VR コントローラの常時モーション検出システムの実装

佐藤　凌誠　（指導教員　西村　俊二）

令和２年１月２４日

Continuous motion detecting using VR controller

RYOSEI SATO　(ACADEMIC ADVISOR　SHUNJI NISHIMURA)

概要：VRは近年身近なものになってきており，グラフィック技術の発展も進んでいるが，VR空間内での操作方法は完成度が高いとは言えない． 現在，ニューラルネットワークを用いたジェスチャによる操作が可能にはなっているものの，その操作方法ではユーザが検出してほしい旨を入力によって示しており，自動的な検出ではない．そこで本研究では，自動的なジェスチャの検出を可能にするシステムの実装を行った．

キーワード: VR，ニューラルネットワーク

1．緒言

近年，インターネット上では，3DCG を用いた創作が勢いを増している．2019 年は特に動画サイトにおいて，バーチャルユーチューバーと呼ばれる，演者が VR 空間でキャラクターの 3D モデルを操作し，架空のキャラクターになりきって動画配信を行う，という形態が爆発的なブームを起こした．これにより，VR 空間でキャラクターに”なりきる”コンテンツが流行している．また，VR コンテンツの体験のみに焦点を当てたレジャー施設なども増えており，VR はより身近なコンテンツとなってきている．

コンテンツとしてだけが VR の利用方法ではなく，外科手術や飛行機の操作といった，高いリスクやコストを伴う教習や，工事現場や災害時など，安全教育などにも VR 空間でのシミュレーションが運用されている．さらに医療の面で，自閉症や学習障害，摂食障害などの心的療法にも，VR 空間でのシミュレーションは高い効果が示されており[1]，さまざまの職種において重要な役割を果たしている．

これらの VR の活躍において重要なのは，高い臨場感，没入感が得られていることである．VR 空間での出来事を，実際の出来事として錯覚するほどの没入感がなければ，コンテンツとしての魅力も薄れ，医療などでの実用も不可能である．この得られる没入感を更に高めることができれば，より良い VR 空間でのシミュレーションが可能になると考えた．

VR を体験する中で，高い没入感を得るためには，操作が直感的であるかどうかは重要な点であると考える．まず，操作者が VR 空間に対して操作を行う方法には，基本的な二種類が存在する．操作者が持っているコントローラを動かす方法と，単純にコントローラのボタンを押す方法である．例えば図 1 のように，視界のインターフェースに，コントローラから伸びるカーソルを合わせて，ボタンを押して選択する，といった操作や，コントローラを VR 空間での何かの物体に近づけ，触る，というような操作は，先ほどの二つの操作を組み合わせて行っているにすぎないのである．

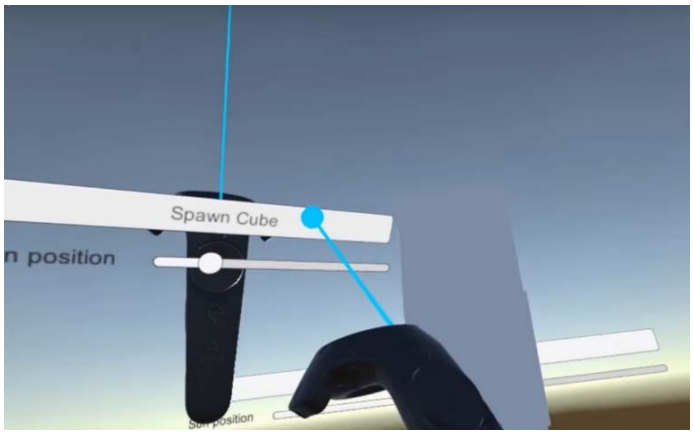

図 1　インターフェース操作

物体を手に取る操作を行うには，コントローラを物体まで近づけるが，これは現実で物体を取る動作とあまり差がない．しかし，インターフェースの中から特定のアイテムを選び，出現させるというような操作は，インターフェースがまるで目の前に存在しているようなリアリティは体験するかもしれないが，その操作の過程と結果は大きく乖離しており，直感的であるとはいえない．直感的な操作というのは，この操作をしたからこの結果になる，という流れを抵抗なく認識できることが重要で

ある．このことから，先ほどの 2 種類の操作方法のみでなく，操作者にとってわかりやすい“動作”を定め，それを操作に取り入れることで，より直感的な，新しい操作方法が確立できるのではないかと考えた．これは，従来のハンドジェスチャ等に類似しており，操作者による複数個の決められた動作に対して，何らかの異なるアクションを関連付けて，その動作をきっかけにしてアクションを起こす，という操作方法である．図 2 は，抜刀する動作に剣を生成する処理を紐づけている．

