

令和 5 年度

卒業研究報告書

仮想筋電義手の開発に関する研究

指導教員 戸崎 哲也
報告者 河合 将暉

神戸市立工業高等専門学校
電子工学科

(論文要旨)

上肢切断者が筋電義手を装着し、日常生活で扱えるようになるには相応の訓練が必要とされる。この筋電義手の訓練を代替するものとして、仮想筋電義手シミュレータが先行研究によって開発されてきた。本研究では、先行研究で一般的に使用されていた配布されている仮想筋電義手3Dモデルとは異なる3Dスキャナ“EinScan HX”を用いた仮想筋電義手3Dモデルを使用することにより筋電義手装着訓練の訓練効果に影響があるかを検討した。また、入力インターフェースにFirstVRを採用することによる訓練効果への影響を検討するために、シミュレータのプロトタイプを提案した。手法として、3Dスキャナで実際の左腕から3Dモデルを取得し、ノイズ成分をBlenderのスムージング機能を用いて3Dモデル表面の平滑化を行った。その3Dモデルにボーン配置を行い、Unityにインポートすることで物体保持アニメーションを作製し、筋電義手の動作と似た動作をするようなシステム構成にした。ここで、キーボード・マウスを入力インターフェースとしたPC版のシミュレータでは実際の上肢切断者が操作できないという課題点を“FirstVR”を導入することで解決した。FirstVRは腕に巻くだけで3軸ジャイロセンサや加速度センサにより腕の位置をトラッキングすることが可能であり、加えて筋変位センサによるジェスチャ認識機能も備わっている。これにより、仮想筋電義手シミュレータが実際の上肢切断者でも扱えるようになった上に、筋変位によるジェスチャ認識のしきい値を実際の筋電義手に近づけることにより訓練効果の向上を見込めると考えた。

本研究では前段階として、FirstVRのジェスチャ認識率、最適サンプル数の調査と、健常者を対象にした6段階リッカート尺度を用いた定性評価アンケート調査を行い、入力インターフェースをキーボード・マウスとしたシステムとFirstVRのシステムとを比較した。まず、FirstVRのジェスチャ認識率は実験協力者各個人による分散が大きく、最高値は100%，最低値は50%と大幅に差ができてしまっている。おそらくジェスチャ認識率が70%以下の場合では複合的な要因が考えられ、装着位置や各個人の筋肉量の違いによって差ができるていると考察した。このため、sample数を比較する際に個人差が含まれないように80%未満のデータを除外すると、どのsample数においても $96.6 \pm 3.34\%$ のジェスチャ認識率が結果として得られた。これより、ジェスチャ認識率では最適sample数を導くことはできなかった。これを踏まえて、評価指標を独自に導入することにより、各筋変位センサの測定値ごとに比較し、実験協力者による分散がもっとも少なく、データの学習量が小さいものを最適sample数とした。この評価指標により、sample数7が最適であるとして、シミュレータを構成した。次に、定性評価アンケート調査ではFirstVRのシステムがより没入感が高いという結果が得られた。しかし、操作性や入力遅延といった点では優位性がみられることはなく、さらにFirstVRの装着位置・ジェスチャ学習のプロセスなどの条件を加えて改善することが求められる。

目次

第1章: はじめに	1
1.1 研究背景	1
1.2 関連研究	1
1.3 研究目的	2
第2章: 3D モデルの製作	3
2.1 EinScan HX	3
2.1.1 製品仕様	3
2.2 Blender	4
2.2.1 スムージング	4
2.2.2 ポーン構成	4
2.3 Unity	4
2.3.1 シェーダー	5
2.3.2 衝突判定	5
2.3.3 アニメーション	6
2.3.4 ビルド	6
2.4 FirstVR	8
2.4.1 デバイス構成	8
2.4.2 トラッキング	8
2.4.3 ジェスチャ認識/キャリブレーション	8
第3章: 研究手順	9
3.1 使用器具	9
3.2 VH の 3D モデル取り込み	9
3.3 VH のスムージング処理・ポーン配置	9
3.4 オブジェクト保持表現	9
3.5 FirstVR の性能評価手法	10
3.6 シミュレータの構成	11
3.7 シミュレータの定性評価手法	11
第4章: 研究結果	12
4.1 VH (Virtual Hand) の 3D モデル構成	12
4.2 オブジェクト保持	13
4.3 FirstVR の性能評価	15
4.4 シミュレータの構成	21
4.5 シミュレータの定性評価	23
第5章: 考察	25
第6章: おわりに	26
謝辞	27
参考文献	28
付録	i
A.1 測定データ	i
A.2 ソースコード	i

第 1 章 はじめに

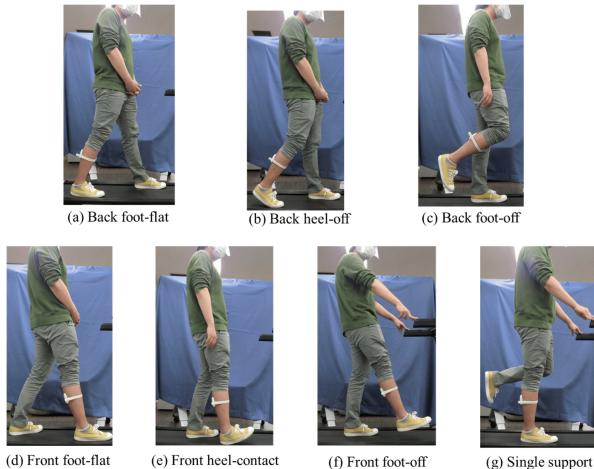
1.1 研究背景

上肢切断者が筋電義手を装着する際、自在に扱うことができるよう訓練を行う必要がある。VR を用いた筋電義手トレーニングの効果については先行研究 [1][2] で検討されており、筋電義手操作の幻肢痛緩和や効果があると立証されている。これらの研究では仮想筋電義手の 3D モデルについては長方形のオブジェクトや、腕を模して製作された 3D モデルを用いることが一般的であった。また、近年では“カグラ”[3] という医療機器は上肢機能障害者のリハビリテーションのために運用されるなど VR トレーニングシステムは義手装具者以外にも需要が高まっている。本研究は 2023 年度、神戸高専 Digital Fabrication Lab に新規導入されたハンディ 3D スキャナ “Ein Scan HX”[4] の活用方法の一例として後続の研究に利用していただきたい。

本論文の構成として第 2 章、第 3 章で 3D モデルの製作や Unity・FirstVR について解説を行う。そして、第 4 章に研究手法を提示し、第 5 章に研究結果を示す。

1.2 関連研究

FirstVR を用いた関連研究としては足に FirstVR を装着することで足の筋変位による足の接地状態の検出を行う研究 [6] が行われている。この研究では図 1.1 の歩行状態を 7 段階に分割した姿勢を FirstVR にそれぞれ学習させ、加えてジャイロセンサを用いてどの姿勢が足の接地状態（立脚期）の検出に最適なのかを調査している。

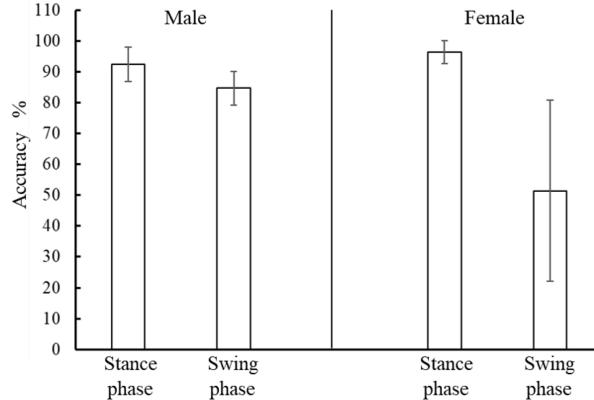


文献 [6] より引用

図 1.1: 歩行位相検出アルゴリズムのキャリブレーション姿勢

文献 [6] によると、足の浮遊状態（遊脚期）の検出には体軸よりも足が後ろになっている状態が検出精度が高くなる傾向にあるということが報告されており、その姿勢状態を学習させて実験協力者 10 名（男性 6 名、女性 4 名）にトレッドミルを用いて歩行データを計測されていた。

文献 [6] より引用した男女別の遊脚期 (Swing phase) と立脚期 (Stance phase) の陽性率を図 1.2 に示す.



文献 [6] より引用

図 1.2: 男性 (左) と女性 (右) の立脚期と遊脚期の陽性率

図 1.2 の結果より, 立脚期の検出精度中央値は 90 %であり, 歩行位相検出システムとして十分な結果を得られている. しかし, この方式では女性の遊脚期の検出精度が低いことから個人の筋肉量に検出精度が依存しているといった課題がこの研究によって報告されている.

1.3 研究目的

本研究では, 仮想筋電義手モデルのリアリティによる訓練効果や幻肢痛緩和効果に着目し 3D スキャナで取り込んだ仮想筋電義手 3D モデル (VH:Virtual Hand) を用いた VR トレーニングシステムを構成し, VH の見た目によって訓練効果に変化があるかを評価することを目的とする. 加えて, このトレーニングシステムのインターフェースに “FirstVR”[5] を用いることで, 実際の筋電義手や高価な筋電センサ使用することなく, 安価で訓練が行えると考えたため, FirstVR による訓練効果の影響を調べるために前段階として性能評価や定性評価を行い, 入力インターフェースによる没入感や操作性などの違いを調査することにした.

