
  rIndex04vec, rMiddle01vec, rMiddle02vec, rMiddle03vec,
rMiddle04vec,
  rRing01vec, rRing02vec,  rRing03vec, rRing04vec, rLittle01vec,
```

```
  rLittle02vec, rLittle03vec, rLittle04vec, rThumb01vec,
rThumb02vec,
  rThumb03vec;
  private Vector3 rWristInitvec, rIndex01Initvec, rIndex02Initvec,
  rIndex03Initvec, rIndex04Initvec, rMiddle01Initvec,
rMiddle02Initvec,
  rMiddle03Initvec, rMiddle04Initvec, rRing01Initvec,
rRing02Initvec,
  rRing03Initvec, rRing04Initvec, rLittle01Initvec,
rLittle02Initvec,
  rLittle03Initvec, rLittle04Initvec, rThumb01Initvec,
rThumb02Initvec,
  rThumb03Initvec;
  private bool enableCyberGlove, enableMyo, initCon, connected,
disconnect,
  mirror;
  private float temp, temp2, timePassed, sendToServer;
  void Start () {
    enableCyberGlove =
    humanModel.GetComponent<Controls>().returnEnableCyberGlove();
    enableMyo =
humanModel.GetComponent<Controls>().returnEnableMyo();
    sendToServer =
    humanModel.GetComponent<Controls>().returnsendToServer();
    mirror = humanModel.GetComponent<Controls>
().returnEnableMirroring ();
    initCon = false;
    connected = false;
    disconnect = false;
    timePassed = 0.0f;
    if (enableCyberGlove) {
      findBones();
      setInitVec();
    }
  }
  void Update () {
    if (enableCyberGlove && initCon) {
      // Keep checking the server for messages
      if(connected){
        timePassed += Time.deltaTime;
        if(sendToServer < timePassed){
          SendToServer ("Connected");
          timePassed = 0.0f;
        }
      }
      if (disconnect && connected){
        SendToServer ("Disconnect");
        connected = false;
      }
      SocketResponse ();
      if (Input.GetKeyDown ("t")) {
        resetFingerRotation();
      }
    }
  }
```

```
  void OnGUI() {
    if (enableCyberGlove) {
      // If connection has not been made, display button to connect
      if (initCon) {
        if (myTCP.socketReady == false || disconnect) {
          if (GUILayout.Button ("Connect")) {
            // Try to connect
            Debug.Log ("Attempting to connect..");
            myTCP.setupSocket ();
            if(myTCP.socketReady == true){
              connected = true;
              disconnect = false;
            }
          }

        }

        // Once connection has been made, display editable text
field with
         a button to send that string to the server (see function
below)
        if (myTCP.socketReady == true && connected) {
          if (GUILayout.Button ("Disconnect", GUILayout.Height
(20))) {
            Debug.Log ("Attempting to disconnect..");
            disconnect = true;
            connected = false;
            SendToServer ("Disconnected");
          }
        }
      } else {
        if (GUILayout.Button ("Connect")) {
          // Try to connect
          Debug.Log ("Attempting to connect..");
          myTCP.setupSocket ();
          if(myTCP.socketReady == true){
            initCon = true;
            connected = true;
          }
        }
      }
    }
  }
  // Socket-reading script
  void SocketResponse() {
    string serverSays = myTCP.readSocket();
    if (serverSays != "") {
    }
    if (!serverSays.Contains ("Established") && serverSays != "") {
      serverMsg2Rot (serverSays);
      modifyRotR ();
      applyRotR ();
      if(mirror){
        modifyRotL ();
        applyRotL ();
      }
```

```
    }
  }
  // Send message to the server
  public void SendToServer(string str) {
    myTCP.writeSocket(str);
  }
  // Finding and declaring all bones
  public void findBones(){
    lIndex02 = lIndex01.transform.GetChild(0).gameObject;
    lIndex03 = lIndex02.transform.GetChild(0).gameObject;
    lIndex04 = lIndex03.transform.GetChild(0).gameObject;
    lMiddle02 = lMiddle01.transform.GetChild(0).gameObject;
    lMiddle03 = lMiddle02.transform.GetChild(0).gameObject;
    lMiddle04 = lMiddle03.transform.GetChild(0).gameObject;
    lRing02 = lRing01.transform.GetChild(0).gameObject;
    lRing03 = lRing02.transform.GetChild(0).gameObject;
    lRing04 = lRing03.transform.GetChild(0).gameObject;
    lLittle02 = lLittle01.transform.GetChild(0).gameObject;
    lLittle03 = lLittle02.transform.GetChild(0).gameObject;
    lLittle04 = lLittle03.transform.GetChild(0).gameObject;
    lThumb02 = lThumb01.transform.GetChild(0).gameObject;
    lThumb03 = lThumb02.transform.GetChild(0).gameObject;
    rIndex02 = rIndex01.transform.GetChild(0).gameObject;
    rIndex03 = rIndex02.transform.GetChild(0).gameObject;
    rIndex04 = rIndex03.transform.GetChild(0).gameObject;
    rMiddle02 = rMiddle01.transform.GetChild(0).gameObject;
    rMiddle03 = rMiddle02.transform.GetChild(0).gameObject;
    rMiddle04 = rMiddle03.transform.GetChild(0).gameObject;
    rRing02 = rRing01.transform.GetChild(0).gameObject;
    rRing03 = rRing02.transform.GetChild(0).gameObject;
    rRing04 = rRing03.transform.GetChild(0).gameObject;
    rLittle02 = rLittle01.transform.GetChild(0).gameObject;
    rLittle03 = rLittle02.transform.GetChild(0).gameObject;
    rLittle04 = rLittle03.transform.GetChild(0).gameObject;
    rThumb02 = rThumb01.transform.GetChild(0).gameObject;
    rThumb03 = rThumb02.transform.GetChild(0).gameObject;
  }
  // Assigning the initial values for all of the bones in the hands
  public void setInitVec(){
    // Left side
    lWristInitvec = lWrist.transform.eulerAngles;
    lIndex01Initvec = lIndex01.transform.eulerAngles;
    lIndex02Initvec = lIndex02.transform.eulerAngles;
    lIndex03Initvec = lIndex03.transform.eulerAngles;
    lIndex04Initvec = lIndex04.transform.eulerAngles;
    lMiddle01Initvec = lMiddle01.transform.eulerAngles;
    lMiddle02Initvec = lMiddle02.transform.eulerAngles;
    lMiddle03Initvec = lMiddle03.transform.eulerAngles;
    lMiddle04Initvec = lMiddle04.transform.eulerAngles;
    lRing01Initvec = lRing01.transform.eulerAngles;
    lRing02Initvec = lRing02.transform.eulerAngles;
    lRing03Initvec = lRing03.transform.eulerAngles;
    lRing04Initvec = lRing04.transform.eulerAngles;
    lLittle01Initvec = lLittle01.transform.eulerAngles;
    lLittle02Initvec = lLittle02.transform.eulerAngles;
    lLittle03Initvec = lLittle03.transform.eulerAngles;
```

