

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

指導教官 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)

ああ
あああ
ああ
あああああああああああああああああああああああああああああああああああああああ
あああああああああああああああああああああああああああああああああああああああ
あああああああああああああああああああああああああああああああああああああああ
あああああああああああああああああああああああああああああああああああああああ

目次

第1章: 序論	1
1.1 研究背景	1
1.2 研究目的	1
第2章: 理論:3D モデルの製作	2
2.1 EinScan HX	2
2.1.1 製品仕様	2
2.2 Blender	3
2.2.1 スムージング	3
2.2.2 ボーン構成	3
第3章: 理論:Unity・FirstVR	4
3.1 Unity	4
3.1.1 シエーダー	4
3.1.2 オブジェクト	4
3.1.3 衝突判定	4
3.1.4 保持表現	5
3.1.5 ビルド	5
3.2 FirstVR	6
3.2.1 デバイス構成	7
3.2.2 トラッキング	7
3.2.3 ジェスチャ認識	7
第4章: 研究手順	8
4.1 使用器具	8
4.2 3D スキャナ	8
4.2.1 3D モデルの取り込み	8
4.3 Blender	8
4.3.1 スムージング処理	8
4.3.2 ボーン配置	9
4.4 Unity	9
4.4.1 オブジェクトの構成	9
4.4.2 ステージ構成	10
4.4.3 アニメーション設定	10
4.4.4 オブジェクト保持	10
4.5 FirstVR	10
4.6 ビルド	10
4.7 評価	10
第5章: 研究結果	11
5.1 FirstVR	11
5.2 シミュレータの構成	13
5.3 シミュレータの定性評価	15
第6章: まとめ	16
第7章: 今後の課題	17
参考文献	19

第 1 章 序論

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] で検討されているが、3D モデルのリアリティについて検討されていなかったため、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。このトレーニングシステムのインターフェースに “FirstVR” を用いることで、実際の筋電義手を使用することなく、安価で訓練が行えると考えた。また、近年では “カグラ”[6] という製品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”[6] の活用方法の一例としても後続の研究に活用していただきたい。

1.2 研究目的

本研究の目的は VR トレーニングシステムのリアリティについて入力デバイス・VH のモデルという観点から研究を行いそのリアリティが幻肢痛の緩和や運動機能の訓練効果にどのように影響を与えるかを検討することが最終目標である。まずその第 1 段階として、FirstVR を用いた VR トレーニングシステムを構成し、どの程度リアルにシミュレートできるかを検討した。また、FirstVR がどの程度の精度でジェスチャを検出できるか検証も行った。なお、本研究におけるリアリティの評価方法として、実験協力者を対象とした定性評価アンケートを用いて評価した。

第2章 理論:3D モデルの製作

本章では、本研究で利用した“Ein Scan HX”をはじめ、各種ソフトウェアの利用方法およびその使い方について解説していく。また、機器を選定した理由についても記述する。

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。製品仕様については次項で解説する。



図 2.1: EinScan HX

2.1.1 製品仕様

EinScan HX の製品仕様について表 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	
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	STL,OBJ,PLY,ASC,3MF,P3	
本体サイズ	108×110×237 mm	
本体重量	710g	

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、3D オブジェクト表面をペイントソフトのようになぞるだけで表面を平滑化し、頂点を揃える機能である。