第 2 章 3D モデルの製作

本章では、3D モデル製作に使用した “Ein Scan HX” と “Blender” について解説する。また、それぞれの役割と機能についても説明を加える。

2.1 EinScan HX

EinScan HX は株式会社サンステラが提供するハンディ 3D スキャナである。主な使用用途としては、工業製品などの比較的大きな物をスキャンし、リバースエンジニアリングや測定などに用いられている。図 2.1 に EinScan の外観を示す。



図 2.1: EinScan HX

2.1.1 製品仕様

文献 [4] から引用した EinScan の製品仕様を図 2.1 に示す。

表 2.1: EinScan HX の製品仕様

スキャン形式	Rapid スキャン	レーザースキャン
スキャン精度	0.05 mm	0.04 mm
ポイント間隔	0.25~3.00 mm	0.05~3.00 mm
被写体長 3D 精度	±0.1 mm	±0.06 mm
シングルスキャン精度	420 × 440 mm	380 × 400 mm
光源	ブルー LED	ブルーレーザー 7 本
被写体深度	200-700 mm	350-610 mm
対象物との距離	470 mm	470 mm
テクスチャスキャン	あり	なし
安全性	クラス対象外	クラス 1
データ出力形式	STL,OBJ,PLY,ASC,3MF,P3	
本体サイズ	108×110×237 mm	
本体重量	710 g	

文献 [4] から引用

ここで、被写体長 3D 精度というのはスキャン時に取得するポイントの最大累積誤差を示したもので、測定する点数が増える場合や取得点の距離が大きい場合に誤差が累積し、大きくなっていく。

2.2 Blender

Blender とは、オープンソースの完全無料統合型 3DCG・2D・映像編集ソフトウェアである。本研究では“EinScan HX”によって出力した.obj 形式の 3D モデルを編集する目的で使用した。次項からは本研究で用いた Blender の機能について解説する。

2.2.1 スムージング

スムージング機能とは、文献 [7] によると 3D オブジェクト等の点群データにおいて、ノイズを削減するために近傍データを用いて平均化処理を行い、データ点列をスムーズになるようにすることである。Blender では、この処理をペイントツールのようにオブジェクトをなぞるだけでその軌跡に従うように平均化処理が行われている。スムージング処理のイメージ画像を図 2.2 に示す。

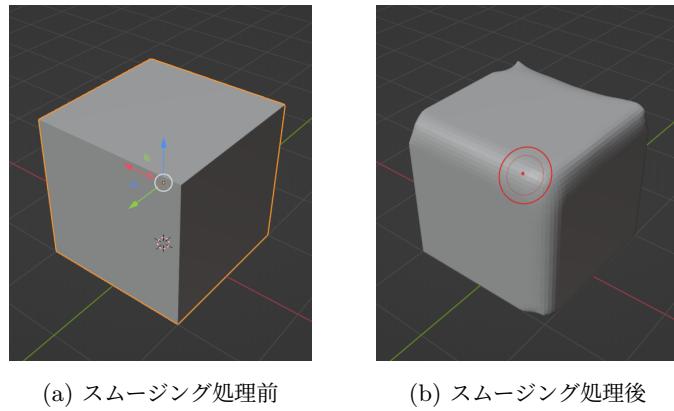


図 2.2: 頂点数 35000 点の立方体オブジェクトのスムージング処理比較

2.2.2 ボーン構成

ボーンは、3D オブジェクトを変形させる際に頂点の移動を制御する支柱の役割を果たす機能である。Unity にインポートした後に物体保持のアニメーションを作成する際、VH の変形を制御するために用いている。ボーンによるメッシュ変形の様子を図 2.3 に示す。

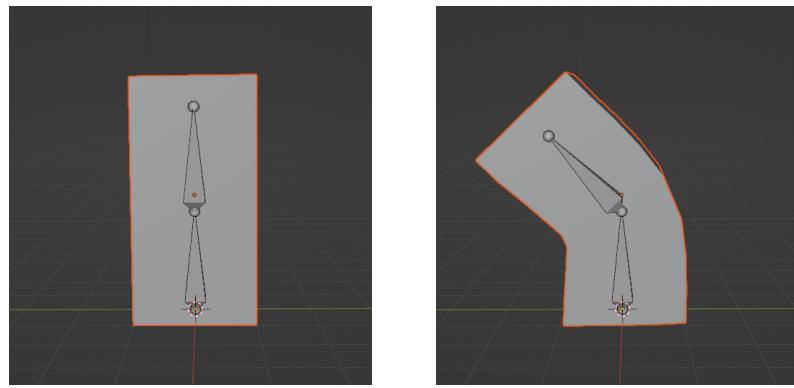


図 2.3: ボーンによるメッシュ変形制御

2.3 Unity

Unity は、Unity Technologies 社が提供するゲーム制作を中心とした統合開発環境のこと、主にスマートフォン向けゲームの制作に用いられている。特徴としては、他社製の開発環境よりも比較的簡単にゲームの制作をすることが可能で、プログラムを必須としない点に強みをもつ。本研究では VR トレーニングシステムを構築する上で利用した機能について解説を加える。

2.3.1 シェーダー

本研究では、Unity にインポートした VH の見た目をよりリアルにするため、標準搭載のシェーダーではなく、“Reflex Shader 2.2”というシェーダーを用いて表示した。このシェーダーは主に VRChat の 3D モデル表示に用いられ、標準のシェーダーと比較して鏡面光が抑制され、モデルのコントラストが強調されていることがわかる。

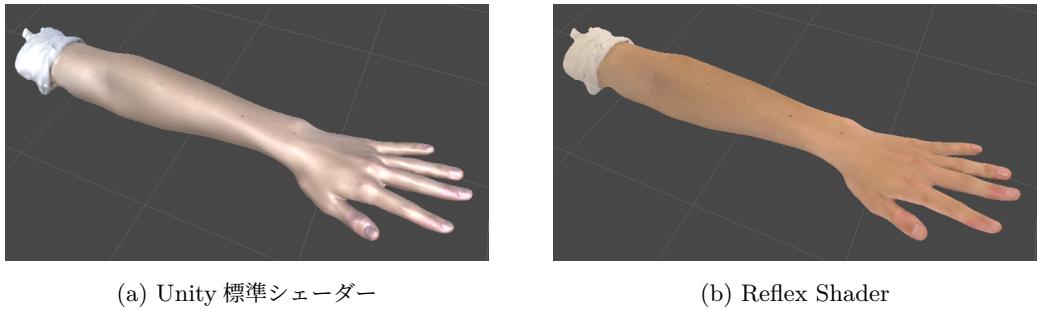


図 2.4: シェーダーによる 3D モデルの違い

2.3.2 衝突判定

本研究では衝突判定に Unity に標準搭載されている Collider コンポーネントと Raycast コンポーネントを用いた衝突判定を行った。次項にそれぞれの衝突判定の違いについて解説する。

Collider

Collider コンポーネントにはオブジェクトの被衝突領域を設定する役割があり、異なる 2 つのオブジェクトに附加された Collider の領域が重なることで衝突フラグを立てる。本研究では PC 版シミュレータにこの方式の衝突判定を用いている。

Raycast

Raycast コンポーネントはオブジェクトの中心から任意方向・任意長さに直線を引き、その直線と Collider コンポーネントを持つオブジェクトの被衝突領域が重なると衝突フラグを立てる。本研究では iOS 版シミュレータにこの方式の衝突判定を用いている。

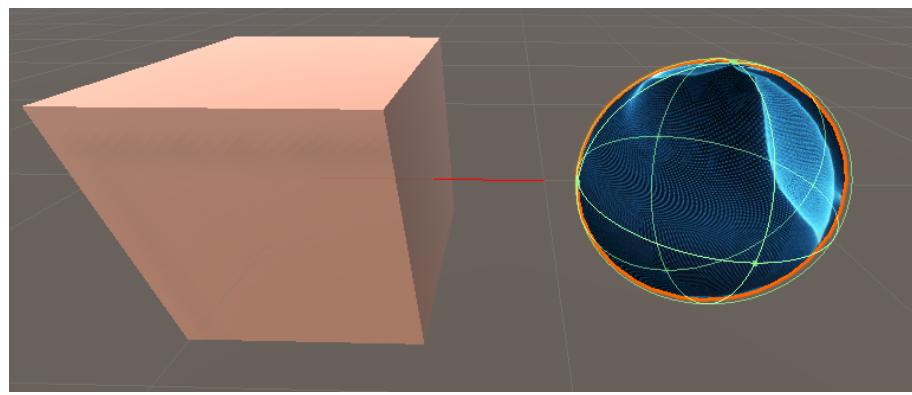


図 2.5: Raycast のイメージ図

図 2.5 のように左側の立方体オブジェクトの中心から射出されている赤い直線が Ray である。この直線と右側の球体オブジェクトを囲う黄緑の領域が Collider コンポーネントの被衝突領域となっている。なお、図 2.5 の Ray や Collider は表示されないようになっている。

2.3.3 アニメーション

オブジェクト保持状態を表現するために VH にオブジェクトごとの保持アニメーションを作製した。Unity におけるアニメーションは各オブジェクトの動きを記録し、再生フラグが立った場合にアニメーションが再生されるようになっている。また、アニメーションを作製する際、初期状態とアニメーションした後のオブジェクトの位置を記録することで自動的にその移動が補完されるようになっている。図 2.6 にアニメーションクリップ機能の画面を示す。

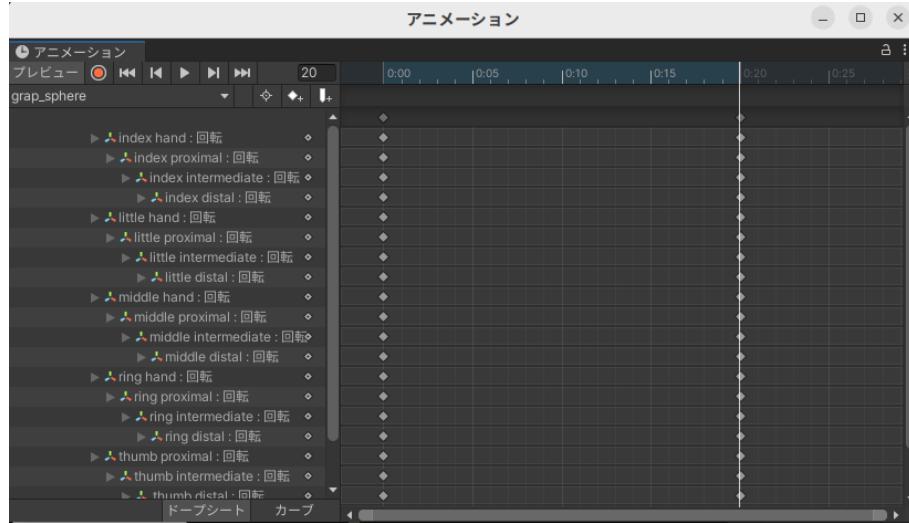


図 2.6: アニメーションクリップ機能

複数のアニメーションの状態遷移を制御する役割としてアニメーター機能が Unity に備わっている。図 2.7 の動作ではシーンの実行直後に緑色ブロックの Entry からオレンジブロックの GrapSphere に遷移し、アニメーションが実行され、赤色ブロックの End に遷移する。アニメーター機能では、遷移条件を付加することも可能である。