```
    lLittle04Initvec = lLittle04.transform.eulerAngles;
    lThumb01Initvec = lThumb01.transform.eulerAngles;
    lThumb02Initvec = lThumb02.transform.eulerAngles;
    lThumb03Initvec = lThumb03.transform.eulerAngles;
    // Right side
    rWristInitvec = rWrist.transform.eulerAngles;
    rIndex01Initvec = rIndex01.transform.eulerAngles;
    rIndex02Initvec = rIndex02.transform.eulerAngles;
    rIndex03Initvec = rIndex03.transform.eulerAngles;
    rIndex04Initvec = rIndex04.transform.eulerAngles;
    rMiddle01Initvec = rMiddle01.transform.eulerAngles;
    rMiddle02Initvec = rMiddle02.transform.eulerAngles;
    rMiddle03Initvec = rMiddle03.transform.eulerAngles;
    rMiddle04Initvec = rMiddle04.transform.eulerAngles;
    rRing01Initvec = rRing01.transform.eulerAngles;
    rRing02Initvec = rRing02.transform.eulerAngles;
    rRing03Initvec = rRing03.transform.eulerAngles;
    rRing04Initvec = rRing04.transform.eulerAngles;
    rLittle01Initvec = rLittle01.transform.eulerAngles;
    rLittle02Initvec = rLittle02.transform.eulerAngles;
    rLittle03Initvec = rLittle03.transform.eulerAngles;
    rLittle04Initvec = rLittle04.transform.eulerAngles;
    rThumb01Initvec = rThumb01.transform.eulerAngles;
    rThumb02Initvec = rThumb02.transform.eulerAngles;
    rThumb03Initvec = rThumb03.transform.eulerAngles;
  }
  // Assigning the correct values from the server message to the
rotation
  vectors
  public void serverMsg2Rot(string serverSays){
    lWristvec = new Vector3 (float.Parse (serverSays.Substring (0,
6)),
    float.Parse (serverSays.Substring (7, 6)),
    float.Parse (serverSays.Substring (14, 6)));
    rWristvec = lWristvec;
    lIndex01vec = new Vector3 (float.Parse (serverSays.Substring
(84, 6)),
    float.Parse (serverSays.Substring (91, 6)),
    float.Parse (serverSays.Substring (98, 6)));
    rIndex01vec = lIndex01vec;
    lIndex02vec = new Vector3 (float.Parse (serverSays.Substring
(105, 6)),
    float.Parse (serverSays.Substring (112, 6)),
    float.Parse (serverSays.Substring (119, 6)));
    rIndex02vec = lIndex02vec;
    lIndex03vec = new Vector3 (float.Parse (serverSays.Substring
(126, 6)),
    float.Parse (serverSays.Substring (133, 6)),
    float.Parse (serverSays.Substring (140, 6)));
    rIndex03vec = lIndex03vec;
    lIndex04vec = new Vector3 (float.Parse (serverSays.Substring
(147, 6)),
    float.Parse (serverSays.Substring (154, 6)),
    float.Parse (serverSays.Substring (161, 6)));
    rIndex04vec = lIndex04vec;
```

```
    lMiddle01vec = new Vector3 (float.Parse (serverSays.Substring
(168, 6)),
    float.Parse (serverSays.Substring (175, 6)),
    float.Parse (serverSays.Substring (182, 6)));
    rMiddle01vec = lMiddle01vec;
    lMiddle02vec = new Vector3 (float.Parse (serverSays.Substring
(189, 6)),
    float.Parse (serverSays.Substring (196, 6)),
    float.Parse (serverSays.Substring (203, 6)));
    rMiddle02vec = lMiddle02vec;
    lMiddle03vec = new Vector3 (float.Parse (serverSays.Substring
(210, 6)),
    float.Parse (serverSays.Substring (217, 6)),
    float.Parse (serverSays.Substring (224, 6)));
    rMiddle03vec = lMiddle03vec;
    lMiddle04vec = new Vector3 (float.Parse (serverSays.Substring
(231, 6)),
    float.Parse (serverSays.Substring (238, 6)),
    float.Parse (serverSays.Substring (245, 6)));
    rMiddle04vec = lMiddle04vec;
    lRing01vec = new Vector3 (float.Parse (serverSays.Substring
(252, 6)),
    float.Parse (serverSays.Substring (259, 6)),
    float.Parse (serverSays.Substring (266, 6)));
    rRing01vec = lRing01vec;
    lRing02vec = new Vector3 (float.Parse (serverSays.Substring
(273, 6)),
    float.Parse (serverSays.Substring (280, 6)),
    float.Parse (serverSays.Substring (287, 6)));
    rRing02vec = lRing02vec;
    lRing03vec = new Vector3 (float.Parse (serverSays.Substring
(294, 6)),
    float.Parse (serverSays.Substring (301, 6)),
    float.Parse (serverSays.Substring (308, 6)));
    rRing03vec = lRing03vec;
    lRing04vec = new Vector3 (float.Parse (serverSays.Substring
(315, 6)),
    float.Parse (serverSays.Substring (322, 6)),
    float.Parse (serverSays.Substring (329, 6)));
    rRing04vec = lRing04vec;
    lLittle01vec = new Vector3 (float.Parse (serverSays.Substring
(336, 6)),
    float.Parse (serverSays.Substring (343, 6)),
    float.Parse (serverSays.Substring (350, 6)));
    rLittle01vec = lLittle01vec;
    lLittle02vec = new Vector3 (float.Parse (serverSays.Substring
(357, 6)),
    float.Parse (serverSays.Substring (364, 6)),
    float.Parse (serverSays.Substring (371, 6)));
    rLittle02vec = lLittle02vec;
    lLittle03vec = new Vector3 (float.Parse (serverSays.Substring
(378, 6)),
    float.Parse (serverSays.Substring (385, 6)),
    float.Parse (serverSays.Substring (392, 6)));
    rLittle03vec = lLittle03vec;
```