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に VH の変形を制御するために用いている。

第3章 理論:Unity・FirstVR

3.1 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。

3.1.1 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。

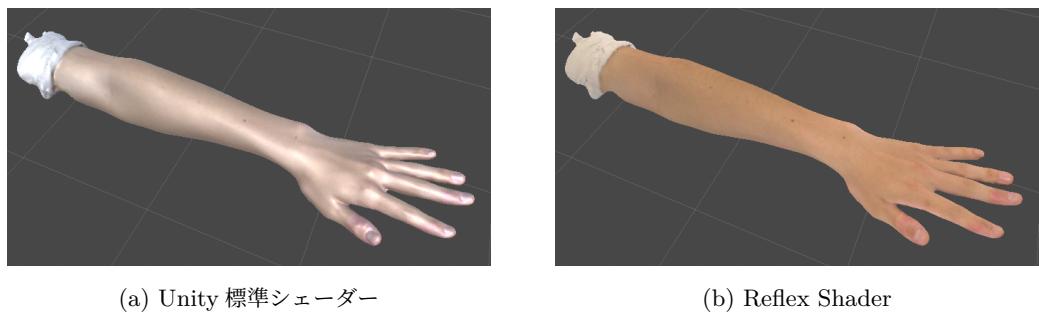


図 3.1: シェーダーによる 3D モデルの違い

3.1.2 オブジェクト

Unity におけるオブジェクトは Unity エディタ上に配置されるものを総称してオブジェクトと呼び、各オブジェクトにはシーンに配置された段階でシーン上のオブジェクトの座標を示す Transform というコンポーネントが付加されている。その他にも衝突判定を付加する Collider コンポーネントなど様々な種類のコンポーネントを付加することができる。

3.1.3 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付加された 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。