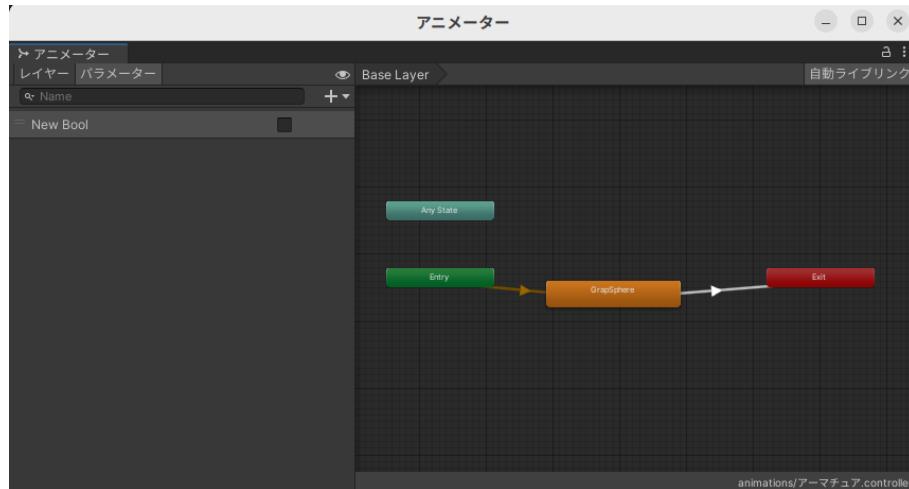


図 2.7: アニメーター機能

2.3.4 ビルド

本項では Unity で構成したシミュレータのビルド方法について解説する。以下にビルドの手順を示す。

1. Unity エディタ上のツールバーから”File”を選択
2. “Build Settings”の項目を選択すると、別ウィンドウに Build Settings が開かれる
3. 追加したいシーンファイルを Unity エディタ上で開いておき、Build Settings のウィンドウから”Add

Open Scenes”を選択

4. Build Settings ウィンドウの左にある Platform から実行したい OS を選択し、ウィンドウ右下の”Switch Platform”を選択
5. Build Settings ウィンドウ右下の”build”を選択してビルド開始

以上の手順で Unity で作成したシーンのビルドが完了する。Windows,LinuxなどのOSではビルドしたフォルダに実行ファイルが作成されているため、そのファイルからシミュレータを実行することができる。また,AndroidOS に関しても、作成した実行ファイルを AndroidOS に転送し、ファイル解凍することで実行できる。

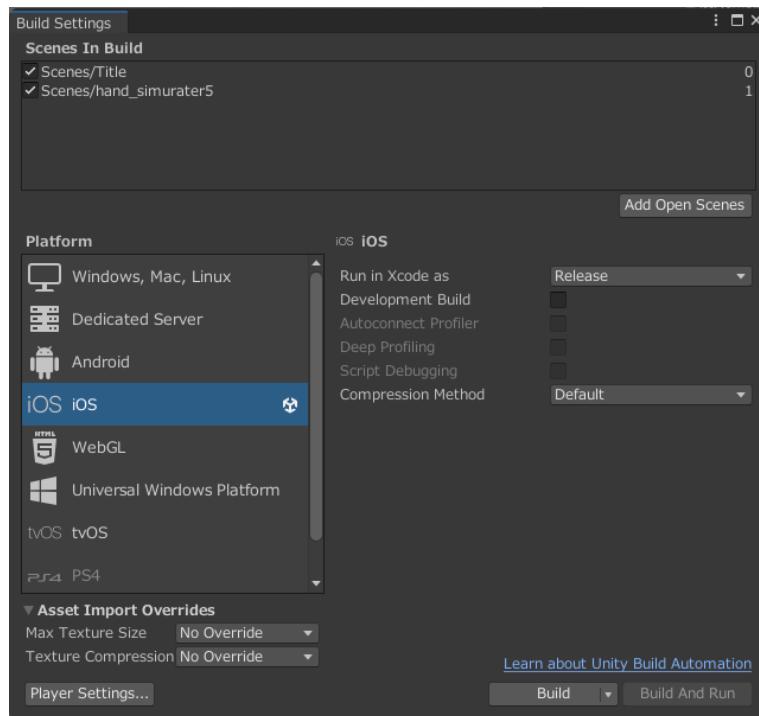


図 2.8: BuildSettings 画面

iOS に関してはビルド・実行方法が異なるため、以下に解説する。

iOS のビルド方法

本項では,iOS 用アプリの実行手順を以下に示す。

1. ビルドされたフォルダを MacOS に転送
2. MacOS でビルドされたフォルダを開き、フォルダ内の “Unity-iPhone.xcodeproj” を Xcode で開く
3. iOS 端末を接続し、端末の設定から端末をデバッグモードにする
4. Xcode ウィンドウ上部の Any iOS Device を選択し、接続した端末を選択
5. Signing & Capabilities を選択し、Team の欄に Apple ID を入力
6. 本シミュレータでは Bluetooth を用いるため、info を選択し、Key の一覧に “Privacy - Bluetooth Peripheral Usage Description” を追加し、Value を “Uses BLE to communicate with devices.” にする
7. iPhone にアプリがインストールされたら、端末の『設定』から『プライバシーとセキュリティ』を開き『Bluetooth』を選択
8. インストールしたアプリに Bluetooth の権限を許可する
9. アプリ一覧からインストールしたアプリを実行する

以上の手順で iOS 用のシミュレータをビルドして実行できる。この手順で一番重要なのが、6 の手順でこれがなければアプリを実行しても Made by Unity のポップアップがでた直後に動かなくなってしまう。これらのエラーは

Xcode 上でログが残されているため、発生しているエラーを解消すれば実行することができる。大抵の場合、端末側の権限付与ができておらずアプリが実行できないエラーが多発するためよく確認をすること。

2.4 FirstVR

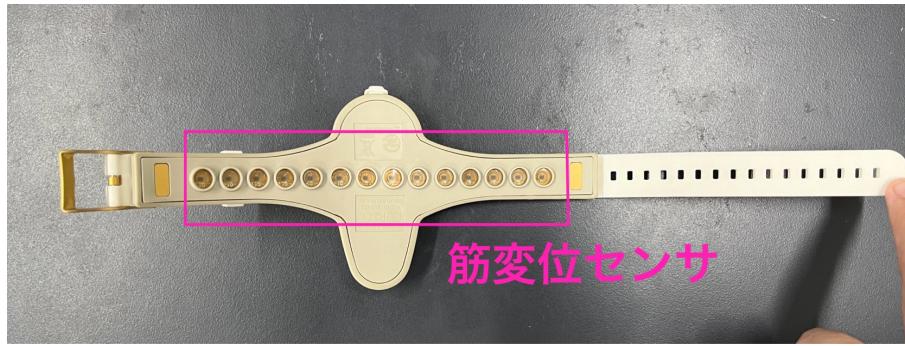
FirstVR は H2L 株式会社 [5] によって開発・製造されている筋変位センサを用いたウェアラブルデバイスである。この製品は現状 iOS と Android のみ対応しており、アプリケーション開発をする際は H2L から公開されている“FirstVR sdk”という Unity 開発用 API が公開されている

2.4.1 デバイス構成

FirstVR の外観を図 2.9 に示す。



(a) 表面画像



(b) 裏面画像

図 2.9: FirstVR

図 2.9(b) のように裏面には筋変位センサが 14 チャンネル配置されており、ここから装着した筋肉の変位を測定している。筋変位センサは近赤外線を射出し肌表面に反射させることで距離を測定し、その値を筋変位センサの測定値として出力している。

2.4.2 トラッキング

FirstVR には 3 軸ジャイロセンサと加速度センサが搭載されており、Unity で作製したアプリ上で測定値を確認することができる。本研究ではこのジャイロセンサを用いて、腕のピッチ、ヨー、ロールの 3 軸の回転をシミュレータに反映させている。

2.4.3 ジェスチャ認識/キャリブレーション

FirstVR のジェスチャ認識方法は、ジェスチャ状態と非ジェスチャ状態における筋変位センサの各チャンネルごとの値をサンプリングし、機械学習させることでジェスチャ状態のセンサ値をしきい値としてジェスチャ認識を行っている。

第 3 章 研究手順

3.1 使用器具

本研究での使用器具を表 3.1 に示す.

表 3.1: 本研究における使用器具

No	機器名	型番	シリアル No	備考
1	EinScan HX			
2	FirstVR	UHL-01		
3	スマートフォン 1	iphone SE2		iOS16.7.2
4	スマートフォン 2	iphone 13		iOS17.2.1
5	スマートフォン 3	HUAWEI Nova Lite2		AndroidOS 9
6	PC-1			Ubuntu22.04
7	PC-2	Mac mini2		MacOS

本研究で使用したソフトウェアを表 3.2 に示す.

表 3.2: 使用ソフトウェア

No	ソフトウェア名	バージョン	使用 OS	備考
1	Blender	3.0.1	Ubuntu	
2	Unity	2022.3.11f1	Ubuntu	
3	Xcode	15.0.1	MacOS	

3.2 VH の 3D モデル取り込み

ハンディ 3D スキャナである EinScan HX を用いて左腕をスキャンする. 本研究では, VH にテクスチャを貼って用いることを前提としているため, スキャン形式を Rapid スキャンモードで, 3D モデルの精度を高めるために頂点数を 50 万点で出力し, 出力形式としてテクスチャがメッシュに割当されている obj 形式を選択する.

3.3 VH のスムージング処理・ボーン配置

obj 形式で取り込んだ 3D モデルは測定によるノイズが含まれており, 特に掌と手の甲の境界線上にが段差のように途切れてしまう. このノイズを除去・補完するために, オブジェクト表面の凹凸を平坦にする効果があるスムージング処理を行う. また, アニメーション作製のために処理後の 3D モデルにボーンを配置する.

3.4 オブジェクト保持表現

オブジェクト保持表現として Unity のアニメーション機能を用いて保持可能な球体・円柱・立方体の 3 つのオブジェクトごとのアニメーションを作製し, アニメーションクリップ機能を用いてジェスチャをしていない状態と各アニメーションを衝突判定と入力インターフェースによって遷移させる.