```
    lLittle04vec = new Vector3 (float.Parse (serverSays.Substring
(399, 6)),
    float.Parse (serverSays.Substring (406, 6)),
    float.Parse (serverSays.Substring (413, 6)));
    rLittle04vec = lLittle04vec;
    lThumb01vec = new Vector3 (float.Parse (serverSays.Substring
(21, 6)),
    float.Parse (serverSays.Substring (28, 6)),
    float.Parse (serverSays.Substring (35, 6)));
    rThumb01vec = lThumb01vec;
    lThumb02vec = new Vector3 (float.Parse (serverSays.Substring
(42, 6)),
    float.Parse (serverSays.Substring (49, 6)),
    float.Parse (serverSays.Substring (56, 6)));
    rThumb02vec = lThumb02vec;
    lThumb03vec = new Vector3 (float.Parse (serverSays.Substring
(63, 6)),
    float.Parse (serverSays.Substring (70, 6)),
    float.Parse (serverSays.Substring (77, 6)));
    rThumb03vec = lThumb03vec;
  }
  // Modifying the recieved vector so that it fits the model
  public void modifyRotL(){
    // Left Wrist
    temp = lWristvec.z;
    //Debug.Log (lWristvec);
    lWristvec.z = lWristvec.x;
    lWristvec.x = lWristvec.y;
    lWristvec.y = temp;
    // Left Thumb 01
    temp = lThumb01vec.z;
    lThumb01vec.z = -lThumb01vec.x;
    lThumb01vec.x = lThumb01vec.y;
    lThumb01vec.y = -temp;
    // Left Thumb 02
    temp = lThumb02vec.z;
    lThumb02vec.z = -lThumb02vec.x;
    lThumb02vec.x = lThumb02vec.y;
    lThumb02vec.y = -temp;
    // Left Thumb 03
    temp = lThumb03vec.y;
    lThumb03vec.y = -lThumb03vec.x;
    lThumb03vec.x = temp;
    // Left Index 01
    temp = lIndex01vec.y;
    lIndex01vec.y = -lIndex01vec.x;
    lIndex01vec.x = temp;
    // Left Index 02
    temp = lIndex02vec.y;
    lIndex02vec.y = -lIndex02vec.x;
    lIndex02vec.x = temp;
    // Left Index 03
    temp = lIndex03vec.y;
    lIndex03vec.y = -lIndex03vec.x;
    lIndex03vec.x = temp;
    // Left Index 04
```

```
    temp = lIndex04vec.y;
    lIndex04vec.y = -lIndex04vec.x;
    lIndex04vec.x = temp;
    // Left Middle 01
    temp = lMiddle01vec.y;
    lMiddle01vec.y = -lMiddle01vec.x;
    lMiddle01vec.x = temp;
    // Left Middle 02
    temp = lMiddle02vec.y;
    lMiddle02vec.y = -lMiddle02vec.x;
    lMiddle02vec.x = temp;
    // Left Middle 03
    temp = lMiddle03vec.y;
    lMiddle03vec.y = -lMiddle03vec.x;
    lMiddle03vec.x = temp;
    // Left Middle 04
    temp = lMiddle04vec.y;
    lMiddle04vec.y = -lMiddle04vec.x;
    lMiddle04vec.x = temp;
    // Left Ring 01
    temp = lRing01vec.y;
    lRing01vec.y = -lRing01vec.x;
    lRing01vec.x = temp;
    // Left Ring 02
    temp = lRing02vec.y;
    lRing02vec.y = -lRing02vec.x;
    lRing02vec.x = temp;
    // Left Ring 03
    temp = lRing03vec.y;
    lRing03vec.y = -lRing03vec.x;
    lRing03vec.x = temp;
    // Left Ring 04
    temp = lRing04vec.y;
    lRing04vec.y = -lRing04vec.x;
    lRing04vec.x = temp;
    // Left Little 01
    temp = lLittle01vec.y;
    lLittle01vec.y = -lLittle01vec.x;
    lLittle01vec.x = temp;
    // Left Little 02
    temp = lLittle02vec.y;
    lLittle02vec.y = -lLittle02vec.x;
    lLittle02vec.x = temp;
    // Left Little 03
    temp = lLittle03vec.y;
    lLittle03vec.y = -lLittle03vec.x;
    lLittle03vec.x = temp;
    // Left Little 04
    temp = lLittle04vec.y;
    lLittle04vec.y = -lLittle04vec.x;
    lLittle04vec.x = temp;
  }
  public void modifyRotR(){
    // Right Wrist
    temp = -rWristvec.z;
    rWristvec.z = rWristvec.x;
```