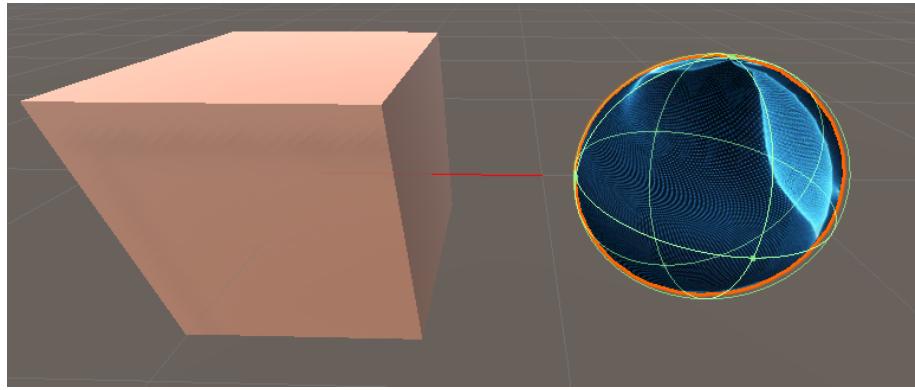


図 3.2: Raycast のイメージ図

図 3.2 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。

3.1.4 保持表現

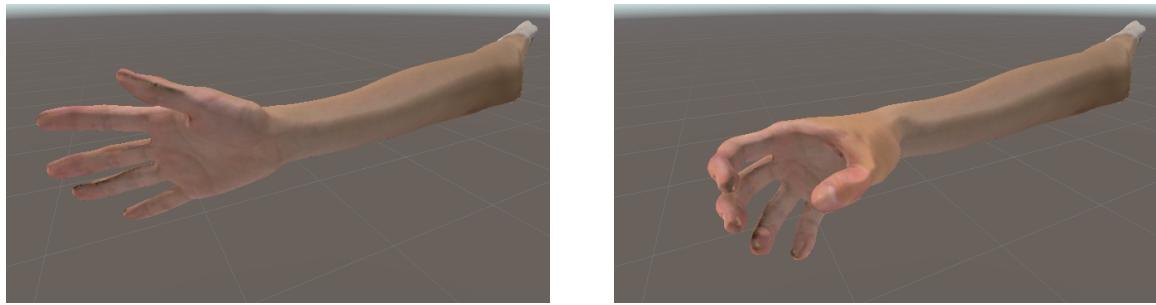


図 3.3

3.1.5 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。

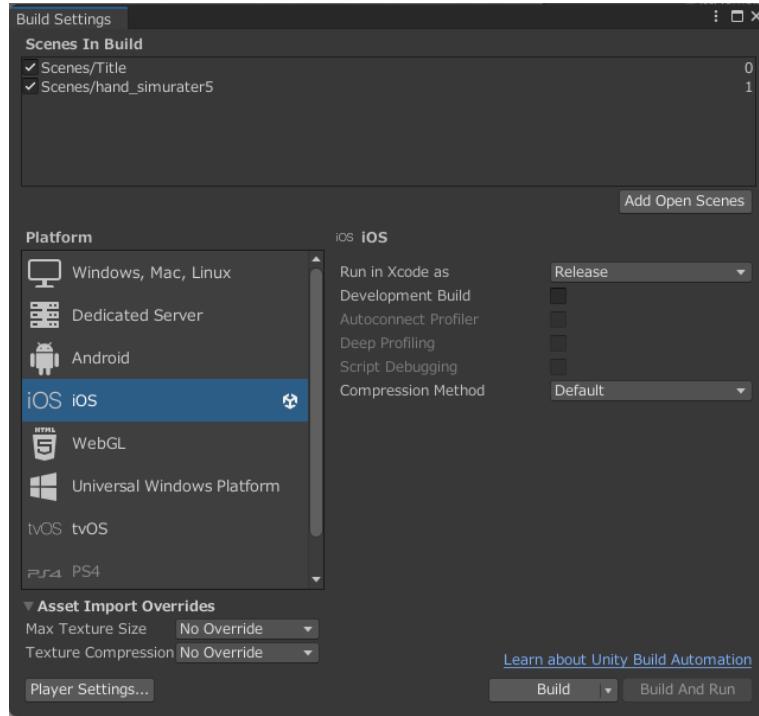


図 3.4: BuildSettings 画面

iOS,Android OS に関してはビルド・実行方法が異なるため、それぞれ以下に解説する。

iOS

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る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。

3.2 FirstVR

FirstVR は

3.2.1 デバイス構成



(a) 表面画像



(b) 裏面画像

図 3.5: FirstVR

3.2.2 トラッキング

FirtVR 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。

3.2.3 ジェスチャ認識

第 4 章 研究手順

4.1 使用器具

本研究での使用器具を表 4.1 に示す。

表 4.1: 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

以下に本研究で使用したソフトウェアを表 4.2 に示す。

表 4.2: 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	

4.2 3D スキャナ

4.2.1 3D モデルの取り込み

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した本研究では、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 形式を選択した。