3.5 FirstVR の性能評価手法

本研究で構成するシミュレータの入力感度を検証するため、実験協力者として電子工学科の 31 名を対象に以下の手順で FirstVR の評価を行う。また、ジェスチャ状態は物体保持のアニメーションと同期させるため、じゃんけんのゲームのジェスチャを学習させる。FirstVR 性能評価の配置図を図 3.1 に示す。

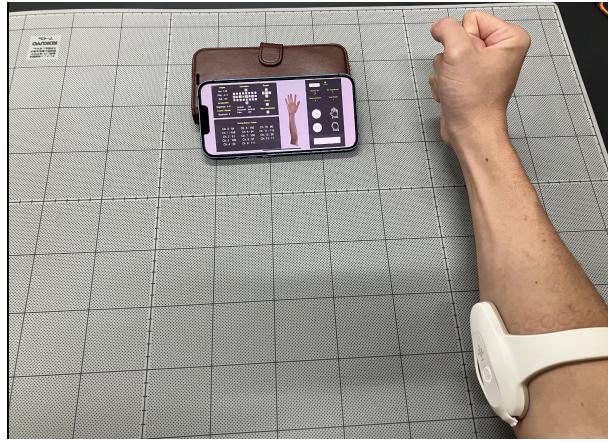


図 3.1: FirstVR の性能評価の参考図

1. 装着位置の決定

FirstVR の動作原理より、近赤外線で筋変位を取得しているため、かなり密着させて装着する必要がある。そのため本研究では、筋肉量が一番多い位置を肘から前腕の 1/4 程度の距離と推定し、実験協力者の肘から手首までの長さを測定し、肘を原点に 1/4 の距離（約 7cm）で装着する。

2. FirstVR の接続と学習

端末と FirstVR を接続し、測定する sample 数でジェスチャをしていない状態とジェスチャしている状態を学習させた。

3. FirstVR の測定

ジェスチャをしていない状態の筋変位センサの値を 1 回測定し、ゲームのジェスチャにおけるセンサ値の変化量を 5 回測定した。

4. FirstVR のジェスチャ保持特性の検証

ノイズの少なかった sample 数を選び出し、ジェスチャ状態で手の角度を上下左右に動かした場合のジェスチャ状態の変化を動画に撮影して検証する。

ノイズの判定方法は実際の腕の動作とアプリケーションの表示が異なる場合にノイズと判定した。sample 数の選出はノイズの回数が同数だった場合はシミュレータの負荷軽減のために sample 数が少ない方を選出する。

FirstVR の評価基準としてジェスチャ認識率と独自の評価指標による性能評価を行なった。各実験協力者におけるジェスチャ認識率 G_r は各 sample 数 $D = 11$ 、ジェスチャ誤判定の回数 m として、測定回数 $s = 5$ のとき式 3.1 によって求められる。

$$G_r = \left(1 - \frac{m}{D \times s} \right) \times 100 [\%] \quad (3.1)$$

また、各 sample 数におけるジェスチャ認識率 G_s は実験協力者 $n = 31$ として同様に式 3.2 で求められる。

$$G_s = \left(1 - \frac{m}{n \times s} \right) \times 100 [\%] \quad (3.2)$$

これらを用いて、ジェスチャの認識率を比較し、最適な sample 数の検討を行う。

加えて、FirstVR で筋変位を測定した 14 チャンネル光変位センサの測定値を用いてジェスチャ認識の精度を確認するため、各実験協力者・各 sample 数ごとの評価指標として総変化量 X を定める。総変化量の算出は測定回数 $s = 5$ 、チャンネル数 $r = 14$ としてジェスチャ状態で測定した筋変位センサの値を M_{sr} とジェスチャしていない状態の筋変位センサの値 N_r とすると式 3.3 と示すことができる。この評価指標を用いて各 sample

数ごとに分散を調べ、最適な sample 数の検討を行う。

$$X = \frac{1}{5} \sum_{s=1}^5 \sum_{r=0}^{13} |N_r - M_{sr}| \quad (3.3)$$

3.6 シミュレータの構成

Blender で処理した VH を Unity にインポートし、保持可能オブジェクトを球体、円柱、立方体の 3 つ用意し、大別して PLAYER, STAGE, UI の構成にする。シミュレータは入力インターフェースとしてキーボード・マウスを用いる PC 版と、FirstVR を用いる iOS 版の 2 種類を構成した。図 3.2 にシミュレータの構成図を示す。

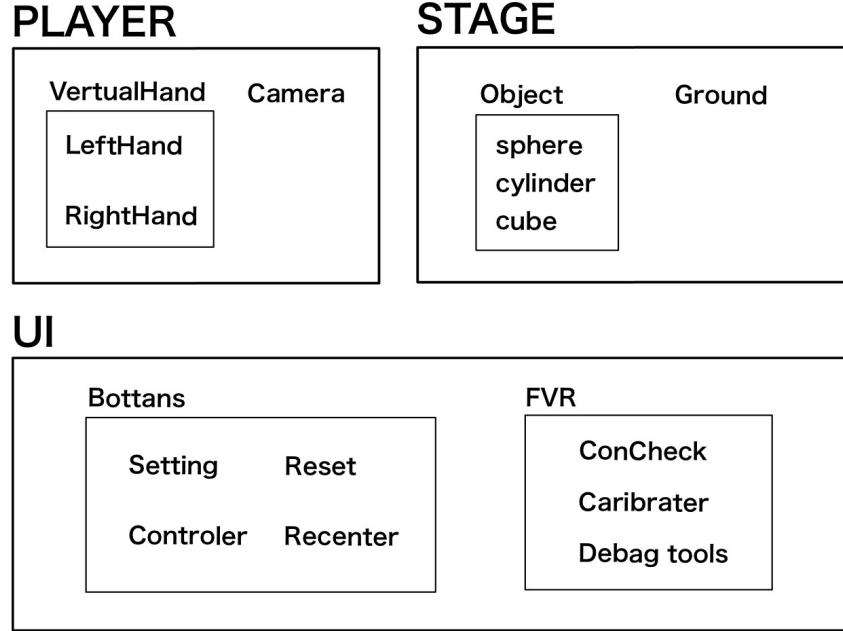


図 3.2: シミュレータ構成図

PLAYER には VH として左手用に LeftHand、右手用に RightHand のオブジェクトを配置し、画面表示用に Camera オブジェクトを追加した。STAGE には保持可能オブジェクト (Object) として球体 (sphere)、円柱 (cylinder)、立方体 (cube) を配置し、オブジェクトが落下し続けないように Ground を配置している。UI には主に設定画面やオブジェクト位置のリセット、PLAYER の操作などのシミュレータを快適に扱えるようにする Bottans と、FirstVR の接続やキャリブレーション設定を行う FVR が配置されている。それぞれのオブジェクトにはシミュレータ空間上で物理演算を付与しているため、オブジェクトの落下や回転はリアルに行われるようになっている。ここで、PC 版には FirstVR の接続が必要ないため、UI の FVR 部分は実装していない。各機能については 4.4 節にて後述する。

3.7 シミュレータの定性評価手法

電子工学科学生 11 名に協力していただき、PC 版シミュレータと iOS 版シミュレータの 2 種類の操作説明を説明した後、5 分程度体験してもらい、各シミュレータにおいて以下の 3 項目について 6 段階リッカート尺度を用いた定性評価アンケート調査を行う。

1. シミュレータの没入感
2. インターフェースの操作性
3. シミュレータの応答性

評価点数が高いほどそれぞれの項目において高得点の評価となるように設定し、調査結果に対して分析を行う。

第 4 章 研究結果

4.1 VH (Virtual Hand) の 3D モデル構成

3D スキャナで取り込んだ左腕 3D モデルと実際の左腕の比較を図 4.1 に示す.

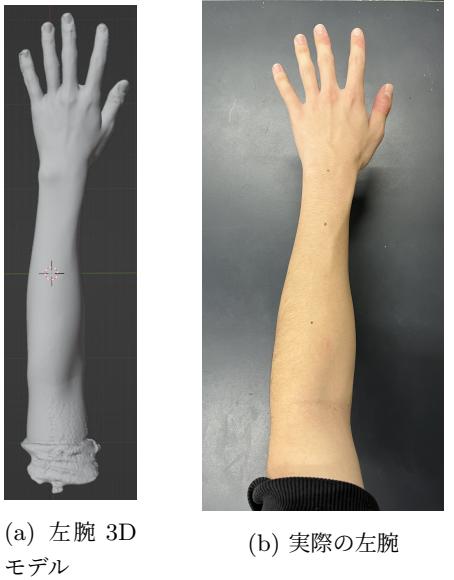


図 4.1: 取り込んだ左腕 3D モデルと実際の左腕の比較

実際の左腕と比較して, 手の甲の表現や筋の出方の表現がとても精巧に再現されているように感じられる. しかし, 少し腕周りの太さが細くなっているようにも見られる. また, 図 4.2(a) 取り込んだ直後の 3D モデルでは撮影時のノイズが含まれているため, Blender のスムージング機能を用いて図 4.2(b) のように 3D モデルを平滑化した.

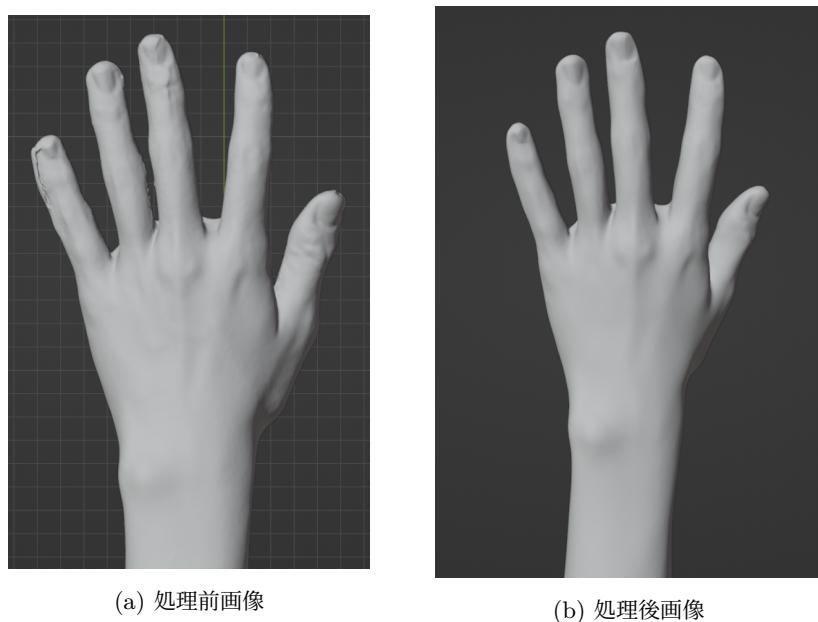


図 4.2: スムージング処理前後の 3D モデル比較

図 4.2(a) では小指や人差し指の先端部分に測定ノイズによる 3D モデルの凹凸が見られるが, 図 4.2(b) ではスムージング処理により平滑化されていることがわかる.

また,3D モデルを Unity にインポートする際, オブジェクト保持のアニメーションを作製するためにメッシュの制御を行う必要があるため, 図 4.3 のようにボーンを配置した.

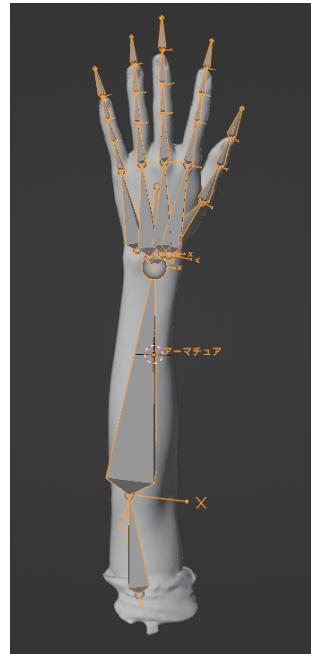


図 4.3: ボーン配置図

図 4.3 より, ボーンは関節を目安に配置しており, 各指の先端にはエンドボーンと呼ばれるアニメーション制御用のボーンを追加した. このエンドボーンを移動させることで各指のボーン変形を自動的に補完するようになっている.

4.2 オブジェクト保持

球体, 円柱, 立方体のオブジェクト 3 種類の保持アニメーションを図 4.4~4.6 に示す.