```
rWristvec.x = -rWristvec.y;
rWristvec.y = temp;
// Right Thumb 01
temp = -rThumb01vec.z;
rThumb01vec.z = -rThumb01vec.x;
rThumb01vec.x = -rThumb01vec.y;
rThumb01vec.y = -temp;
// Right Thumb 02
temp = -rThumb02vec.z;
rThumb02vec.z = -rThumb02vec.x;
rThumb02vec.x = -rThumb02vec.y;
rThumb02vec.y = -temp;
// Right Thumb 03
temp = rThumb03vec.y;
rThumb03vec.y = rThumb03vec.x;
rThumb03vec.x = temp;
// Right Index 01
temp = -rIndex01vec.y;
rIndex01vec.y = rIndex01vec.x;
rIndex01vec.x = temp;
// Right Index 02
temp = -rIndex02vec.y;
rIndex02vec.y = rIndex02vec.x;
rIndex02vec.x = temp;
// Right Index 03
temp = -rIndex03vec.y;
rIndex03vec.y = rIndex03vec.x;
rIndex03vec.x = temp;
// Right Index 04
temp = -rIndex04vec.y;
rIndex04vec.y = rIndex04vec.x;
rIndex04vec.x = temp;
// Right Middle 01
temp = -rMiddle01vec.y;
rMiddle01vec.y = rMiddle01vec.x;
rMiddle01vec.x = temp;
// Right Middle 02
temp = -rMiddle02vec.y;
rMiddle02vec.y = rMiddle02vec.x;
rMiddle02vec.x = temp;
// Right Middle 03
temp = -rMiddle03vec.y;
rMiddle03vec.y = rMiddle03vec.x;
rMiddle03vec.x = temp;
// Right Middle 04
temp = -rMiddle04vec.y;
rMiddle04vec.y = rMiddle04vec.x;
rMiddle04vec.x = temp;
// Right Ring 01
temp = -rRing01vec.y;
rRing01vec.y = rRing01vec.x;
rRing01vec.x = temp;
// Right Ring 02
temp = -rRing02vec.y;
rRing02vec.y = rRing02vec.x;
rRing02vec.x = temp;
```

```
    // Right Ring 03
    temp = -rRing03vec.y;
    rRing03vec.y = rRing03vec.x;
    rRing03vec.x = temp;
    // Right Ring 04
    temp = -rRing04vec.y;
    rRing04vec.y = rRing04vec.x;
    rRing04vec.x = temp;
    // Right Little 01
    temp = -rLittle01vec.y;
    rLittle01vec.y = rLittle01vec.x;
    rLittle01vec.x = temp;
    // Right Little 02
    temp = -rLittle02vec.y;
    rLittle02vec.y = rLittle02vec.x;
    rLittle02vec.x = temp;
    // Right Little 03
    temp = -rLittle03vec.y;
    rLittle03vec.y = rLittle03vec.x;
    rLittle03vec.x = temp;
    // Right Little 04
    temp = -rLittle04vec.y;
    rLittle04vec.y = rLittle04vec.x;
    rLittle04vec.x = temp;
  }
  // Applying the rotation sent from the server
  public void applyRotL(){
    lWrist.transform.Rotate(lWristvec);
    lIndex01.transform.Rotate(lIndex01vec);
    lIndex02.transform.Rotate(lIndex02vec);
    lIndex03.transform.Rotate(lIndex03vec);
    lIndex04.transform.Rotate(lIndex04vec);
    lMiddle01.transform.Rotate(lMiddle01vec);
    lMiddle02.transform.Rotate(lMiddle02vec);
    lMiddle03.transform.Rotate(lMiddle03vec);
    lMiddle04.transform.Rotate(lMiddle04vec);
    lRing01.transform.Rotate(lRing01vec);
    lRing02.transform.Rotate(lRing02vec);
    lRing03.transform.Rotate(lRing03vec);
    lRing04.transform.Rotate(lRing04vec);
    lLittle01.transform.Rotate(lLittle01vec);
    lLittle02.transform.Rotate(lLittle02vec);
    lLittle03.transform.Rotate(lLittle03vec);
    lLittle04.transform.Rotate(lLittle04vec);
    lThumb01.transform.Rotate(lThumb01vec);
    lThumb02.transform.Rotate(lThumb02vec);
    lThumb03.transform.Rotate(lThumb03vec);
  }
  public void applyRotR(){
    rWrist.transform.Rotate(rWristvec);
    rIndex01.transform.Rotate(rIndex01vec);
    rIndex02.transform.Rotate(rIndex02vec);
    rIndex03.transform.Rotate(rIndex03vec);
    rIndex04.transform.Rotate(rIndex04vec);
    rMiddle01.transform.Rotate(rMiddle01vec);
    rMiddle02.transform.Rotate(rMiddle02vec);
```

```
    rMiddle03.transform.Rotate(rMiddle03vec);
    rMiddle04.transform.Rotate(rMiddle04vec);
    rRing01.transform.Rotate(rRing01vec);
    rRing02.transform.Rotate(rRing02vec);
    rRing03.transform.Rotate(rRing03vec);
    rRing04.transform.Rotate(rRing04vec);
    rLittle01.transform.Rotate(rLittle01vec);
    rLittle02.transform.Rotate(rLittle02vec);
    rLittle03.transform.Rotate(rLittle03vec);
    rLittle04.transform.Rotate(rLittle04vec);
    rThumb01.transform.Rotate(rThumb01vec);
    rThumb02.transform.Rotate(rThumb02vec);
    rThumb03.transform.Rotate(rThumb03vec);
  }
  public void resetFingerRotation() {
    lWrist.transform.eulerAngles = lWristInitvec;
    lIndex01.transform.eulerAngles = lIndex01Initvec;
    lIndex02.transform.eulerAngles = lIndex02Initvec;
    lIndex03.transform.eulerAngles = lIndex03Initvec;
    lIndex04.transform.eulerAngles = lIndex04Initvec;
    lMiddle01.transform.eulerAngles = lMiddle01Initvec;
    lMiddle02.transform.eulerAngles = lMiddle02Initvec;
    lMiddle03.transform.eulerAngles = lMiddle03Initvec;
    lMiddle04.transform.eulerAngles = lMiddle04Initvec;
    lRing01.transform.eulerAngles = lRing01Initvec;
    lRing02.transform.eulerAngles = lRing02Initvec;
    lRing03.transform.eulerAngles = lRing03Initvec;
    lRing04.transform.eulerAngles = lRing04Initvec;
    lLittle01.transform.eulerAngles = lLittle01Initvec;
    lLittle02.transform.eulerAngles = lLittle02Initvec;
    lLittle03.transform.eulerAngles = lLittle03Initvec;
    lLittle04.transform.eulerAngles = lLittle04Initvec;
    lThumb01.transform.eulerAngles = lThumb01Initvec;
    lThumb02.transform.eulerAngles = lThumb02Initvec;
    lThumb03.transform.eulerAngles = lThumb03Initvec;
    if (enableMyo) {
      lWrist.transform.Rotate(new Vector3(0.0f, 90.0f, 0.0f));
    }
    // Right
    rWrist.transform.eulerAngles = rWristInitvec;
    rIndex01.transform.eulerAngles = rIndex01Initvec;
    rIndex02.transform.eulerAngles = rIndex02Initvec;
    rIndex03.transform.eulerAngles = rIndex03Initvec;
    rIndex04.transform.eulerAngles = rIndex04Initvec;
    rMiddle01.transform.eulerAngles = rMiddle01Initvec;
    rMiddle02.transform.eulerAngles = rMiddle02Initvec;
    rMiddle03.transform.eulerAngles = rMiddle03Initvec;
    rMiddle04.transform.eulerAngles = rMiddle04Initvec;
    rRing01.transform.eulerAngles = rRing01Initvec;
    rRing02.transform.eulerAngles = rRing02Initvec;
    rRing03.transform.eulerAngles = rRing03Initvec;
    rRing04.transform.eulerAngles = rRing04Initvec;
    rLittle01.transform.eulerAngles = rLittle01Initvec;
    rLittle02.transform.eulerAngles = rLittle02Initvec;
    rLittle03.transform.eulerAngles = rLittle03Initvec;
    rLittle04.transform.eulerAngles = rLittle04Initvec;
```