4.3 Blender

4.3.1 スムージング処理

.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った。

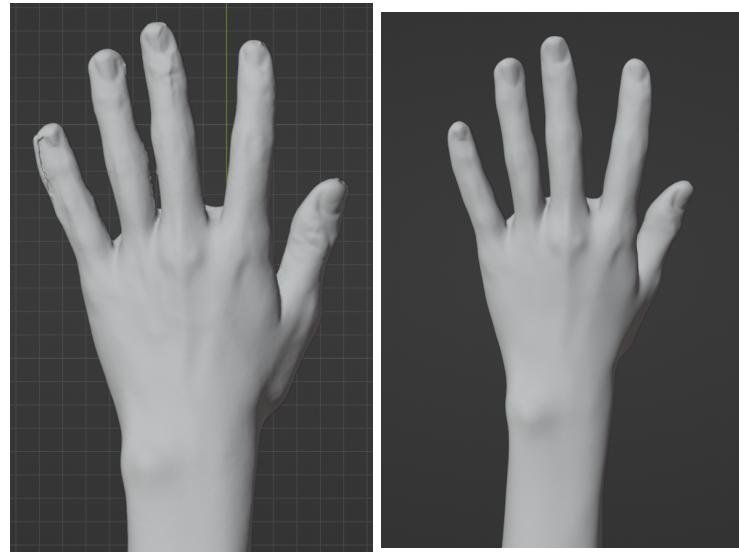


図 4.1: スムージング処理比較

4.3.2 ボーン配置

スムージング処理後の 3D モデルを基準に図のようにボーンを配置した。

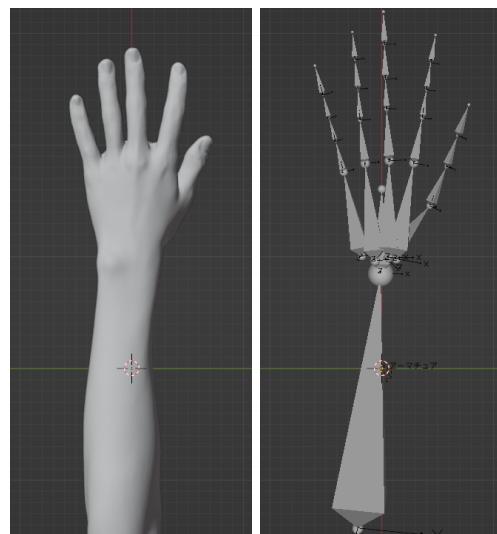


図 4.2: ボーン配置図

4.4 Unity

4.4.1 オブジェクトの構成

- Player
- Target
- Stage

4.4.2 ステージ構成

4.4.3 アニメーション設定

アニメーションクリップ作成

アニメーター設定

4.4.4 オブジェクト保持

4.5 FirstVR

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、以下の手順で FirstVR の評価を行った。

1. 装着位置の決定

FirstVR の動作原理より、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、被験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）で装着した

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、手を握るジェスチャの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した。

4.6 ビルド

4.7 評価

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、各被験者・各 sample 数ごとの評価指標として総変化量 X を定めた。総変化量の算出は測定回数 $s:1 \sim 5$ 、チャンネル数 $r:0 \sim 13$ 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

$$X = \frac{1}{5} \sum_{s=1}^5 \sum_{r=0}^{13} |N_r - M_{sr}| \quad (4.1)$$

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ることにより、どの sample 数をシミュレータで用いれば良いかを検討した。