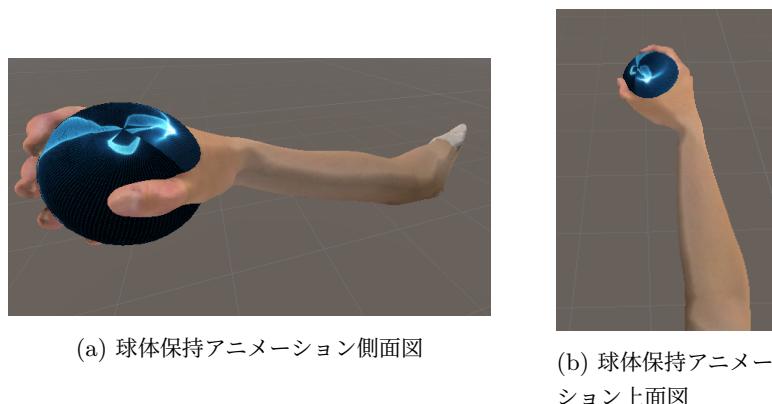
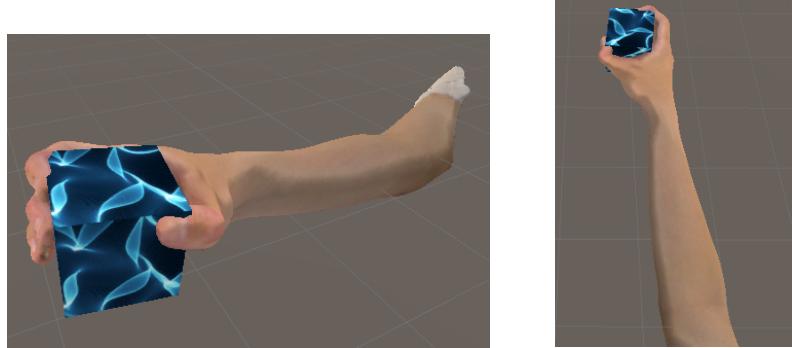


図 4.4: 球体オブジェクトの保持表現



(a) 円柱保持アニメーション側面図
(b) 円柱保持アニメーション上面図

図 4.5: 円柱オブジェクトの保持表現



(a) 立方体保持アニメーション側面図
(b) 立方体保持アニメーション上面図

図 4.6: 立方体オブジェクトの保持表現

オブジェクトが静止状態の場合でも動いている場合でも衝突条件を見たせばオブジェクトを保持することができる。実際にボールや水筒などの形の似たものを掴んでいる写真を参考にアニメーションを作製したため、かなり現実に近い掴み方をしているように見える。しかし、保持アニメーションを行う際、オブジェクトの位置は VH の位置に紐付けられるのだが、オブジェクトの回転までは制御していないので円柱や立方体は保持する角度によって指や掌を貫通するような掴み方になる。次に、アニメーター構成を図 4.7 に示す。

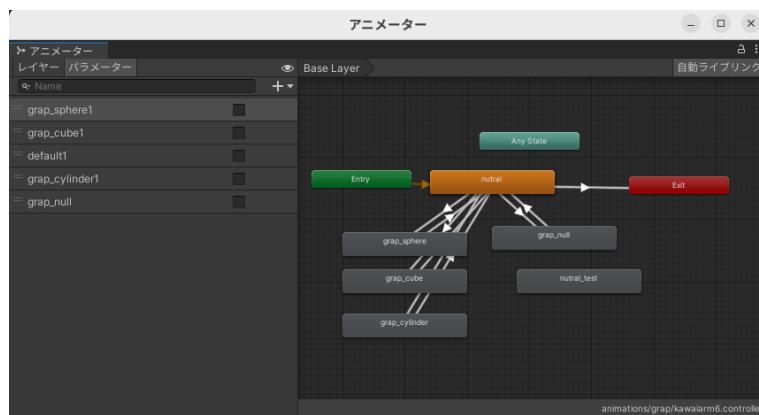


図 4.7: アニメーター構成

各アニメーションの遷移フラグとして `grap_sphere1`, `grap_cylinder1`, `grap_cube1` を用意した。ジェスチャをしていない状態で `nutral` という名にもアニメーションをしないクリップをループ再生し続け、衝突判定とジェスチャ認識が同時に行われた場合に対応するオブジェクトのフラグを立てて、オブジェクト保持のアニメーションを再生させる。

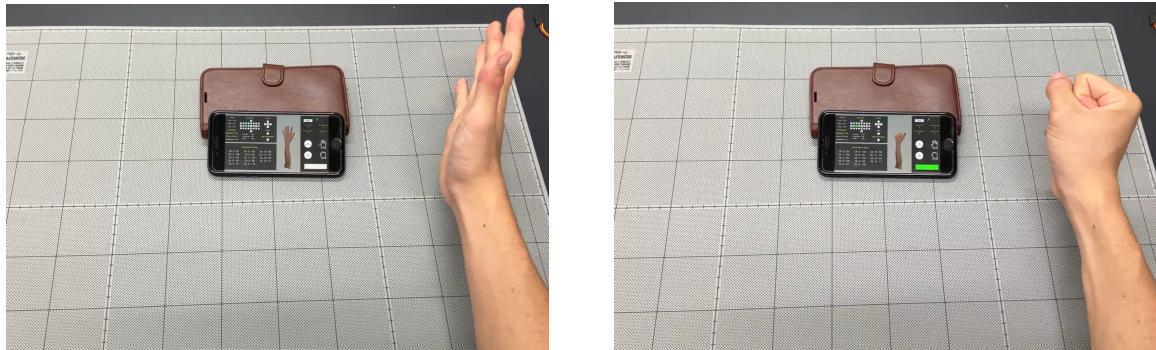
4.3 FirstVR の性能評価

FirstVR を装着する様子を図 4.8 に示す。



図 4.8: FirstVR 装着方法

机上に 5cm 四方のグリッド線が描かれたボードを用いて肘の設置店から手首までを測定し、その 1/4 の距離に FirstVR を装着させた。図 4.9 に測定時の様子を示す。



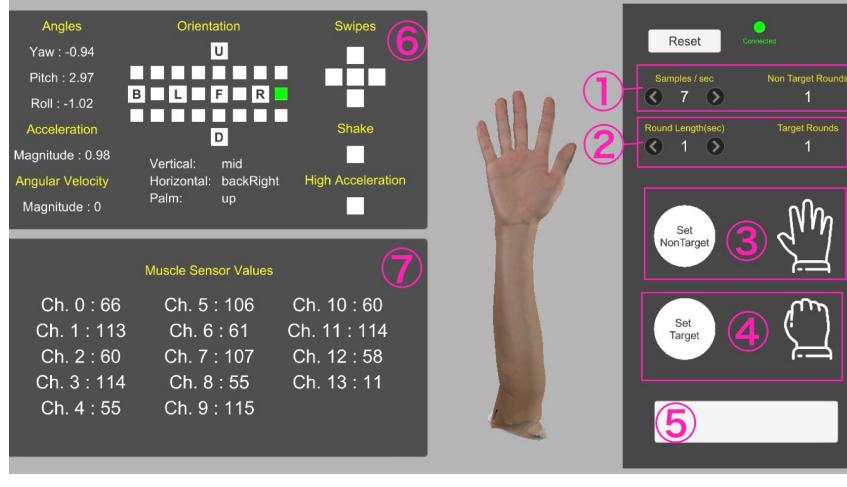
(a) ジェスチャしていない状態

(b) ジェスチャ状態

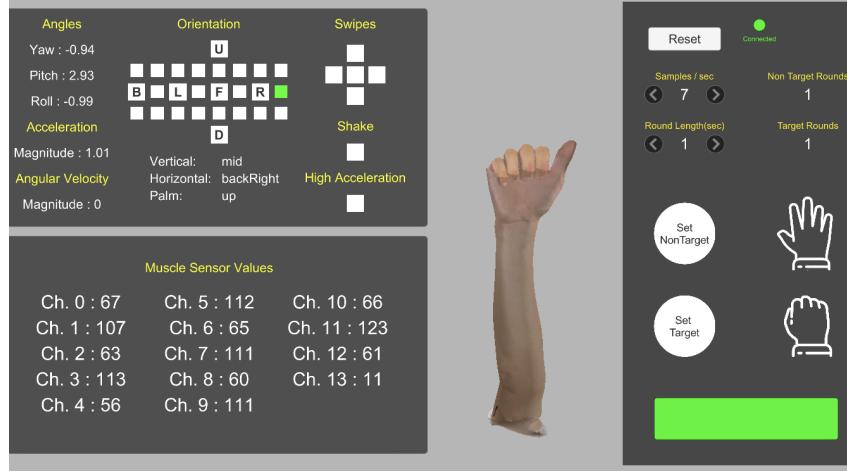
図 4.9: FirstVR 性能評価の様子

図 4.9 のようにジェスチャ状態を 1 回、ジェスチャしていない状態を 5 回測定し、それを各 sample 数で繰り返した。全員分のデータは記載することができないため付録にある QR コードから FVRDataFiles を開いてもらうと各実験協力者のデータが閲覧できるようになっている。データの傾向としては、ジェスチャ状態の登録の際、強く握りすぎない方がジェスチャ認識のノイズが低減した。

FirstVR の性能評価用アプリケーションの構成を図 4.10 に示す.



(a) ジェスチャ未判定時



(b) ジェスチャ判定時

図 4.10: FirstVR 性能評価用アプリケーションの構成

図 4.10a の①で学習させる sample 数を設定でき、②では学習時間を設定することができる。また、ジェスチャしていない状態を学習させるときは③のボタン、ジェスチャ状態を学習させるときは④のボタンで学習させることができる。学習が完了すると、①と②の右側のカウントが増えていく。FirstVR の性能評価の際は簡易のために学習時間を 1 に固定して sample 数のみを変化させた。ジェスチャ判定時には⑤のバーが白色から緑色に変化するため、ジェスチャをした瞬間の筋変位センサの値を測定し、そのタイミングでバーの色が変化していなければジェスチャ認識ができていないものとした。⑥は性能評価に用いなかったが、トラッキングのデータを取ることができ、手のヨー、ピッチ、ロールの値やスワイプ方向の取得が可能である。⑦は FirstVR 裏面にある筋変位センサの各チャンネル毎の測定値を見ることができる。この値を用いて、最適 sample 数の選定を行った。

各実験協力者におけるジェスチャ認識率を表 4.1 に示す.