```
    rThumb01.transform.eulerAngles = rThumb01Initvec;
    rThumb02.transform.eulerAngles = rThumb02Initvec;
    rThumb03.transform.eulerAngles = rThumb03Initvec;
  }
}
```

TCPkod

Denna kod sköter socketen som används för kommunikation med Matlab. Koden är baserad på exempelkod hämtad från https://girlscancode.wordpress.com/2015/02/01/update-unity-and-tcpip-socket-connections/.

```
using UnityEngine;
using System.Collections;
using System;
using System.IO;
using System.Net.Sockets;
public class TCPConnection : MonoBehaviour {
  // The name of the connection, not required but better for
overview if you have
  more than 1 connections running
  public string serverName = "Localhost";
  public GameObject humanModel = null;
  // IP-address of the server, 127.0.0.1 is for your own computer
  private string serverIP = "127.0.0.1";
  // Port for the server, make sure to unblock this in your router
firewall if you
  want to allow external connections
  private int serverPort = 125;
  // A boolean for connection status
  public bool socketReady = false;
  TcpClient mySocket;
  NetworkStream theStream;
  StreamWriter theWriter;
  StreamReader theReader;
  void Start () {
  }
  // Try to initiate connection
  public void setupSocket() {
    try {
      mySocket = new TcpClient(serverIP, serverPort);
      theStream = mySocket.GetStream();
      theWriter = new StreamWriter(theStream);
      theReader = new StreamReader(theStream);
      socketReady = true;
    }
    catch (Exception e) {
      Debug.Log("Socket error:" + e);
    }
  }
  // Send message to server
  public void writeSocket(string theLine) {
    if (!socketReady)
      return;
```

```
        String tmpString = theLine + "\r\n";
        theWriter.Write(tmpString);
        theWriter.Flush();
    }
    // Read message from server
    public string readSocket() {
        String result = "";
        if (theStream.DataAvailable) {
            Byte[] inStream = new Byte[mySocket.SendBufferSize];
            theStream.Read(inStream, 0, inStream.Length);
            result += System.Text.Encoding.UTF8.GetString(inStream);
        }
        return result;
    }
    // Disconnect from the socket
    public void closeSocket() {
        if (!socketReady)
            return;
        theWriter.Close();
        theReader.Close();
        mySocket.Close();
        socketReady = false;
    }
    // Keep connection alive, reconnect if connection lost
    public void maintainConnection(){
        if(!theStream.CanRead) {
            setupSocket();
        }
    }
}
```

Myokod

Denna lpd sköter avläsningen av Myo:n och roterar armen utifrån detta. Koden vibrerar även Myo:n beroende på vilket objekt som vidrörs och knyter handen respektive öppnar handen om CyberGlove:n inte är aktiverad. Koden är baserat på en testuppsättning som ingår i Thalmics SDK för Myo.

```
using UnityEngine;
using System.Collections;
using LockingPolicy = Thalmic.Myo.LockingPolicy;
using Pose = Thalmic.Myo.Pose;
using UnlockType = Thalmic.Myo.UnlockType;
using VibrationType = Thalmic.Myo.VibrationType;
// Orient the object to match that of the Myo armband.
// Compensate for initial yaw (orientation about the gravity
vector) and
// roll (orientation about the wearer's arm) by allowing the
user to set a
// reference orientation.
// Pressing the 'r' key resets the reference orientation.
public class JointOrientation : MonoBehaviour
```

```
{
  // Myo game object to connect with.
  // This object must have a ThalmicMyo script attached.
  public GameObject myo = null;
  public GameObject lForearm = null;
  public GameObject rForearm = null;
  public GameObject lIndex02 = null;
  public GameObject lMiddle02 = null;
  public GameObject lRing02 = null;
  public GameObject lLittle02 = null;
  public GameObject lThumb02 = null;
  public GameObject rIndex02 = null;
  public GameObject rMiddle02 = null;
  public GameObject rRing02 = null;
  public GameObject rLittle02 = null;
  public GameObject rThumb02 = null;
  public GameObject humanModel = null;
  private float fingerRotx, fingerRotz, fingerRoty;
  private float tempOrientX;
  private bool handClosed = false;
  private Transform[] allChildren;
  private bool mirror;
  private bool enableCyberGlove;
  private bool enableMyo;
  // A rotation that compensates for the Myo armband's
orientation parallel
  // to the ground, i.e. yaw.
  // Once set, the direction the Myo armband is facing
becomes "forward"
  // within the program.
  // Set by making the fingers spread pose or pressing "r".
  private Quaternion _antiYaw = Quaternion.identity;
  private Quaternion _lantiYaw = Quaternion.identity;
  // A reference angle (degrees) representing how the
armband is rotated
  // about the wearer's arm, i.e. roll. Set by pressing "r".
  private float _referenceRoll = 0.0f;
  // The pose from the last update. This is used to
determine if the pose
  // has changed so that actions are only performed upon
making them rather
  // than every frame during which they are active.
  private Pose _lastPose = Pose.Unknown;
  private bool updateReference;
  private bool firstRun;
  void Start(){
    enableMyo =
humanModel.GetComponent<Controls>().returnEnableMyo ();
    mirror =
humanModel.GetComponent<Controls>().returnEnableMirroring();
```