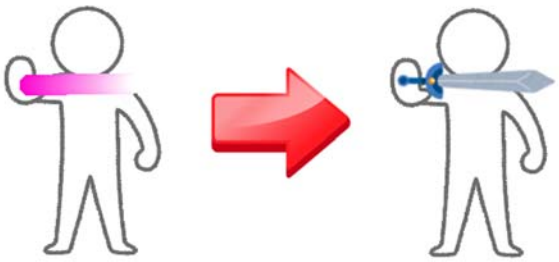

図 2　ジェスチャによる剣の生成

この操作方法を実現するには，事前に複数個の動作を登録し，操作者の動作が，登録した動作のいずれかと同じものか，あるいは異なるものかを判別しなければならない．なお，これについては先行研究[2]によって，ニューラルネットワークと呼ばれる深層学習を用いてある程度の実装が可能であることが示されている．具体的には，操作者が動作の初めにボタンを押し，終わりに離す，というようにボタン入力によって操作者が自分の動作を一定時間分に切り抜き，その切り抜いた動作をニューラルネットワークで判別する，というものである．しかし，操作にボタン入力を用いるのであれば，それはボタン入力に動作が追加されただけで，新しい操作方法を確立しているとは言えない．そこで本論文では，ジェスチャとして動作を行っている，という操作者の意思を示さず，動作の計測のみによってのジェスチャの検出を可能とするためのニューラルネットワークを用いた評価を行い，考察した．

2. 理論

2.1　検出

2.1.1　　ニューラルネットワークとは

人間の脳内には，図３のようなニューロンと呼ばれる神経細胞が存在する．このニューロンは，シナプスと呼ばれるシグナル伝達の結合によりネットワークを形成し，各ニューロン間で信号を送りあっている[3]．シナプスは結合強度に差があり，強く結合したニューロンと弱く結合したニューロンが存在する．このニューロンは脳内に無数に存在し，そのニューロンへ入力となる刺激が入ってきたとき，活動電位と呼ばれる電気信号を発生させ，他の細胞に情報を伝達する．ひとつのニューロンに複数のニューロンから入力したり，活動電位がおきる閾値を変化させたりして，非常に複雑な電気信号の回路のようなものができ，この中で情報の修飾が行われる．このようにニューロンとシナプスがネットワークのようなものを構成し，学習していくことで，人間の脳は情報処理を得ているのである．

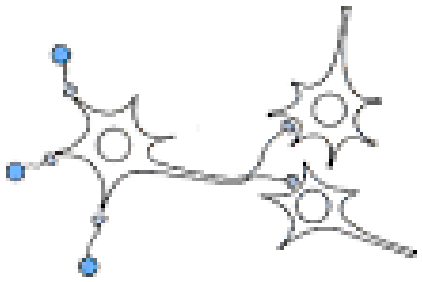

図 3　ニューロンの結合

ニューラルネットワークは，人工ニューロンとも呼ばれており，上記の通り人間の脳内で行われている，ニューロンのネットワークを用いた情報処理の，結合の強さや電気信号の送受信といった特性を，プログラム上で再現して，深層学習を行い，問題解決を行うための計算モデルである．ニューラルネットワークには様々な種類が存在するが，今回はその中でも順伝播型ニューラルネットワークと呼ばれる，基礎的なものを使用する．

2.1.2　　順伝播型ニューラルネットワーク

順伝播型ニューラルネットワークには，図４に示すように入力，出力の二つの層からなる単純パーセプトロンと，その間に隠れ層と呼ばれる層を一層以上間に追加した多層パーセプトロンが存在する．今回は隠れ層が一層である多層パーセプトロンを用いる．

各層はノードと呼ばれるものが連なって形成されており，これは脳内のニューロンと同じような役割を働く．入力層には入力の数，出力層には出力の数だけのノードがあり，これらがそれぞれの入力と出力に対応する．隠れ層のノードは，バイアスと呼ばれる個別の値を保持している．ノードは前後の層の全てのノードと結合しており，それぞれに異なる結合の強さを表す値が存在する．これは重さと呼ばれ，脳内でのシナプスと同じような役割である．入力層のノードにそれぞれ入力の値を渡すと，各ノードは次の層の各ノードに対して，そのノードとの結合の重さを入力の値に掛けた値を渡す．次の層は，その受け取った値にそれぞれが持つバイアスの値を足して，2.1.3 の活性化関数に掛ける．そうして得られた値

がそのノードの持つ値となる．この値は，また同じように次の層へと渡され，同じように計算される．このような過程を出力層まで繰り返し，最後に出力層各ノードが受け取った値が，出力値となる．

バイアスと重みを用いて出力値を算出するが，このバイアスと重みの値は初めから定まっているのではなく，入力に対して正しい出力値となるような値を導き出さなければならない．これがニューラルネットワークの学習である．教師を用いて，どのような入力に対してどのような出力となるのかのサンプルのデータを与え，それを基に，2.1.4 に示す誤差逆伝播法を用いて学習していく．

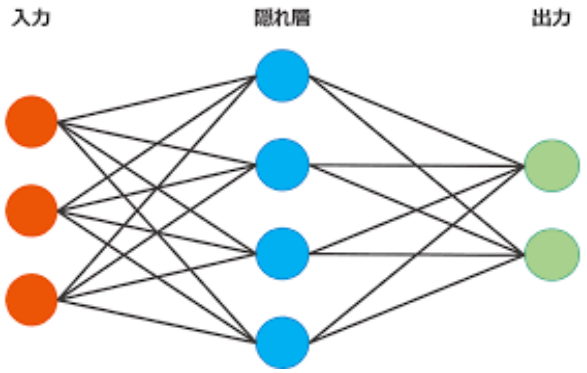

図 4　ニューラルネットワーク

2.1.3　活性化関数

活性化関数は，ニューラルネットワークにおいて，ネットワークを強化するために適用される非線形関数であり，図５のように隠れ層が次の層に値を渡す際に適用される[3].