表 4.1: 各実験協力者におけるジェスチャ認識率

Person	誤検知回数 [回]	ジェスチャ認識率 [%]
1	4	92.73
2	0	100.00
3	1	98.18
4	3	94.55
5	0	100.00
6	3	94.55
7	0	100.00
8	0	100.00
9	4	92.73
10	17	69.09
11	0	100.00
12	0	100.00
13	0	100.00
14	16	70.91
15	0	100.00
16	0	100.00
17	2	96.36
18	3	94.55
19	27	50.91
20	1	98.18
21	3	94.55
22	1	98.18
23	0	100.00
24	5	90.91
25	7	87.27
26	1	98.18
27	0	100.00
28	2	96.36
29	1	98.18
30	14	74.55
31	4	92.73

表 4.1 より, ジェスチャ誤検知の回数が極端に多い場合とほとんど誤検知がないというように分布していることがわかる. また, 10 回以上誤検知が起きている実験協力者のデータでは特定の sample 数によらずに誤検知が発生しているため, sample 数によるジェスチャ認識率のデータ含めてしまうとノイズによってデータが正しく求められないため除外した.

各 sample 数ごとのジェスチャ認識率を表 4.2 に示す.

表 4.2: sample 数ごとのジェスチャ認識率

sample 数	誤検知回数 [回]	ジェスチャ認識率 [%]
7	9	93.33
10	2	98.52
20	4	97.04
30	3	97.78
40	5	96.30
50	6	95.56
60	6	95.56
70	2	98.52
80	0	100.00
90	5	96.30
100	3	97.78
合計	45	96.97

表 4.2 より, 最低値 93.33 %, 最高値 100 %のジェスチャ認識率だった. したがって, 96.66 ± 3.34 %の範囲で変動していることがわかる. しかし, どの sample 数においてもジェスチャの認識率が 90 %を上回っているため, このデータからどの sample 数が最適か判定できなかった.

FirstVR の sample 数における総変化量を表 4.3 に示す.

表 4.3: 各実験協力者における sample 数ごとの総変化量一覧

Person	sample 数										
	7	10	20	30	40	50	60	70	80	90	100
1	44.4	25.6	21.8	19.4	12.2	12.6	13.4	22.4	8.2	11.8	27.2
2	37.0	23.4	30.2	17.2	27.2	31.8	45.4	26.2	29.4	19.2	18.2
3	23.4	20.2	28.4	20.6	21.8	30.4	17.2	23.6	39.0	17.4	22.0
4	49.0	69.8	108.8	100.6	78.0	62.2	61.4	40.8	60.6	36.6	37.6
5	38.2	53.2	51.2	34.6	35.4	43.6	46.6	48.2	47.4	32.0	37.6
6	31.8	15.6	17.6	38.4	23.2	23.4	32.0	37.8	14.8	22.4	22.0
7	69.2	44.8	57.0	25.8	33.2	39.6	23.0	33.6	53.8	29.8	40.0
8	67.0	66.0	55.4	65.0	67.0	78.6	93.6	59.0	66.8	59.4	65.4
9	44.4	25.6	21.8	19.4	12.2	12.6	13.4	22.4	8.2	11.8	27.2
10	20.8	13.8	15.4	16.4	13.2	17.8	43.8	17.6	12.0	83.6	14.4
11	24.8	22.8	26.2	21.0	27.8	48.8	41.2	15.6	23.2	14.2	19.4
12	34.2	29.8	19.8	25.8	24.0	18.2	32.4	25.4	35.4	35.6	25.2
13	20.6	17.4	21.6	29.0	19.0	22.8	27.6	32.2	56.8	44.0	31.2
14	19.8	22.0	11.6	10.0	16.8	17.4	12.4	17.6	21.6	15.8	11.4
15	50.8	58.4	51.2	56.6	67.4	37.6	45.8	52.4	51.6	59.6	56.2
16	35.2	50.2	62.6	36.4	27.2	39.6	38.8	43.4	45.4	52.4	48.8
17	49.4	42.2	41.2	36.4	49.4	34.4	33.2	34.4	22.2	13.8	36.0
18	23.8	26.8	36.2	38.0	28.0	35.2	34.0	24.4	29.2	39.6	25.8
19	33.6	11.8	10.4	16.8	17.8	11.0	19.4	13.6	24.4	13.2	9.6
20	32.6	36.4	33.0	17.6	22.6	26.8	32.2	25.2	20.8	30.4	13.8
21	32.8	42.6	36.6	30.6	26.4	41.4	32.6	20.2	25.0	15.8	17.6
22	42.2	27.6	37.8	33.8	39.2	50.0	34.6	38.0	38.8	26.8	24.2
23	51.8	40.4	44.0	44.6	41.4	39.2	42.8	38.4	41.6	43.0	30.0
24	38.0	30.6	38.2	35.6	23.8	20.4	24.4	24.2	22.6	30.6	16.8
25	25.2	33.0	32.0	23.8	28.0	20.0	31.4	13.6	19.6	19.4	21.2
26	39.8	32.8	30.2	38.2	34.8	22.6	30.4	24.0	24.0	12.6	20.6
27	68.8	78.0	50.4	51.6	44.2	50.8	40.0	48.2	46.4	48.8	62.0
28	20.4	29.8	40.6	45.0	23.8	25.2	27.4	39.0	25.2	28.8	39.6
29	32.6	29.8	21.6	19.0	22.4	17.6	21.2	28.2	41.4	35.6	20.8
30	23.2	8.0	13.6	23.2	21.2	25.4	24.0	12.6	16.0	24.4	17.8
31	44.4	25.6	21.8	19.4	12.2	12.6	13.4	22.4	8.2	11.8	27.2
標本分散	189.9	287.3	375.3	315.9	259.9	241.7	252.4	143.3	256.2	290.8	196.0

総変化量が 10 前後の数値では FirstVR がジェスチャに対してほとんど応答していなかった. この結果を用いて, 同じ sample 数における実験協力者ごとの数値の分散が少ない sample 数が最適とした.

表 4.3 の各 sample 数における分散を図 4.11 に示す.

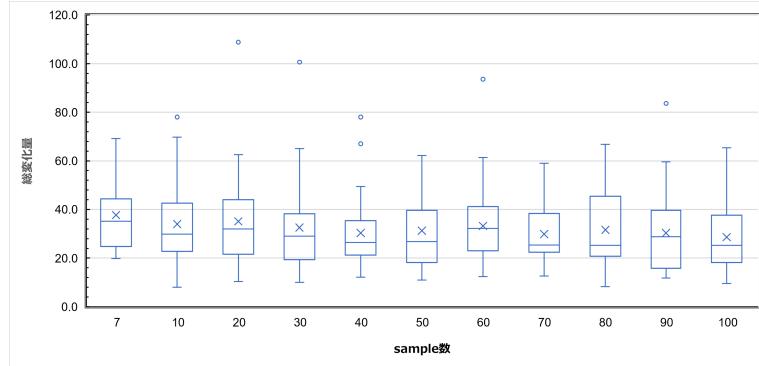


図 4.11: sample 数ごとの総変化量分散

図 4.11 より, 標本分散は sample70 が最も小さく, 次いで sample7 が分散が少なくなっていることがわかる. ここで, sample 数 7,70,100 以外のデータは分散がこの 3 種類よりも比較的大きく, 外れ値も含んでいるため, 安定して動作していると考えにくい. また, この 3 種類の中で最もシミュレータに対する負荷が小さいデータとして sample7 を選定した.

4.4 シミュレータの構成

PC 版シミュレータの画面構成を図 4.12 に示す.

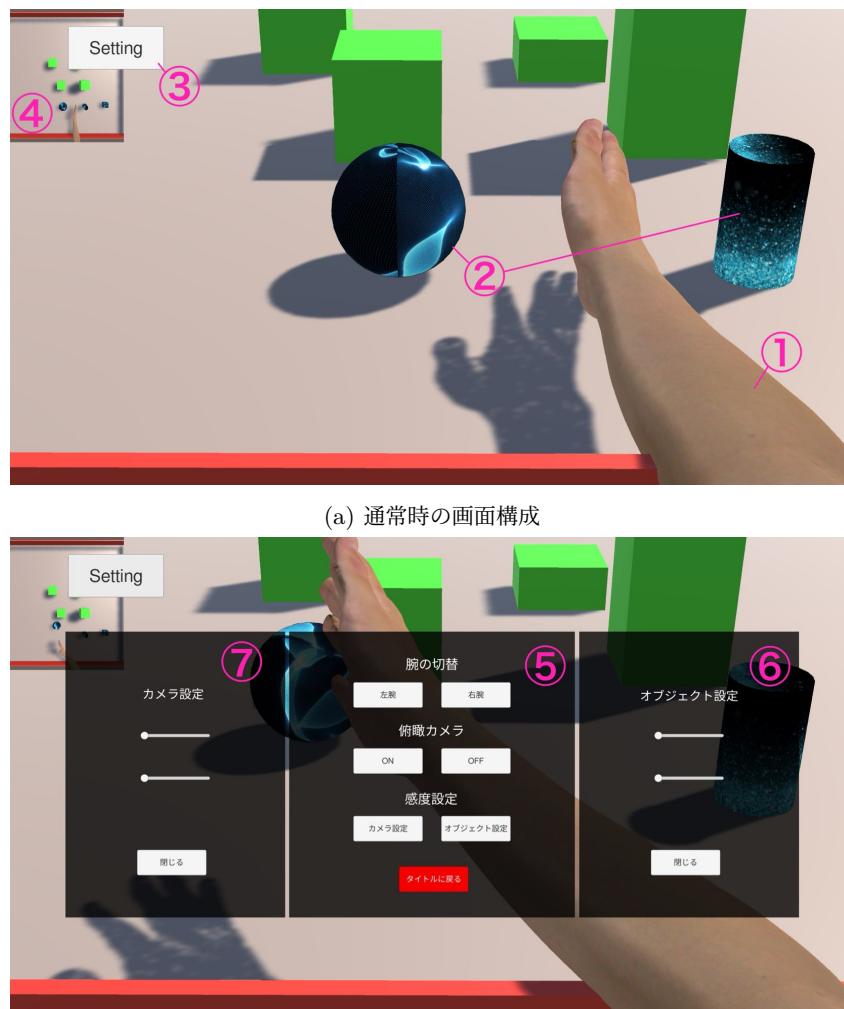


図 4.12: PC 版シミュレータの実行画面

1. VirtualHand

Blender から取り込んだ 3D モデルで、掌部分に Collider を配置して衝突判定を行う。マウスの移動に追従して腕の位置も変化する。

2. Object

球体 (sphere), 円柱 (cylinder), 立方体 (cube) のオブジェクトを保持可能オブジェクトとして配置した。

3. メニュー起動ボタン

メインメニューを開くためのボタンである。

4. 俯瞰カメラ

シミュレータのオブジェクトを上から見下ろしたカメラである。PLAYER や Object の位置を確認できるように配置した。

5. メインメニュー

シミュレータ起動用のタイトル画面遷移や俯瞰カメラの ON/OFF,

6. オブジェクト設定

7. カメラ設定

iOS 版シミュレータの画面構成を図 4.13 に示す。



図 4.13: iOS 版シミュレータの実行画面

1. VirtualHand
2. Object
3. Controller
4. 俯瞰カメラ
5. FVRConcheck
6. Buttons
7. デバッグ用カメラ
8. 座標モニタ
9. メインメニュー
10. オブジェクト感度設定
11. Calibrator