```
    enableCyberGlove =
humanModel.GetComponent<Controls>().returnEnableCyberGlove()
;
    fingerRotx =
humanModel.GetComponent<Controls>().returnfingerRotx();
    fingerRoty =
humanModel.GetComponent<Controls>().returnfingerRoty();
    fingerRotz =
humanModel.GetComponent<Controls>().returnfingerRotz();
    if (mirror) {
      lForearm.transform.eulerAngles = new Vector3 (0.0f,
0.0f, 0.0f);
      lForearm.transform.parent.eulerAngles = new Vector3
(0.0f, 75.0f,
      10.0f);
    } else {
      lForearm.transform.parent.eulerAngles = new Vector3
(0.0f, 20.0f,
      70.0f);
      lForearm.transform.eulerAngles = new Vector3 (0.0f,
80.0f, 40.0f);
    }
    if (enableMyo) {
      firstRun = true;
    } else if (enableMyo == false && mirror == true){
      Vector3 temp = rForearm.transform.eulerAngles;
      temp.y += 10.0f;
      //temp.x -= 90.0f;
      lForearm.transform.eulerAngles = temp;
    }
  }
  // Update is called once per frame.
  void Update ()
  {
    if (enableMyo) {
      updateReference = false;
      // Access the ThalmicMyo component attached to the Myo
object.
      ThalmicMyo thalmicMyo = myo.GetComponent<ThalmicMyo>
();
      // Update references when the q key is pressed.
      if(!enableCyberGlove){
        if (thalmicMyo.pose != _lastPose && enableCyberGlove
== false) {
          _lastPose = thalmicMyo.pose;
          if (thalmicMyo.pose == Pose.FingersSpread &&
handClosed == true){
            handClosed = false;
            // Rotating one finger at a time
```

```
            // Index
            allChildren = rIndex02.GetComponentsInChildren
<Transform> ();
            foreach (Transform child in allChildren) {
              child.transform.Rotate (-fingerRotx, -
fingerRoty,
              -fingerRotz);
            }
            // Middle
            allChildren = rMiddle02.GetComponentsInChildren
<Transform> ();
            foreach (Transform child in allChildren) {
              child.transform.Rotate (-fingerRotx, -
fingerRoty,
              -fingerRotz);
            }
            // Ring
            allChildren = rRing02.GetComponentsInChildren
<Transform> ();
            foreach (Transform child in allChildren) {
              child.transform.Rotate (-fingerRotx, -
fingerRoty,
              -fingerRotz);
            }
            // Little
            allChildren = rLittle02.GetComponentsInChildren
<Transform> ();
            foreach (Transform child in allChildren) {
              child.transform.Rotate (-fingerRotx, -
fingerRoty,
               -fingerRotz);
            }
            // Thumb
            allChildren = rThumb02.GetComponentsInChildren
<Transform> ();
            foreach (Transform child in allChildren) {
              child.transform.Rotate (-fingerRotx, -
fingerRoty,
               -fingerRotz);
            }
            rThumb02.transform.Rotate (0.0f, 50.0f, 0.0f);
            rThumb02.transform.parent.Rotate (0.0f, 10.0f,
15.0f);
            if (mirror) {
              // Rotating one finger at a time for the left
arm
              // Index
              allChildren =
lIndex02.GetComponentsInChildren<Transform>();
              foreach (Transform child in allChildren) {
```

```
                child.transform.Rotate (fingerRotx,
fingerRoty,
                fingerRotz);
              }
              // Middle
              allChildren =
lMiddle02.GetComponentsInChildren<Transform>();
              foreach (Transform child in allChildren) {
                child.transform.Rotate (fingerRotx,
fingerRoty,
                fingerRotz);
              }
              // Ring
              allChildren = lRing02.GetComponentsInChildren
<Transform> ();
              foreach (Transform child in allChildren) {
                child.transform.Rotate (fingerRotx,
fingerRoty,
                fingerRotz);
              }
              // Little
              allChildren =
lLittle02.GetComponentsInChildren<Transform>();
              foreach (Transform child in allChildren) {
                child.transform.Rotate (fingerRotx,
fingerRoty,
                fingerRotz);
              }
              // Thumb
              allChildren =
lThumb02.GetComponentsInChildren<Transform>();
              foreach (Transform child in allChildren) {
                child.transform.Rotate (fingerRotx,
fingerRoty,
                fingerRotz);
              }
              lThumb02.transform.Rotate (0.0f, -50.0f,
0.0f);
              lThumb02.transform.parent.Rotate (0.0f, -
10.0f, 15.0f);
            }
            ExtendUnlockAndNotifyUserAction (thalmicMyo);
          }
          if (thalmicMyo.pose == Pose.Fist && handClosed ==
false) {
            handClosed = true;
            // Rotating one finger at a time
            // Index
            allChildren = rIndex02.GetComponentsInChildren
<Transform> ();
```

```
            foreach (Transform child in allChildren) {
              child.transform.Rotate (fingerRotx,
fingerRoty, fingerRotz);
            }
            // Middle
            allChildren = rMiddle02.GetComponentsInChildren
<Transform> ();
            foreach (Transform child in allChildren) {
              child.transform.Rotate (fingerRotx,
fingerRoty, fingerRotz);
            }
            // Ring
            allChildren = rRing02.GetComponentsInChildren
<Transform> ();
            foreach (Transform child in allChildren) {
              child.transform.Rotate (fingerRotx,
fingerRoty, fingerRotz);
            }
            // Little
            allChildren = rLittle02.GetComponentsInChildren
<Transform> ();
            foreach (Transform child in allChildren) {
              child.transform.Rotate (fingerRotx,
fingerRoty, fingerRotz);
            }
            // Thumb
            rThumb02.transform.parent.Rotate (0.0f, -10.0f,
-15.0f);
            rThumb02.transform.Rotate (0.0f, -50.0f, 0.0f);
            allChildren = rThumb02.GetComponentsInChildren
<Transform> ();
            foreach (Transform child in allChildren) {
              child.transform.Rotate (fingerRotx,
fingerRoty, fingerRotz);
            }
            if (mirror) {
              // Rotating one finger at a time
              // Index
              allChildren =
lIndex02.GetComponentsInChildren<Transform>();
              foreach (Transform child in allChildren) {
                child.transform.Rotate (-fingerRotx, -
fingerRoty,
                 -fingerRotz);
              }
              // Middle
              allChildren =
lMiddle02.GetComponentsInChildren<Transform>();
              foreach (Transform child in allChildren) {
```