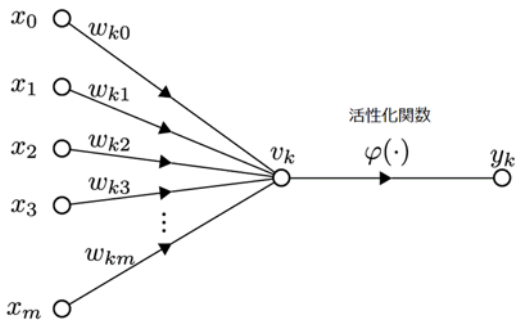

図 5　活性化関数

この活性化関数がなければ，ニューラルネットワーク内で行われる計算は，入力に重みの合計を掛けるだけの線形で単純な計算で済んでしまい，これだけでは問題解決をするだけの能力も生まれない．各ノードがそれぞれ活性化関数を持つことで，非線形化され，重みとバイアスも大きな意味を持つのである．今回は誤差逆伝播法を用いるので，その際によく用いられるシグモイド関数と，ソフトマックス関数を用いる．

シグモイド関数は，(2.1)に示される通り tanh 関数であるが，これは(2.2)に示される標準シグモイド関数を線形返還し，単純化したものである．この関数は微分の計算が容易であることも有用である．入力の絶対値がどれだけ大きくても，出力値は図 6 に示すとおり，-1 から+1 の間のいずれかとなる．

$$\varphi(x) = \tanh x \tag{2.1}$$

$$\varphi(x) = \frac{1}{1+e^{-x}} = \frac{\tanh(x/2)+1}{2} \tag{2.2}$$

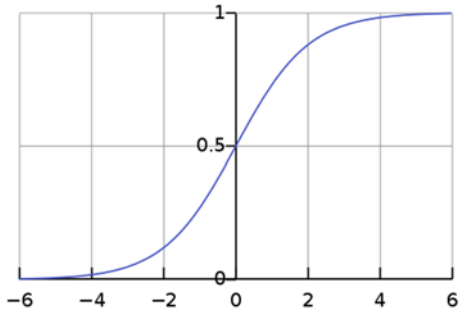

図 6　出力値

ソフトマックス関数は，(2.3)に示されるように，分類が多クラスである問題の際に，すべてのクラスの合計が１になるよう，各クラスへの出力値を算出する関数である．

$$S(y_i) = \frac{e^{y_i}}{\Sigma_j e^{y_i}} \tag{2.3}$$

2.1.4　誤差逆伝播法

誤差逆伝播法は，図 7 に示すように，誤差を最小化して，関数を近似するするための手法である[3]．流れは以下の通りである．

1. ニューラルネットワークに教師データを与え，その入力に対する出力値を求める．
2. 得られた出力値と教師データとの差から，出力層の各ノードについての誤差を算出する．
3. 個々のノードの期待される出力値と倍率，要求された出力と実際の出力の差（局所誤差）を計算する．
4. 各ニューロンの重みを，局所誤差が小さくなるよう調整する．
5. より大きな重みで接続された一つ前の層のノードに対して，局所誤差の責任があると判定する．
6. そのように判定された層のノードの，さらに前の層のノードについて同様の処理を行う．

この手法は名前の通り，誤差の存在するノードから逆方向に伝播するように変更を加えていく手法であり，これを逆伝播と呼ぶ．通常は変更が素早く収束し，誤差の原因の局所解を求めることができる．局所誤差が小さく

なるように重みを変更するためには，損失関数と呼ばれる誤差を表現する関数と，それを微分した導関数が用いられる．

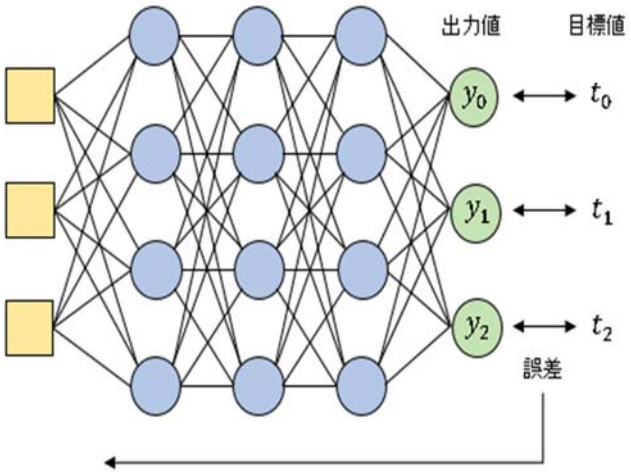

図 7　誤差逆伝播法

2.1.5　誤差と損失関数

学習を行っていく過程で，ニューラルネットワークの出力値と求められている正しい出力値の間に誤差が発生するが，それが目標とする値からどれほど離れているのかを精度として評価する関数を損失関数と呼ぶ．今回は損失関数として，二乗和誤差や残差平方和と呼ばれるもの(2.4)を用いる[4].