12. DebugTools

4.5 シミュレータの定性評価

PC 版, iOS 版シミュレータの定性評価の様子を図 4.14 に示す。



図 4.14: シミュレータ定性評価の様子

PC 版ではキーボード・マウスを入力インターフェースとしているため, 左クリック押下時にオブジェクト保持のフラグを立て, 右クリックを押下しながらマウスを移動させることによって視点を動かすことができる。iOS 版では iPhone にシミュレータを表示させ, FirstVR のジェスチャ認識機能によってじゃんけんのゲームの状態を学習することでオブジェクト保持ができる。この学習が終了してから約 5 分間シミュレータを評価した。

次に, アンケート調査の結果を図 4.15~4.17 に示す。

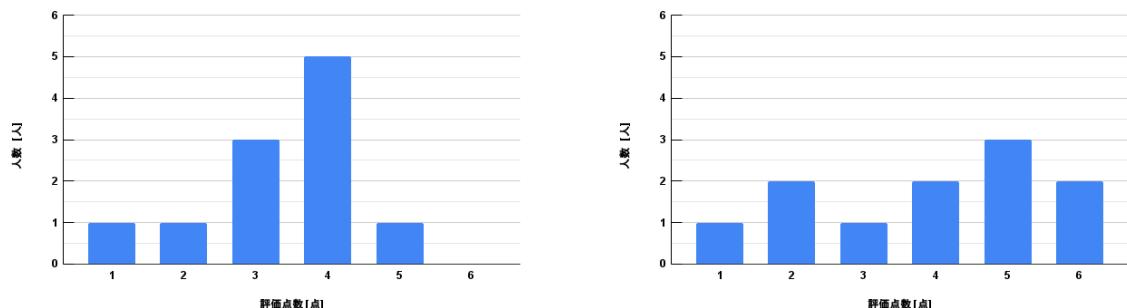


図 4.15: 没入感の評価比較

PC 版の平均値 3.36 点 中央値は 4 点 最高値 5 点 最低値 1 点 標本分散 iOS 版の平均値 3.90 点 中央値は 4 点 最高値 6 点 最低値 1 点 標本分散

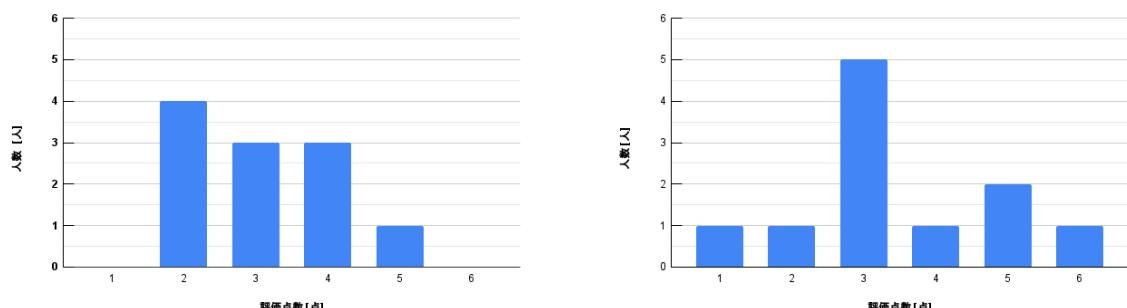


図 4.16: 操作性の評価比較

PC 版の平均値 3.09 点 中央値は点 最高値 5 点 最低値 1 点 標本分散 iOS 版の平均値 3.90 点 中央値は 4 点 最高値 6 点 最低値 1 点 標本分散

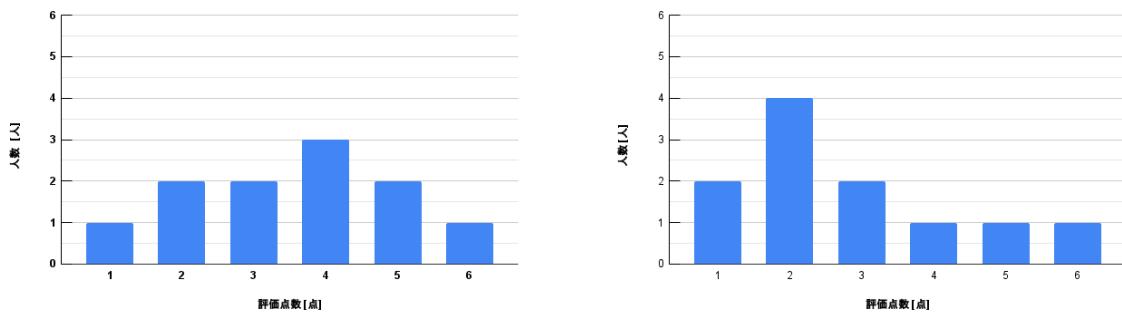


図 4.17: 操作遅延の評価比較

PC 版の平均値 3.09 点 中央値は 3 点 最高値 5 点 最低値 2 点 標本分散 iOS 版の平均値 3.90 点 中央値は 4 点 最高値 6 点 最低値 1 点 標本分散

シミュレータについて感想や改善点をお聞かせください(任意)

4 件の回答

モニタのサイズによって、没入感(違和感)が変わるように感じた。

判定が難しい(特に立方体)
手で掘む位置が少し前過ぎる気がする

pc版では視点移動をキーボードに割り当てるなどして操作性を向上させてほしい

iosの時に腕をひっくり返した時(手のひらが天井)は再現されていなかった

図 4.18: シミュレータ定性評価のコメント

第5章 考察

FirstVR を入力インターフェースとして実際の筋電義手を用いるより安価でジェスチャ認識によってある程度オブジェクトを持ち上げと保持ができるシミュレータを構成することができた。しかし、ゲーのジェスチャのまま腕を動かすとジェスチャ状態から外れてしまうことがある。これは FirstVR が筋肉の変位を測定しているため、腕のひねりや向きによって筋収縮が起こり、ジェスチャのしきい値を下回ってしまうためであると考えられる。また、ジェスチャに似ている動作や部分的に同じ動きをしてもジェスチャ状態であると判定されてしまうこともある。例えば、ゲーのジェスチャを登録した際に、手首を掌屈させるとジェスチャ状態と誤検知されてしまったり、じゃんけんのチョキのジェスチャをしても誤検知されてしまうこともある。このように、FirstVR は筋変位を測定しているため、類似する筋肉の動きで誤検知する場合が多いので、似たようなジェスチャを複数登録することはかなり難しい。だが、ジェスチャ状態が外れる腕の位置で学習回数を重ねることで少しだけジェスチャ状態が外れにくくなることがわかっている。この条件を調べていくことでさらに複雑なジェスチャや指の動きをトラッキングできるような可能性もあると考えている。

次に、FirstVR を上肢切断者に装着してシミュレータを使ってもらうことが本研究の最終目的として捉えているがそもそも FirstVR で筋変位を読み取れるかどうか不明なことも課題点として挙げられる。後天的に切断した場合はそれまでにゲーのジェスチャを行った経験があるため、筋肉の動きを再現することが可能かもしれないが、先天的に切断している場合はまずゲーのジェスチャがどういう筋肉の状態なのかということから訓練する必要がある。文献 [8] によると、上肢切断にはレベルが存在し、今回の実験で装着した位置はおよそ前腕切断に相当する位置で、本シミュレータで FirstVR を用いることができる手は手関節離断、前腕切断の 2 つである。FirstVR を上腕で用いることができるようになれば肘関節切断や上腕切断の状態でもシミュレータが用いることができるようになる。

FirstVR を用いたシミュレータは文献 [3] のカグラのように VR ゴーグルを用いて扱うことができればより訓練効果に影響があるのではないかと考えられる。その際は iOS 版に搭載しているコントローラボタンではなく iPhone と FirstVR の距離や iPhone 自体のセンサを用いて移動や顔の振り向きを検知する必要がある。

本研究では腕の 3D モデルを EinScanHX によって作製したが、今後 3D スキャナが一般的に普及して、体格が近い人の腕をスキャンできるようになると想定すると、そのデータをシミュレータに用いるためにスムージングからボーン配置までを自動的に行えるアプリケーションがあれば Blender を用いて作業する工程を省略することができるようになる。

第6章 おわりに

本研究では PC 版のシミュレータと iOS 版のシミュレータの 2 種類を作製し, 健常者に体験してもらい, その定性評価を行うことで FirstVR を用いたシミュレータの方が没入感が高い傾向があることを示した. しかし, 操作性の点ではキーボード・マウスと明確な差は見受けられず, 遅延に関しては PC 版の方が少ないという課題点も見つかった. また, 表示するモニターの違いで没入感が違ったという意見も挙げられており, 実験方法を再検討する必要がある.

今後の課題としては, FirstVR の性能評価が不十分であったため, シミュレータのオブジェクト保持状態がノイズで保持されていない場合が多く, 操作性の評価が低くなってしまったと考えられる. そのため, FirstVR の条件を以下に挙げる.

- 装着位置
- 使用者の筋肉量
- 学習時間
- 学習回数

これらの観点から条件を分けて実験を行う必要がある. また, FirstVR の内部回路の学習システムについての理解が不足しているためどのような条件が最適なのかということについては公式 HP のリファレンスを詳しく調べていく必要があると考えられる. 加えて, FirstVR がスマートフォンデバイスでしか使用できない点についても BLE 通信のスクリプトファイルを解読することで PC との通信ができる可能性もある. そして, 当初の目標であったオブジェクトの違いによる訓練効果の評価については作成したシミュレータのオブジェクトの機能を引き継いだままモデルだけ置き換えてシミュレータを実装することで定性評価を行うことは可能であると考える. 定量的なデータを取得するためには脳波測定を用いての測定やシミュレータと現実空間の移動量の誤差などを検討していくことになるだろう. 最後に, シミュレータの没入感をあげる方法として, VR ゴーグルを用いたシステムにできるとより訓練効果などに影響を与えるのではないかと思われる.