第 5 章 研究結果

5.1 FirstVR

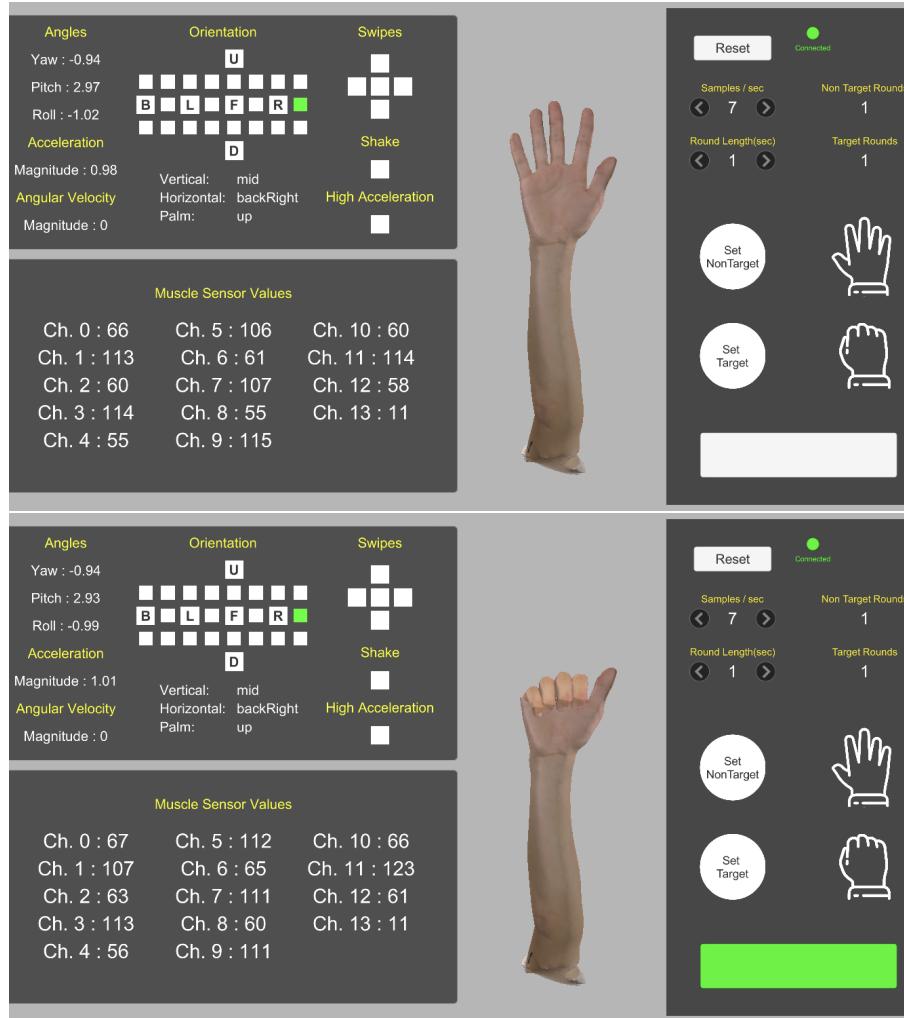


図 5.1: 測定用アプリケーション

表 5.1: 各被験者における sample 数ごとの総変化量一覧

num/sample	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
14	20.6	17.4	21.6	29.0	19.0	22.8	27.6	32.2	56.8	44.0	31.2
16	19.8	22.0	11.6	10.0	16.8	17.4	12.4	17.6	21.6	15.8	11.4
17	50.8	58.4	51.2	56.6	67.4	37.6	45.8	52.4	51.6	59.6	56.2
18	35.2	50.2	62.6	36.4	27.2	39.6	38.8	43.4	45.4	52.4	48.8
19	49.4	42.2	41.2	36.4	49.4	34.4	33.2	34.4	22.2	13.8	36.0
20	23.8	26.8	36.2	38.0	28.0	35.2	34.0	24.4	29.2	39.6	25.8
21	33.6	11.8	10.4	16.8	17.8	11.0	19.4	13.6	24.4	13.2	9.6
23	32.6	36.4	33.0	17.6	22.6	26.8	32.2	25.2	20.8	30.4	13.8
24	32.8	42.6	36.6	30.6	26.4	41.4	32.6	20.2	25.0	15.8	17.6
25	42.2	27.6	37.8	33.8	39.2	50.0	34.6	38.0	38.8	26.8	24.2
26	51.8	40.4	44.0	44.6	41.4	39.2	42.8	38.4	41.6	43.0	30.0
27	38.0	30.6	38.2	35.6	23.8	20.4	24.4	24.2	22.6	30.6	16.8
28	25.2	33.0	32.0	23.8	28.0	20.0	31.4	13.6	19.6	19.4	21.2
32	39.8	32.8	30.2	38.2	34.8	22.6	30.4	24.0	24.0	12.6	20.6
34	68.8	78.0	50.4	51.6	44.2	50.8	40.0	48.2	46.4	48.8	62.0
35	20.4	29.8	40.6	45.0	23.8	25.2	27.4	39.0	25.2	28.8	39.6
36	32.6	29.8	21.6	19.0	22.4	17.6	21.2	28.2	41.4	35.6	20.8
38	23.2	8.0	13.6	23.2	21.2	25.4	24.0	12.6	16.0	24.4	17.8
41	44.4	25.6	21.8	19.4	12.2	12.6	13.4	22.4	8.2	11.8	27.2