$$L = (o_1 - T_1)^2 + (o_2 - T_2)^2 + \cdots + (o_n - T_n)^2 = \sum_{n=1}^{N}(o_m - T_n)^2 \quad (2.4)$$

今回のニューラルネットワークは出力値が 0~1 であり，求められている出力値は 0 か 1 であるため，残差平方和も 0~1 の間となり，これは学習における計算において非常に有用である．

この関数を微分し，導関数を得ることで，誤差が収束する方向へと修正を行うことができる．

2.1.6　算出過程

ニューラルネットワークとコントローラの座標を用いて，操作者の動作が事前に登録したどの動作とどれほど似ているかを算出する．

操作者の動作は，コントローラの三次元座標の連続として計測される．これを 2.2 に示す処理をしてから，入力としてニューラルネットワークの入力層に渡す．出力は登録された動作の数だけあり，それぞれ 0 から 1 の実数で，合計で 1 になるように評価される．最も評価値の高かった動作が操作者の行った動作と判定されるが，このとき評価値が 0.9 を下回っていた場合は，操作者は該当するどの動作も行っていなかったと判定する．

2.2　入力

2.2.1　計測

コントローラの座標を取得する際，座標は自分（ヘッドギア）を中心とする相対座標となり，図 8 のように鉛直上向きが Z 軸，自分（ヘッドギア）の見ている方向の Z 軸に垂直な成分が Y 軸, その二つに垂直な向きが X 軸の方向となる．VR 空間に対する絶対座標はこの計測には関係しない．ヘッドギアとの相対座標が変わらなければ，Z 軸回転をしても計測される座標は変わらない．

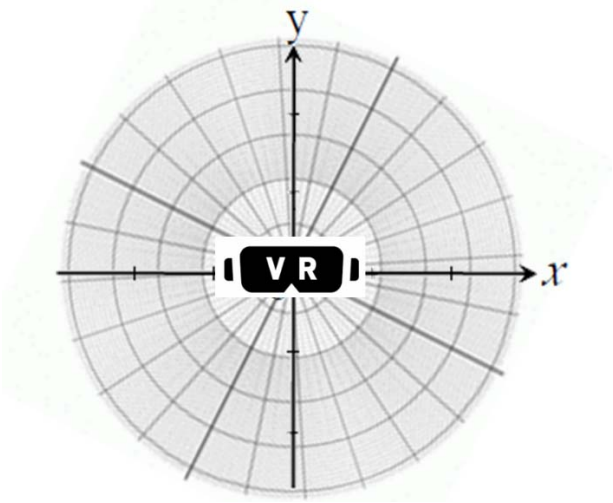

図 8　座標系

2.2.2　前処理

従来の方法では，コントローラの座標の連続をそのまま入力として用いていたのではなく，記録した座標に正規化などの処理を加えてから入力としてニューラルネットワークに渡していた．この処理を工夫することで，下記の誤検出を抑えることができるのではないかと考える．

2.2.3　正規化

まず, 3 次元座標の連続から 11 の個座標を等間隔に取り出し, それらの座標軸ごとの最小値, 最大値を minX, minY，minZ，maxX，maxY，maxZ とする．全ての点に対して，(2.4)で示されるように変換行列を用いて座標を移動させることで，全ての座標が図 9 のように 0~1 に正規化される．

$$\begin{bmatrix} \frac{1}{maxX} & 0 & 0 & -minX \\ 0 & \frac{1}{maxY} & 0 & -minY \\ 0 & 0 & \frac{1}{maxZ} & -minZ \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \quad (2.4)$$

これにより，例えば空中に大きな輪を描いた動作と小さな輪を描いた動作は，同じような座標として正規化される．

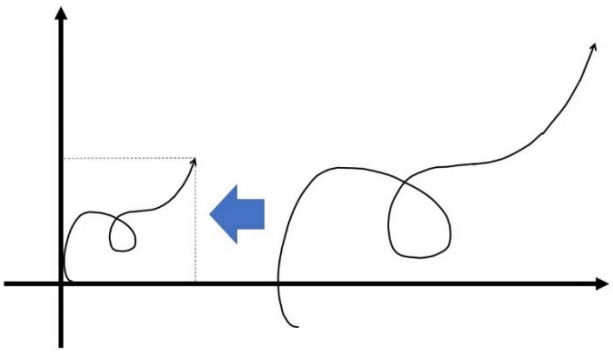

図 9　正規化

2.3　常時検出

2.3.1　変更

2.1.6 に示した検出方法は，操作者が動作を自分の意志で切り抜き，それを事前登録した動作に分類する，というものであった．本研究では操作者の意図によって動作を切り抜かず，常に検出することを可能にする．そのため，一定の長さで操作者の動作を記録し，ニューラルネットワークで出力を計算する，という動作を常時行い続ける必要がある．操作者が VR 空間で操作をしている瞬間，ニューラルネットワークは一定時間分過去から現在にかけて記録されいた動作を検出に掛ける．これを一定間隔ごとに繰り返すこととなる．この際に起こる問題として，負荷の上昇と，誤検出の多発が予想される．