```
                child.transform.Rotate (-fingerRotx, -
fingerRoty,
                -fingerRotz);
              }
              // Ring
              allChildren =
lRing02.GetComponentsInChildren<Transform>();
              foreach (Transform child in allChildren) {
                child.transform.Rotate (-fingerRotx, -
fingerRoty,
                -fingerRotz);
              }
              // Little
              allChildren =
lLittle02.GetComponentsInChildren<Transform>();
              foreach (Transform child in allChildren) {
                child.transform.Rotate (-fingerRotx, -
fingerRoty,
                -fingerRotz);
              }
              // Thumb
              lThumb02.transform.parent.Rotate (0.0f, 10.0f,
-15.0f);
              lThumb02.transform.Rotate (0.0f, 50.0f, 0.0f);
              allChildren =
lThumb02.GetComponentsInChildren<Transform>();
              foreach (Transform child in allChildren) {
                child.transform.Rotate (-fingerRotx, -
fingerRoty,
               -fingerRotz);
              }
            }
            ExtendUnlockAndNotifyUserAction (thalmicMyo);
          }
        }
      }
      if (Input.GetKeyDown ("r")) {
        updateReference = true;
      }
      // Update references. This anchors the joint on-screen
such that it
      // faces forward away from the viewer when the Myo
armband is
      // oriented the way it is when these references are
taken.
      if (updateReference || firstRun) {
          // _antiYaw represents a rotation of the Myo
armband about the Y
          // axis (up) which aligns the forward vector of
the rotation with
```

```
            // Z = 1 when the wearer's arm is pointing in the
reference
            // direction.
            _antiYaw = Quaternion.FromToRotation (
            new Vector3 (myo.transform.forward.x, 0,
            myo.transform.forward.z), new Vector3 (0, 0, -1)
            );
            _lantiYaw = Quaternion.FromToRotation (
            new Vector3 (myo.transform.forward.x, 0,
            myo.transform.forward.z), new Vector3 (-1, 0, 0)
            );
            // _referenceRoll represents how many degrees the
Myo armband is
            // rotated clockwise about its forward axis (when
looking down
            // the wearer's arm towards their hand) from the
reference zero
            // roll direction. This direction is calculated
and explained
            // below. When this reference is taken, the joint
will be rotated
            // about its forward axis such that it faces
upwards when
            // the roll value matches the reference.
            Vector3 referenceZeroRoll = computeZeroRollVector
            (myo.transform.forward);
            _referenceRoll = rollFromZero (referenceZeroRoll,
            myo.transform.forward, myo.transform.up);
        }
        // Current zero roll vector and roll value.
        Vector3 zeroRoll = computeZeroRollVector
(myo.transform.forward);
        float roll = rollFromZero (zeroRoll,
myo.transform.forward,
        myo.transform.up);
        // The relative roll is simply how much the current
roll has changed
        // relative to the reference roll.
        // adjustAngle simply keeps the resultant value within
-180 to 180
        // degrees.
        float relativeRoll = normalizeAngle (roll -
_referenceRoll);
        // antiRoll represents a rotation about the myo
Armband's forward
        // axis adjusting for reference roll.
        Quaternion antiRoll = Quaternion.AngleAxis
(relativeRoll,
        myo.transform.forward);
```

```
      // Here the anti-roll and yaw rotations are applied to
the myo
      // Armband's forward direction to yield
      // the orientation of the joint.
      var orientation = _antiYaw * antiRoll *
Quaternion.LookRotation
      (myo.transform.forward);
      tempOrientX = orientation.x;
      orientation.x = -orientation.z;
      orientation.z = tempOrientX;
      rForearm.transform.rotation = orientation;
      // The above calculations were done assuming the Myo
armbands's +x
      // direction, in its own coordinate system,
      // was facing toward the wearer's elbow. If the Myo
armband is worn
      // with its +x direction facing the other way,
      // the rotation needs to be updated to compensate.
      if (thalmicMyo.xDirection ==
Thalmic.Myo.XDirection.TowardWrist) {
        // Mirror the rotation around the XZ plane in
Unity's coordinate
        system (XY plane in Myo's coordinate
        // system). This makes the rotation reflect the
arm's orientation,
        rather than that of the Myo armband.
        rForearm.transform.rotation = new Quaternion
        (transform.localRotation.x, -
transform.localRotation.y,
        transform.localRotation.z, -
transform.localRotation.w);
      }
      // Mirroring if the mirror-boolean is true
      if (mirror) {
        orientation = _lantiYaw * antiRoll *
Quaternion.LookRotation
        (myo.transform.forward);
        tempOrientX = orientation.x;
        orientation.x = -orientation.z;
        orientation.z = -tempOrientX;
        orientation.y = -orientation.y;
        lForearm.transform.rotation = orientation;
      }
      firstRun = false;
     }
    }
    // Compute the angle of rotation clockwise about the
forward axis
    // relative to the provided zero roll direction.
```