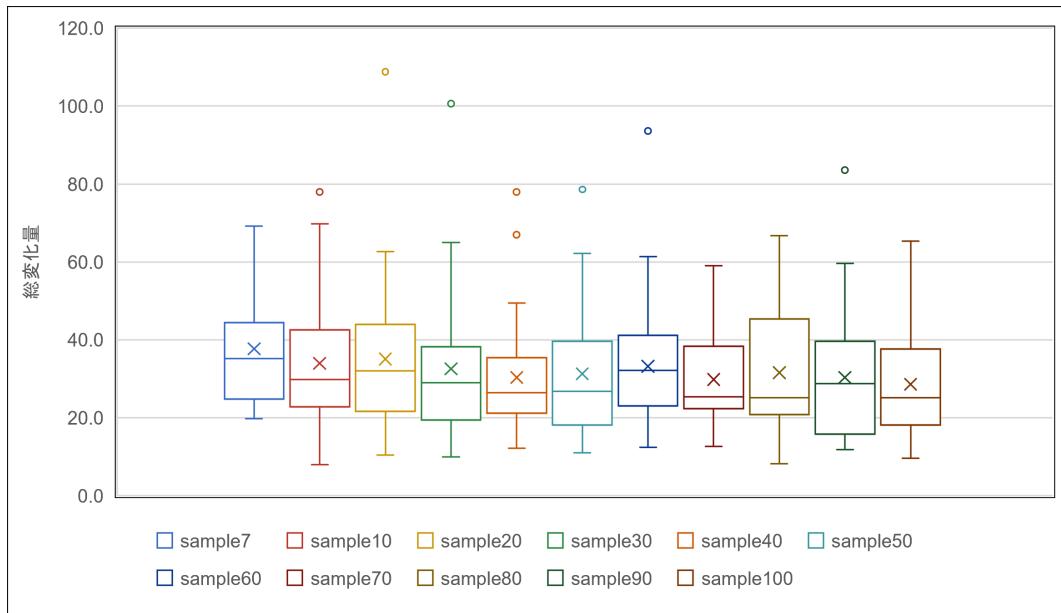
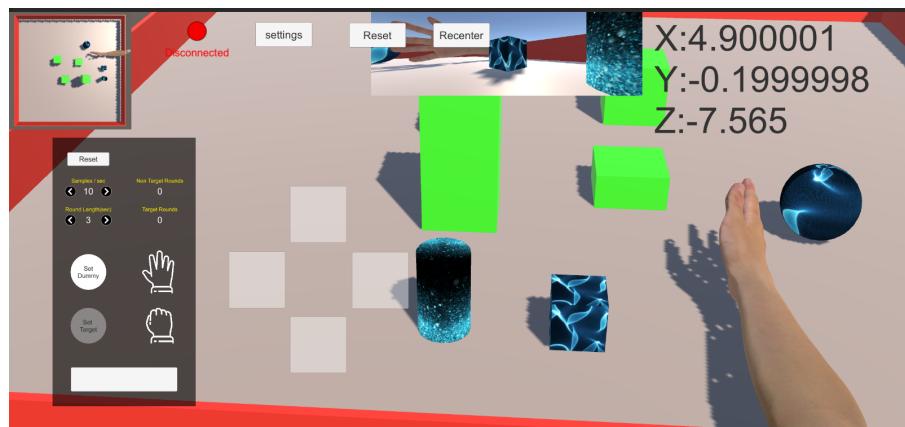