2.3.2　負荷の上昇

本来操作者が意図的に動作を切り抜き，検出を行っていたため，フレーム毎に検出を行った場合，負荷は毎秒 60 回分掛かり，従来とは比べものにならない負荷がかかる．そのため VR 空間内で快適なパフォーマンスを保ち続けられないことが懸念され，検出の軽量化と，適切な検出回数の模索が課題となる．

2.3.3　誤検出

従来の検出は，操作者の検出の意図が明確であったが，今回は操作者の意図に関係なく検出を行うため，検出を意図していないのに検出してしまう誤検出が発生する可能性がある．検出を意図している動作を検出できない未検出が発生することは少ないと予想されるので，意図しない検出を防ぐことが重要な課題となる．特に座標の正規化は，4.1 に示すように意図した検出を分類するためのものであり，検出すること自体には適していないと考えられる．そのため，座標の前処理を工夫することによって，誤検出の防止を図る．

3. 負荷について

3.1　実験環境

使用する PC は GPU が RTX 2080 Ti, CPU が Intel Core i9 9800Hz，RAM は 16GB のもので，HTC 社等が公表している，VR コンテンツを体験するために必要な性能[5]は十分に満たしている．動作の計測には HTC Vive[6]と Unity[7]を用いる．座標は Unity の 3D 空間内で計測されるものを使用する．ニューラルネットワークなどは，Unity 内のスクリプト(C#)で定義されており，実際の計算などは Unity 上で行われる．操作者はヘッドギアを頭部に被り，両手にコントローラを持ち，操作する．部屋の上部の対角線上に 2 つセンサーが備えられており，コントローラの座標は正確に計測できる．

3.2　検出と負荷

操作者が意図した場合のみであった検出を，1 秒に 60 回のフレーム毎の検出に変更し，挙動の確認を行ったところ，初めは Unity の挙動が安定せず，操作が難しい程の遅延が発生した．そこで，デバッグとして行っていた，動作の軌跡上にオブジェクトを配置したり，リアルタイムで処理結果をインターフェース上に表示するといった視認性を上げるための処理などを減らし，軽量化を行ったところ，全く挙動に影響がない程度まで処理を軽くすることができた．オブジェクトを発生させる処理が特に負荷が大きいようだが，ニューラルネットワーク自体の処理は PC の性能の高さもあり，挙動には問題がなかった．

4. 検出方法

4.1　正規化と誤検出

挙動は問題ないが，検出の面では検出を意図した動作を行っていないのに検出されてしまう，誤検出が頻繁に発生し，正常な操作は不可能であった．検出を意図していない動きの中でも特に小さく湾曲するような動きは，正規化によって図 10 のように遥かに大きな弧を描く動作と等しく認識され，誤検出の要因となっていた．

常時の検出ではなく，操作者が自分の動作を切り抜いて分類する場合，そもそも誤検出にはならず，分類するできるかできないかでしかない．正規化を行うと動作が大きく拡大されてしまうような小さい動作が計測されることも考えられず，問題は発生しない．しかし，操作者が意図するかどうかに関係なく検出を行っている場合，登録されている動作に比べて非常に小さな動作でも，正規化によって引き延ばされ，登録されている動作と同じであると評価されてしまうことがある．このため，正

規化は検出を意図した動作の分類には適しているが，動作が登録されているものかどうかを評価するには適しておらず，常時検出に適した新たな前処理の方法を考える必要がある．

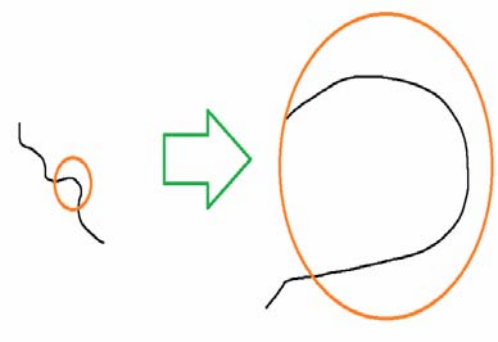

図 10　動作の拡大

4.2　前処理

4.2.1　　手法の変更

座標を正規化する処理は，操作者の検出する意思が明確な場合を想定したもので，これは常時の検出には適していない．そこで，根本から計測手法を変更し，常時検出に適した計測を行うことを目標とする．

4.2.2　　操作者の意思

正規化を行う手法では，どの動作も 0 から 1 に正規化されてしまうため，座標の変化のみが影響し，座標そのものは意味を持っていなかった．そうではなく，動作の座標そのものに意味を持たせるようにすれば，誤検出が抑えられると思われる．