謝辞

一年間丁寧に指導していただいた戸崎 哲也教授、および同研究室の村岡 永遠君に深く感謝いたします。また、実験に協力していただいた電子工学科の31名にも深く感謝いたします。

参考文献

- [1] 芝軒 太郎 他.“VR を利用した筋電義手操作トレーニングシステムの開発と仮想 Box and Block Test の実現”. JRSJ. 2012 July.
- [2] Osumi M, et.al. “Characteristics of Phantom Limb Pain Alleviated with Virtual Reality Rehabilitation”. Pain Med. 2019 May.
- [3] mediVR.Inc.,<https://www.medivr.jp/> 最終閲覧日 2023/01/31
- [4] 株式会社サンステラ, <https://www.einscan.jp/einscan-hx>
- [5] H2L.Inc.,Tokyo106-0032,Japan;satoshi.hosono@h2l.jp
- [6] Tamon Miyake, et.al.“Gait Phase Detection Based on Muscle Deformation with Static Standing-Based Calibration”. MDPI. 2021 Feb pp.05,13.
- [7] JEOL,“平滑化 (スムージング)”
<https://www.jeol.co.jp/words/emterms/20121023.094657.html#gsc.tab=0> 最終閲覧日 2023/01/31
- [8] ottobock,“上肢の切断レベル”
https://www.ottobock.com/ja-jp/prosthetic_ue/info/amputation_level 最終閲覧日 2023/02/01

付録

A.1 測定データ

FirstVR の性能評価の際に取得したデータを図 A.1 に示す.



図 A.1: FVRDataFiles の QR コード

A.2 ソースコード

Unity で使用したスクリプトをソースコード

ソースコード A.1: CalibrationManager

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.XR;
using UnityEngine.UI;
using UnityEngine.Animations;
using System.Linq;
using FVRlib;

public class CalibrationManager : MonoBehaviour
{
    public Animator animator;
    public GameObject Collider;

    public Transform GrapPoint_sphere, GrapPoint_cylinder, GrapPoint_cube;

    // FVR
    public FVRConnection fvr;
    public FVRGesture gesture;

    //Control variables
    int samplesPerSecond = 0;
    int roundLength = 0;
    int tCalibRounds = 0;
    int ntCalibRounds = 0;
```

```

// Texts
public Text samplesPerSecondTxt;
public Text roundLengthTxt;
public Text tCalibRoundsTxt;
public Text ntCalibRoundsTxt;

// Images
public Image targetImg;
public Image nonTargetImg;
public Image testImg;

//Buttons
public Button targetBtn;
public Button nonTargetBtn;
public Button resetBtn;
public Button[] varBtns;

bool cube_col;
bool cylinder_col;
bool sphere_col;

// int grap;

// Start is called before the first frame update
void Start()
{
    fvr = FindObjectOfType (typeof(FVRConnection)) as FVRConnection;

    // Create a new custom gesture
    gesture = fvr.gestureManager.RegisterCustomGesture ("gestureName");

    // Display the default settings
    samplesPerSecond = fvr.gestureManager.calibrationSamplesPerSecond;
    roundLength = (int)fvr.gestureManager.calibrationRoundLength;
    UpdateTexts ();

    // Button control
    targetBtn.interactable = false;
}

void Update()
{
    Vector3 origin = Collider.transform.position;
    Vector3 direction = -Collider.transform.forward;
    Ray ray = new Ray(origin,direction);
    Debug.DrawRay(ray.origin, ray.direction*0.2f, Color.red, 0.01f);
    if(Physics.Raycast(ray, out RaycastHit hit, 1.0f))
    {
        Debug.Log(hit.collider.gameObject.name);
        if(hit.collider.gameObject.name == "Cube")
            cube_col = true;
        if(hit.collider.gameObject.name == "Cylinder")
            cylinder_col = true;
        if(hit.collider.gameObject.name == "Sphere")

```

```

        sphere_col = true;
    }

    if(gesture.held == true)
    {
        animator.SetBool("grap_null",true);
        testImg.color = Color.green;
        if(cube_col == true)
        {
            testImg.color = Color.blue;
            animator.SetBool("grap_cube1",true);
            GameObject cube = GameObject.Find("Cube");
            Rigidbody rb = cube.GetComponent<Rigidbody>();
            rb.isKinematic = true;
            cube.transform.position = GrapPoint_cube.position;
            cube.gameObject.transform.parent = GrapPoint_cube;
        }
        if(cylinder_col == true)
        {
            testImg.color = Color.red;
            animator.SetBool("grap_cylinder1",true);
            GameObject cylinder = GameObject.Find("Cylinder");
            Rigidbody rb = cylinder.GetComponent<Rigidbody>();
            rb.isKinematic = true;
            cylinder.transform.position = GrapPoint_cylinder.position;
            cylinder.gameObject.transform.parent = GrapPoint_cylinder;
        }
        if(sphere_col == true)
        {
            testImg.color = Color.yellow;
            animator.SetBool("grap_sphere1",true);
            GameObject sphere = GameObject.Find("Sphere");
            Rigidbody rb = sphere.GetComponent<Rigidbody>();
            rb.isKinematic = true;
            sphere.transform.position = GrapPoint_sphere.position;
            sphere.gameObject.transform.parent = GrapPoint_sphere;
        }
    }
    else if(gesture.held == false)
    {
        testImg.color = Color.white;

        animator.SetBool("grap_null",false);
        animator.SetBool("grap_sphere1", false);
        animator.SetBool("grap_cylinder1", false);
        animator.SetBool("grap_cube1", false);

        cube_col = false;
        cylinder_col = false;
        sphere_col = false;

        //sphere
        GameObject sphere = GameObject.Find("Sphere");
        Rigidbody rb_sphere = sphere.GetComponent<Rigidbody>();
    }
}

```

```

        rb_sphere.isKinematic = false;
        sphere.gameObject.transform.parent = null;

        //cylinder
        GameObject cylinder = GameObject.Find("Cylinder");
        Rigidbody rb_cylinder = cylinder.GetComponent<Rigidbody>();
        rb_cylinder.isKinematic = false;
        cylinder.gameObject.transform.parent = null;

        //cube
        GameObject cube = GameObject.Find("Cube");
        Rigidbody rb_cube = cube.GetComponent<Rigidbody>();
        rb_cube.isKinematic = false;
        cube.gameObject.transform.parent = null;
    }
}

public void ChangeSPS(int dir){
    samplesPerSecond += 1 * dir;
    samplesPerSecond = samplesPerSecond < 1 ? 1 : samplesPerSecond;
    fvr.gestureManager.calibrationSamplesPerSecond = samplesPerSecond;
    UpdateTexts ();
}

/// <summary>
/// You can change the length of the calibration round.
/// This length should always be higher than 0 and making it too long might affect the
/// results in a negative way.
/// Recomended values are 1~3
/// </summary>
public void ChangeRL(int dir){
    roundLength += 1 * dir;
    roundLength = roundLength < 1 ? 1 : roundLength;
    fvr.gestureManager.calibrationRoundLength = (float)roundLength;
    UpdateTexts ();
}

public void SetTargetPress(){
    StartCoroutine (Calibrate (true));
}

public void SetNonTargetPress(){
    StartCoroutine (Calibrate (false));
}

// Reset the calibration data and start all over again
public void ResetCalibrationPress(){
    fvr.gestureManager.ResetPatternData (gesture);
    tCalibRounds = 0;
    ntCalibRounds = 0;
    UpdateTexts ();
    targetBtn.interactable = false;
    nonTargetBtn.GetComponentInChildren<Text> ().text = "Set\nDummy";
    foreach (Button b in varBtns) {
        b.interactable = true;
    }
}

```

```

        }

    }

    // Updates the display texts
    void UpdateTexts(){
        tCalibRoundsTxt.text = tCalibRounds.ToString ();
        ntCalibRoundsTxt.text = ntCalibRounds.ToString ();
        samplesPerSecondTxt.text = samplesPerSecond.ToString ();
        roundLengthTxt.text = roundLength.ToString ();
    }

    /// <summary>
    /// Calibrate the gesture with target or non-target values.
    /// Calibration requires time, and it's best to let the user know what's going on, so
    /// this process is best done in a coroutine.
    /// </summary>
    IEnumerator Calibrate(bool target){
        if (target) {
            // Setting target values
            fvr.gestureManager.SetTargetData (gesture);
            tCalibRounds++;
        } else {
            // Setting non-target values
            fvr.gestureManager.SetNonTargetData (gesture);
            /// The first time we set a target or non-target value, the round length and
            /// samples per second are ignored and the SVM takes only one value with dummy
            /// data then
            /// the dummy data is replaced with real data.
            /// After the first round the FVRGesture.calibrated flag is set to true and you
            /// are ready to start calibrating with real data
            if (gesture.calibrated) {
                ntCalibRounds++;
            }else{
                nonTargetBtn.GetComponentInChildren<Text> ().text = "Set\nNonTarget";
                foreach (Button b in varBtns) {
                    b.interactable = false;
                }
            }
        }
        // We dont wan't multiple coroutines taking the same data so it's good to block the
        // user from starting a new one before this round is done
        targetBtn.interactable = false;
        nonTargetBtn.interactable = false;
        resetBtn.interactable = false;
        float t = 0;
        while (gesture.registering) {
            /// While the target or non-target data is being set, the FVRGesture.registering
            /// flag will be set to true.
            /// A count down or a image fill loading bar is a good way to let the user know
            /// your app is doing something.
            /// Once the porcess is done, the FVRGesture.registering flag will be set to false
            /// , and we will exit this while loop.
            t += Time.deltaTime;
            if(target)

```

```
        targetImg.fillAmount = t / (float)roundLength;
    else
        nonTargetImg.fillAmount = t / (float)roundLength;
        yield return null;
    }
    UpdateTexts ();
    targetImg.fillAmount = 0;
    nonTargetImg.fillAmount =0;
    // After the process is done you can enable whatever buttons you need to proceed with
    // the calibration or move on with your app.
    targetBtn.interactable = true;
    nonTargetBtn.interactable = true;
    resetBtn.interactable = true;
}
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Movemanager : MonoBehaviour
{
    public Vector3 Up_speed,Down_speed,Left_speed,Right_speed;
    bool forwardmove;
    bool backmove;
    bool rightmove;
    bool leftmove;

    public void forwardButtonDown(){
        forwardmove = true;
    }
    public void forwardButtonUp(){
        forwardmove = false;
    }
    public void backButtonDown(){
        backmove = true;
    }
    public void backButtonUp(){
        backmove = false;
    }
    public void rightButtonDown(){
        rightmove = true;
    }
    public void rightButtonUp(){
        rightmove = false;
    }
    public void leftButtonDown(){
        leftmove = true;
    }
    public void leftButtonUp(){
        leftmove = false;
    }
    void Update()
    {
        if(forwardmove == true){
            transform.position += Up_speed;
        }
        if(backmove == true){
            transform.position += Down_speed;
        }
        if(rightmove == true){
            transform.position += Right_speed;
        }
        if(leftmove == true){
            transform.position += Left_speed;
        }
    }
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using FVRlib;

public class handtrack : MonoBehaviour
{
    public FVRConnection fvr;

    // Update is called once per frame
    void Update()
    {
        this.transform.rotation = fvr.centeredRotation;
    }
}
```