図 5.2: sample 数ごとの総変化量分散

5.2 シミュレータの構成



(a) 通常時の画面構成



(b) メニュー起動時の画面構成

図 5.3: iOS 版シミュレータの実行画面

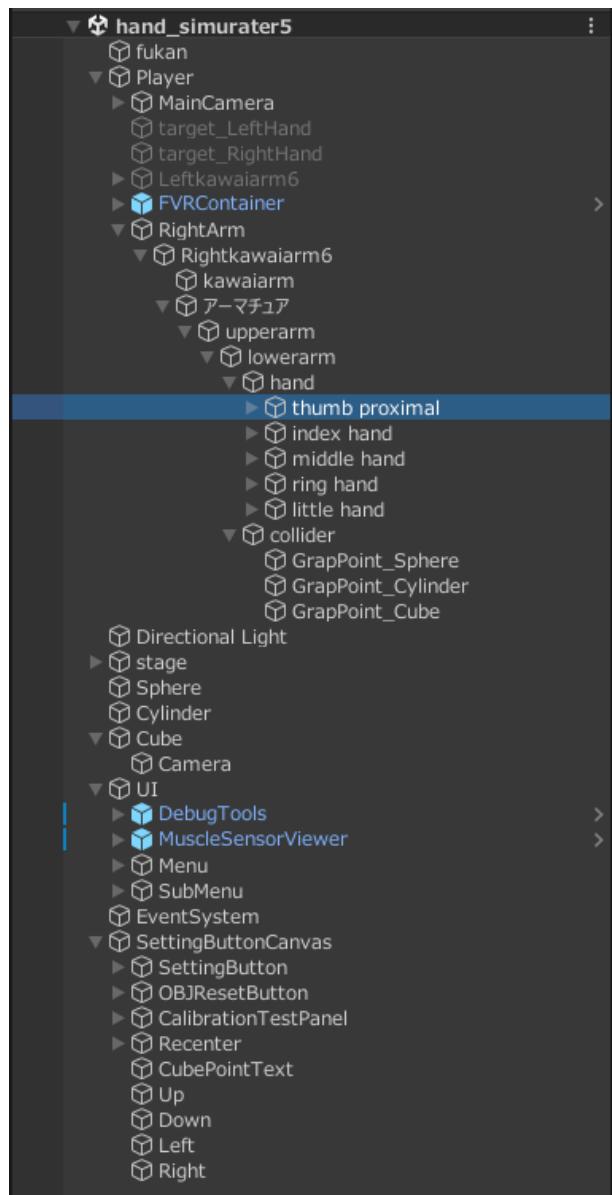


図 5.4: シミュレータのオブジェクト構成

5.3 シミュレータの定性評価

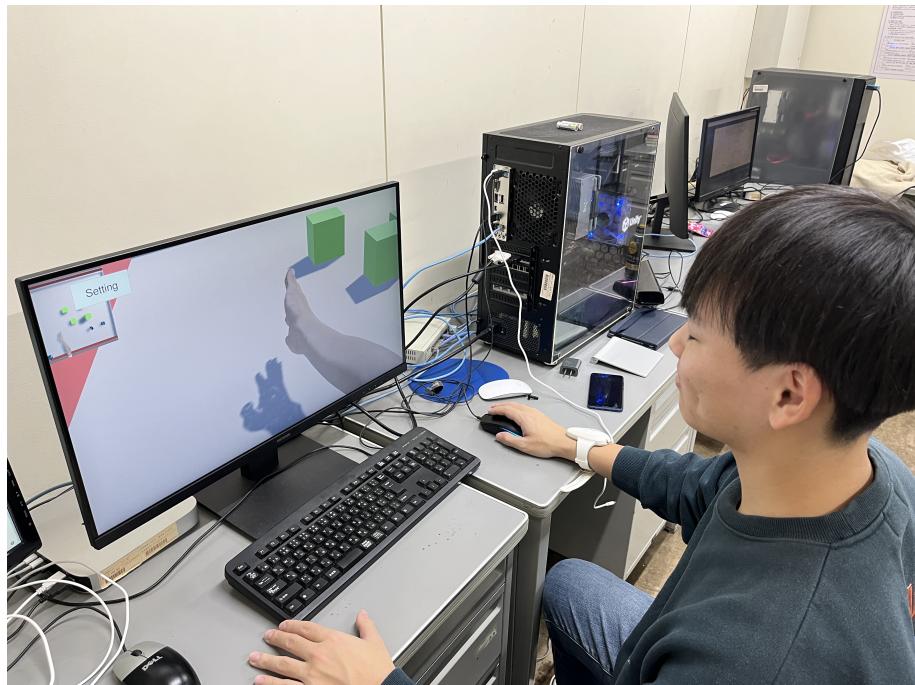


図 5.5: 定性評価の様子

第6章 まとめ

第 7 章 今後の課題

現段階では、シミュレータとして最低限の要素を追加した。しかし、FirstVR における掴みジェスチャの認識精度の低さから本研究で検討しようとしたリアリティについて検討できなかった。

謝辞

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] H2L.Inc.,Tokyo106-0032,Japan;satoshi.hosono@h2l.jp
- [4] Tamon Miyake, etal“Gait Phase Detection Based on Muscle Deformation with Static Standing-Based Calibration”. MDPI. 2021 Feb
- [5] mediVR.Inc.,<https://www.medivr.jp/>
- [6] 株式会社サンステラ, <https://www.einscan.jp/einscan-hx>
- [7]

付録

ソースコード 7.1: 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;
}
}

```

ソースコード 7.2: handtrack.cs

```

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;
    }
}
```

ソースコード 7.3: MoveManager.cs

```
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;
}
}
```