操作者が検出を意図する動作を行うとき，大概は同じ座標でその動作が行われることが予想される．例として，自分の正面に円を描く動作を登録する際に行っていた場合，自分の右側に円を描く動作をして検出を意図することは考えにくく，登録したときと同じような座標で動作が繰り返されるのではないかと考えた．この同じ座標で行われるという点を活かして，新たな計測の手法を考える．

4.2.3　　グリッド化

正規化のように座標を揃えてしまうのではなく，座標の値を活かした検出を行いたいが，前処理を行っていないそのままの値を入力に使うだけでは，正常な検出は行えない．座標自体の意味は生まれるが，ニューラルネットワーク内で行われている計算は単純な掛け算と足し算なので，座標を入力するだけでは，動作ごとにあまり差が生まれないのである．これを改善するため，座標のグリッド化を行う．

グリッド化は，座標を各軸一定区間ごとに区切り，格子状にマスを作って座標を分類する(図 11). 値としては，マスを配列に対応させ，動作が通ったマスの値が+1，通らなかったマスの値を-1 とし，その配列の値をインデックスの順にニューラルネットワークへ渡す．これによって，座標自体が意味を持つとともに，座標をマスごとに分類することによって動作ごとの差もはっきり生まれると考えられる．一方，動作の中での座標の順序が情報として残らないことや，近い座標が連続した際に同じマスに記録されるため，情報が減少してしまうことが懸念される．グリッド化は，各軸 20 区間とし，正負それぞれ 10 区間となるように区切る．

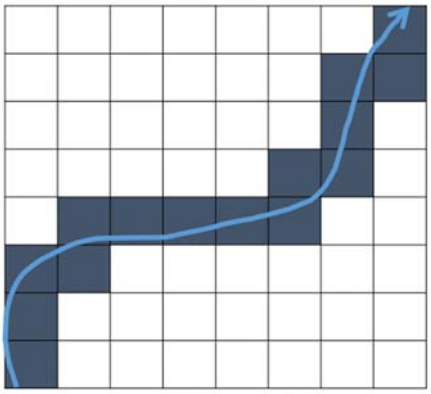

図 11　グリッド化

5. 評価 1

5.1　実験内容

4.2.3 の通りにグリッド化を前処理として導入して実際に検出を行い，従来の前処理の場合と比較する．被験者はあまり VR に慣れていない学生３名で，それぞれが任意の動作を 3 つ登録し，その後に 15 回の検出を意図した動作を行う．従来通り，評価値が 0.9 を超えた場合を検出とする．登録する動作の長さは 1 秒以内とする．登録された動作全体の長さの平均を求め，その長さだけ動作を記録し続け，検出処理に使用する．フレーム毎に検出処理を行っており，一つの動作中に何度も検出が記録されてしまうため，一度検出を記録したらその動作は 1 秒間検出を行わないようにし，被験者にも 1 秒以上間隔を空けて動作を繰り返すよう指示する．

5.2　実験結果

表 1 が実験の結果で，正常な検出，未検出，誤検出 の数を示している．

表 1　結果 1

	被験者 1			被験者 2			被験者 3			平均		
	正	未	誤	正	未	誤	正	未	誤	正	未	誤
従来	15	0	6	15	0	4	15	0	3	15	0	4.3
グリッド	10	5	3	13	2	3	15	0	2	12.7	2.3	2.6

どの被験者の結果を見ても，従来の手法に比べて誤検出が低下したことが見て取れるが，同時に未検出の数が増加している．操作者が動作を登録した際の座標を正確に再現することは不可能であり，そのズレが未検出に繋がっていると思われる．そのため，誤検出を増加させずに，検出を容易化する方法を考える．

6. 検出方法の改善

6.1　マスの増加

現状のグリッド化の手法では，検出を意図した動作をする際，記録された際と同じマスを通らなければ評価値が下がってしまう．そこで，動作の記録の際に，通ったマスを含む周囲 27 (3^3) マスも+1 としてしまうこと（図 12）で，検出を意図した動作の際に，-1 ではなく+1 のマスを通る可能性が高くなり，検出しやすくなると考えた．この手法の場合，単純に+1 のマスが増えることになるため，誤検出が増加してしまうことも予想される．

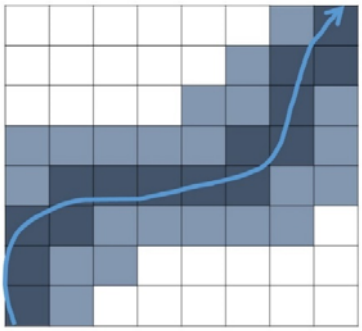

図 12　マスの増加

6.2　マスの拡大

6.1 と同じく，+1 のマスを通る可能性を高くしなければならないので，+1 のマスを増やすのではなく，マス自体を大きくする方法（図 13）を考える．座標を 20 区間ではなく 10 区間で区切ることとすると，3 次元であるためマス自体が占める体積は 20 区間の場合に比べて2^3の 8 倍となり，これも評価値が上がりやすくなり，検出しやすくなるが，同時に誤検出も増加することが予想される．