```
    // As the armband is rotated about the forward axis this
value will
    // change, regardless of which way the
    // forward vector of the Myo is pointing. The returned
value will be
    // between -180 and 180 degrees.
    float rollFromZero (Vector3 zeroRoll, Vector3 forward,
Vector3 up){
        // The cosine of the angle between the up vector and
the zero roll
        // vector. Since both are orthogonal to the forward
vector, this
        // tells us how far the Myo has been turned around
the forward axis
        // relative to the zero roll vector, but we need to
determine
        // separately whether the Myo has been rolled
clockwise or
        // counterclockwise.
        float cosine = Vector3.Dot (up, zeroRoll);
        // To determine the sign of the roll, we take the
cross product of
        // the up vector and the zero roll vector. This
cross product will
        // either be the same or       opposite direction as
the forward
        // vector depending on whether up is clockwise or
counter-clockwise
        // from zero roll.
        // Thus the sign of the dot product of forward and
it yields the
        sign of our roll value.
        Vector3 cp = Vector3.Cross (up, zeroRoll);
        float directionCosine = Vector3.Dot (forward, cp);
        float sign = directionCosine < 0.0f ? 1.0f : -1.0f;
        // Return the angle of roll (in degrees) from the
cosine and the
        // sign.
        return sign * Mathf.Rad2Deg * Mathf.Acos (cosine);
    }
    // Compute a vector that points perpendicular to the
forward direction,
    // minimizing angular distance from world up (positive Y
axis).
    // This represents the direction of no rotation about
its forward axis.
    Vector3 computeZeroRollVector (Vector3 forward)
    {
        Vector3 antigravity = Vector3.up;
```

```
        Vector3 m = Vector3.Cross (myo.transform.forward,
antigravity);
        Vector3 roll = Vector3.Cross (m,
myo.transform.forward);
        return roll.normalized;
    }
    // Adjust the provided angle to be within a -180 to 180.
    float normalizeAngle (float angle)
    {
    float correction = 0.0f;
        if (angle > 180.0f + correction) {
            return angle - 360.0f;
        }
        if (angle < -180.0f + correction) {
            return angle + 360.0f;
        }
        return angle;
    }
    // Extend the unlock if ThalmcHub's locking policy is
standard, and
    // notifies the given myo that a user action was
recognized.
    void ExtendUnlockAndNotifyUserAction (ThalmicMyo myo)
    {
        ThalmicHub hub = ThalmicHub.instance;
        if (hub.lockingPolicy == LockingPolicy.Standard) {
            myo.Unlock (UnlockType.Timed);
        }
    myo.Vibrate (VibrationType.Short);
    }
}
```

Vibrationskod

Denna kod vibrerar Myo:n olika länge beroende på vilken trigger som handen träffar.

För citronen:

```
using UnityEngine;
using System.Collections;
using VibrationType = Thalmic.Myo.VibrationType;
public class LemonTrigger : MonoBehaviour {
  public GameObject myo = null;
  public int vibrationStrength = 1;
  public bool feedbackOn = true;
  void OnTriggerEnter(Collider other) {
    if (feedbackOn) {
      ThalmicMyo thalmicMyo = myo.GetComponent<ThalmicMyo> ();
      ExtendUnlockAndNotifyUserAction (thalmicMyo,
vibrationStrength);
    }
```

```
  }
  void ExtendUnlockAndNotifyUserAction (ThalmicMyo myo, int a)
  {
    // a = 0 results in  a short vibration
    // a = 1 results in  medium vibration
    // a = 2 results in  long vibration
    if (a.Equals (0)) {
      myo.Vibrate (VibrationType.Short);
    } else if (a.Equals (1)) {
      myo.Vibrate (VibrationType.Medium);
    } else if (a.Equals (2)) {
      myo.Vibrate (VibrationType.Long);
    }
  }
}
```

För lejonstatyetten:

```
using UnityEngine;
using System.Collections;
using VibrationType = Thalmic.Myo.VibrationType;
public class LionTrigger : MonoBehaviour {
  public GameObject myo = null;
  public int vibrationStrength = 2;
  public bool feedbackOn = true;
  void OnTriggerEnter(Collider other) {
    if (feedbackOn) {
      ThalmicMyo thalmicMyo = myo.GetComponent<ThalmicMyo> ();
      ExtendUnlockAndNotifyUserAction (thalmicMyo,
vibrationStrength);
    }
  }
  void ExtendUnlockAndNotifyUserAction (ThalmicMyo myo, int a)
  {
    // a = 0 leads to a short vibration
    // a = 1 leads to a medium vibration
    // a = 2 leads to a long vibration
    if (a.Equals (0)) {
      myo.Vibrate (VibrationType.Short);
    } else if (a.Equals (1)) {
      myo.Vibrate (VibrationType.Medium);
    } else if (a.Equals (2)) {
      myo.Vibrate (VibrationType.Long);
    }
  }
}
```

Appendix V – Förstudie av spelmotor

Jämförelse

Spelmotor	**Unity**	**CryEngine**	**Unreal Engine 4**
Pris – full version	1500$ eller 75$ i månaden.	9,9$ i månaden.	19$ i månaden Sänktes till 4$ i månaden under arbetets gång.
Studentpris.	605 euros (endast på en dator) via studica.	Gratis via CryEngine:s hemsida.	Gratis via Unreal:s hemsida.
Gratis Licens	Skillnader mellan pro och free: http://unity3d.com/unity/licenses	-	-
Video som visar utseendet.	http://nvyve.com/our-work/hirsh-log-homes/ Galleri: http://unity3d.com/showcase/gallery	http://www.cryengine.com/features	https://www.youtube.com/watch?v=slCa5v0sDsM
TCP/IP	Ja.	Verkar vara krångligt.	Verkar vara begränsat: http://www.cryengine.com/community/viewtopic.php?f=291&t=105280
Rangordning av antal användare	3	2	1

En hemsida med modeller som kan importeras till samtliga motorer: http://www.turbosquid.com/

Slutsats

CryEngine är för krångligt att använda, och hade tagit alldeles för lång tid att lära sig. Dessutom verkar man behöva gå ner på kodnivå, vilket består av minst 2 språk, för att få det att fungera. Verkar vara optimerat för ”First Person Shooter” - FPS.

Unity fungerar bra och jag har använt det innan. Ett plug-in kan sköta kommunikationen mellan Unity och Matlab.

Unreal Engine ser snyggare ut än Unity, men jag har inte använt det innan. Det finns gratis för studenter, och jag tänkte testa arbeta lite i det.

Unity eller Unreal Engine kommer användas för projektet. Vilken av dessa som väljs kommer bestämmas senare i projektet, då båda testats.