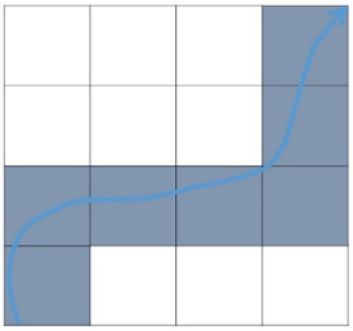

図 13　マスの拡大

7. 評価 2

7.1　内容

評価 1 と同じように，6.1 と 6.2 の前処理を導入して実際に検出を行い，これまでの結果と比較する．それ以外の条件は変わらない．

7.2　結果

結果は表 2 のようになった．正常な検出，　未検出，誤検出の数を示している．上二つの手法の結果は 5.2 のものである．図 14 は結果を離散グラフにしたものであり，横軸が正常な検出の数，縦軸が誤検出の数である．

予想通り，正常な検出が増加するとともに，誤検出も増加している．新たな二つの手法を比べると，+1 のマスを増やす手法の方が，未検出，誤検出の発生をともに抑えられていることが見て取れる．

グラフにして見ると，マスを拡大した場合の誤検出の多さが目立つ．正常な検出の増加と誤検出の抑制がトレードオフのような関係になっている．

表２　結果２

	被験者１			被験者２			被験者３			平均		
	正	未	誤	正	未	誤	正	未	誤	正	未	誤
従来	15	0	6	15	0	4	15	0	3	15	0	4.3
グリッド	10	5	3	13	2	3	15	0	2	12.7	2.3	2.6
+1 増加	13	2	2	13	2	5	15	0	2	13.7	1.3	3.0
マス拡大	13	2	6	12	3	6	15	0	2	13.3	1.7	4.7

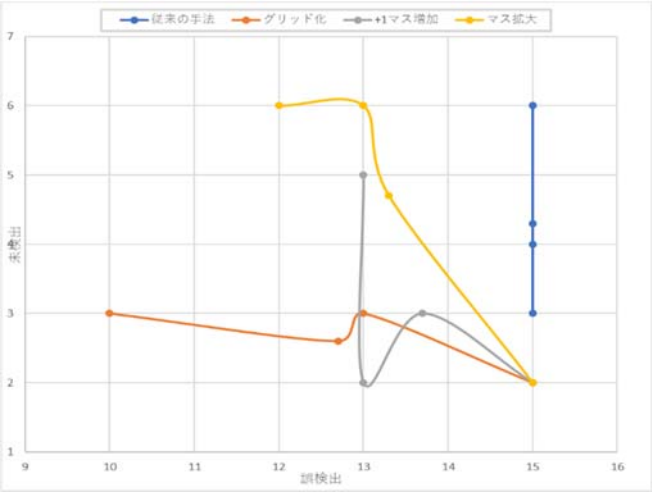

図 14　離散グラフ

8. 結言

本論文では，VR 空間での没入感を向上させるために，自動的な動作の検出を用いた操作方法を検討し，それを実現するための新たな手法の実装を行った．また，その

手法を用いた検出方法が従来の手法を用いた場合に比べてどのような結果となるのかを比較した．

誤った検出を抑えるために新たな手法を提案したが，その手法では正常な検出が行われないケースが増え，単純に改善したとは言えない結果となった．しかし，自動的な動作の検出を操作方法として導入するという点では大きく進歩しており，現実味を帯びてきていると言える．

今後の課題として，区分の数やマスに対する評価値の最適な値を求めることが挙げられる．特にマスに対する評価値の点は，今回は3^3の周囲 27 マスとしたが，上下左右奥手前の周囲 6 マスのみの場合など，様々なケースで実験することや，計測の際に通ったマスのみを＋1 とし，周囲のマスは＋0.5 とするなど，実際に通ったマスとその周囲のマスで評価値に差を付けると，誤検出の点がさらに改善できるのではないかと考える．

グリッド化して，通ったマスに印を付けるような手法であるため，動作の向きの情報が失われており，例えばコントローラを上向きに振るような動作と下向きに振るような動作を区別することができない．これも誤検出の一つの要因となっている．

現在の手法では，マスを大きくするか，正の評価値を持つマスを増やすことで検出率が上がっているが，同時に誤検出も増加しており，一種のトレードオフの関係のようになっている．このため，操作者がマスの大きさ，評価値を変更できるようにすれば，個人の動作の傾向に適した評価の仕方に設定できる．また，これをトレードオフではなく，検出率の向上と誤検出の抑制のどちらも改善できる手法を考えることもこれからの課題である．

謝辞

本研究に際して，様々な点でご指導をして頂きました西村講師，この研究の機会をくださった大分高専情報工学科の先生方，そして多くの知識やご指摘を下さいました西村研究室の先輩・同期の皆様に厚くお礼申し上げます．

参考文献

[1] Francisco Torres GuerreroJorge Lozoya-SantosEduardo Gonzalez MendivilLeticia Neira-TovarPablo G. Ramírez FloresJorge Martin-GutierrezSmart : “Technology: First International Conference -Virtual Riality for Social Phobia Treatment”（2018），p.165-177

[2] Cuteness Technology，Edowonia : “VR Infinite gesture” (2018) at http://edwonstudio.com/vr-gesture [アクセス日: 30 5 2019]

[3] 小高 知宏 : 基礎から学ぶ　人工知能の教科書 (2019) ，株式会社 オーム社，p100-101

[4] 熊沢 逸夫 : 学習とニューラルネットワーク (電子情報通信工学シリーズ) (1998)，東光工業大学，p198-189

[5] HTC : “VR READY PC” (2020) at http://vive.com/jp/ready/ [アクセス日: 1 1 2020]

[6] HTC : “HTC VIVE” (2016) at https://www.vive.com/jp/product/ [アクセス日: 30 5 2019]

[7] Over the Edge Entertainment　:　“Unity 3d” at https://unity.com/　[アクセス日: 30 5 2019]

付録

付録１　正規化を行う関数

```
public static List<Vector3> DownScaleLine(List<Vector3> capturedLine)
		{
			float minX, maxX, minY, maxY, minZ, maxZ;
			Vector3 firstPoint = capturedLine[0];
			minX = maxX = firstPoint.x;
			minY = maxY = firstPoint.y;
			minZ = maxZ = firstPoint.z;

			foreach (Vector3 point in capturedLine)
			{
				minX = getMin(minX, point.x);
				maxX = getMax(maxX, point.x);

				minY = getMin(minY, point.y);
				maxY = getMax(maxY, point.y);

				minZ = getMin(minZ, point.z);
				maxZ = getMax(maxZ, point.z);
			}
			float distX = Mathf.Abs(maxX - minX);
			float distY = Mathf.Abs(maxY - minY);
			float distZ = Mathf.Abs(maxZ - minZ);
			float axisMax = distX;
			axisMax = getMax(axisMax, distY);
			axisMax = getMax(axisMax, distZ);
			Matrix4x4 translate = Matrix4x4.identity;
			translate[0, 3] = -minX;
			translate[1, 3] = -minY;
			translate[2, 3] = -minZ;

			Matrix4x4 scale = Matrix4x4.identity;
			scale[0, 0] = 1 / axisMax;
			scale[1, 1] = 1 / axisMax;
			scale[2, 2] = 1 / axisMax;
			List<Vector3> localizedLine = new List<Vector3>();
			foreach (Vector3 point in capturedLine)
			{
				Vector3 newPoint = translate.MultiplyPoint3x4(point);
				newPoint = scale.MultiplyPoint3x4(newPoint);
				localizedLine.Add(newPoint);
			}
			return localizedLine;
		}
```

付録２　グリッド化と評価値の割り振りを行う関数

```
public static List<double> GridSplit(List<Vector3> list)
        {
            int len = 20;
            int[,,] grid = new int[len * 2, len * 2, len];
            for (int i = 0; i < len * 2; i++)
            {
                for (int j = 0; j < len * 2; j++)
                {
                    for (int k = 0; k < len; k++)
                    {
                        grid[i, j, k] = 0;
                    }
                }
            }
            foreach (Vector3 point in list)
            {
                if (point.z < 0) Back("mainasu");
                int ix = (int)Math.Floor((point.x + 1f) * len);
                if (ix == len) ix = len - 1;
                int iy = (int)Math.Floor((point.y + 1f) * len);
                if (iy == len) iy = len - 1;
                int iz = (int)Math.Floor(point.z * len);
                if (iz == len) iz = len - 1;
                int i, j, k;
                i = -1; j = 0; k = 0;
                if ((ix + i) >= 0 && (ix + i) < len * 2 && (iy + j) > 0 && (iy + k)
< len * 2 && (iz + k) > 0 && (iz + k) < len) grid[ix + i, iy + j, iz + k] = 1;
                i = 1;
                if ((ix + i) >= 0 && (ix + i) < len * 2 && (iy + j) > 0 && (iy + k)
< len * 2 && (iz + k) > 0 && (iz + k) < len) grid[ix + i, iy + j, iz + k] = 1;
                i = 0; j = 1;
                if ((ix + i) >= 0 && (ix + i) < len * 2 && (iy + j) > 0 && (iy + k)
< len * 2 && (iz + k) > 0 && (iz + k) < len) grid[ix + i, iy + j, iz + k] = 1;
                j = -1;
                if ((ix + i) >= 0 && (ix + i) < len * 2 && (iy + j) > 0 && (iy + k)
< len * 2 && (iz + k) > 0 && (iz + k) < len) grid[ix + i, iy + j, iz + k] = 1;
                j = 0; k = 1;
                if ((ix + i) >= 0 && (ix + i) < len * 2 && (iy + j) > 0 && (iy + k)
< len * 2 && (iz + k) > 0 && (iz + k) < len) grid[ix + i, iy + j, iz + k] = 1;
                k = -1;
                if ((ix + i) >= 0 && (ix + i) < len * 2 && (iy + j) > 0 && (iy + k)
< len * 2 && (iz + k) > 0 && (iz + k) < len) grid[ix + i, iy + j, iz + k] = 1;

            }

            List<double> r = new List<double>();
            foreach (double g in grid)
            {
                r.Add(g);
            }
            return r;
        }
```