バーチャルリアリティ環境における
視線の空間および過渡分布の数理解析に関する研究

令和２年度

修士論文

指導教員　ルジェロ•ミケレット教授

横浜市立大学大学院

生命ナノシステム科学研究科

物質システム科学専攻

195224

平井　亮

目次

バーチャルリアリティ環境における
視線の空間および過渡分布の数理解析に関する研究

平井　亮

主指導教員　ルジェロ•ミケレット教授

副指導教員　　橘　勝　　教授

副指導教員　　篠崎　一英　　教授

バーチャルリアリティ環境における
視線の空間および過渡分布の数理解析に関する研究

物質システム科学専攻　　平井亮
指導教員　　ルジェロ•ミケレット教授

【重要語句】
バーチャル•リアリティー（Virtual Reality）：現物・実物（オリジナル）ではないが機能としての本質は同じであるような環境を、ユーザーの五感を含む感覚を刺激することにより理工学的に作り出す技術およびその体系。VR。
視覚探索（Visual Search）：外界の複雑な視界情報に対し、認知・行動に必要となる情報のみに認知資源を割り当てる視覚的注意のメカニズム。
ニューラルネットワーク（Neural Network）：人間の脳の神経回路を模擬した情報処理システムの総称、複雑な特徴パターンの認識能力に優れる。

【研究の背景と目的】

近年、バーチャルリアリティ（Virtual Reality: VR）技術の普及と発展が急速に進んでいる。それに伴い、色々なタイプのヘッドマウントディスプレイ（Head Mount Display: HMD）が提案されており、VR 関連の市場規模も年々拡大してきている。

しかしながら、以下に示すように、現状では未解決の課題は少なくない。

（1）　解像度の粗さ解決
（2）　VR 酔いの解消
（3）　自然な操作を可能にするインターフェイス実現
（4）　リアル世界の高速・高度モデリング技術の確立
（5）　利用者の高速・高度モデリング技術の確立
（6）　HMD、VR グラスを装着した人の顔が分からない問題の解決
（7）　VR 空間内での画像・音源などの連動

一方で、視線トラッキング（Eye Gaze-Tracking）技術の開発も進んできている。これにより、ユーザがどこを注視しているのかを把握することができ、注視情報を応用したインタラクション技術の開発が可能となってきている。

本研究ではこのような背景を鑑み、VR 環境下での視線トラッキングデータの解析を試みることとした。

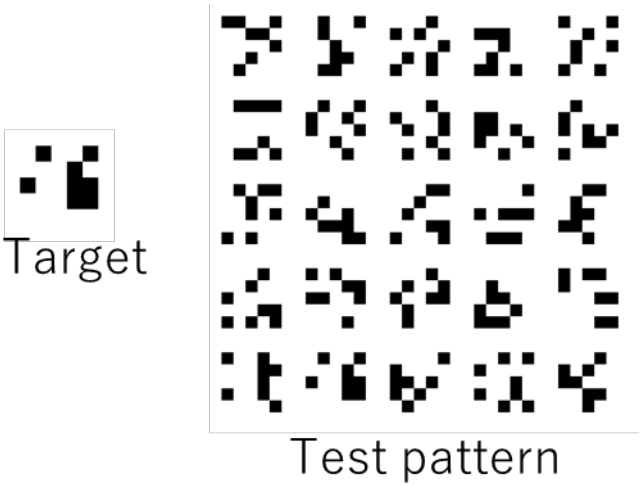
本研究では、FOVE0 VR システムを用いて、VR 環境下における視線トラッキングデータを取得し、視線の動きを数理的に解析し、その特徴づけを行なうことを目的とする。そのために数理モデルを提案し、「凝視」「探索」などの視線の空間的動きと過渡的動きについて自動解析し、特徴づけることを試みた。さらに、近年注目されている機械学習の手法を導入し、「VR 酔い」に関連する視線

の動きを解析・分類することも併せて試みた。

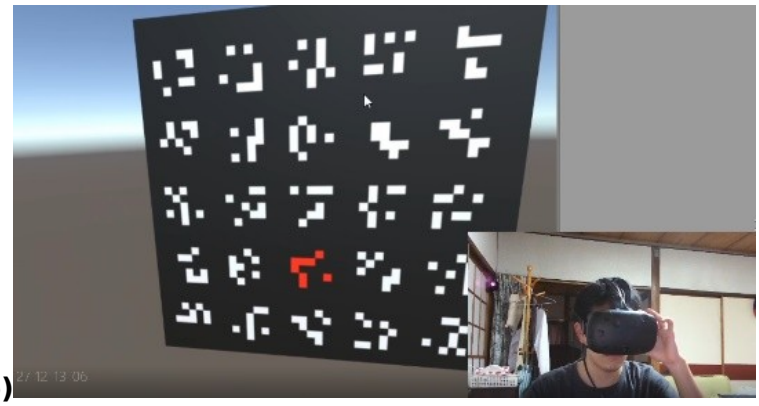
【方法】

視覚探索とゲーム形式テストのそれぞれの実験において頭の向き、頭の位置、キャラクターの位置、キャラクターの位置、右目左目それぞれの視線の向きをリアルタイムにデータファイルとして記録する。

視覚探索では、被験者は、ターゲットとなるパターンを覚え、与えられたテストパターンの中から、ターゲットを探す。この一連の視線の動きを自動的に取得する（図 1.）。



(a)



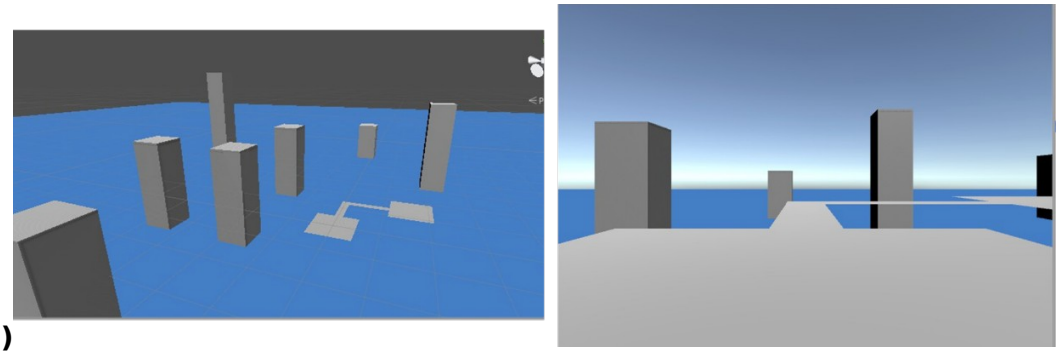
(b)

図 1.　(a)ターゲットとテストパターンの例　(b)視覚探索実験の様子

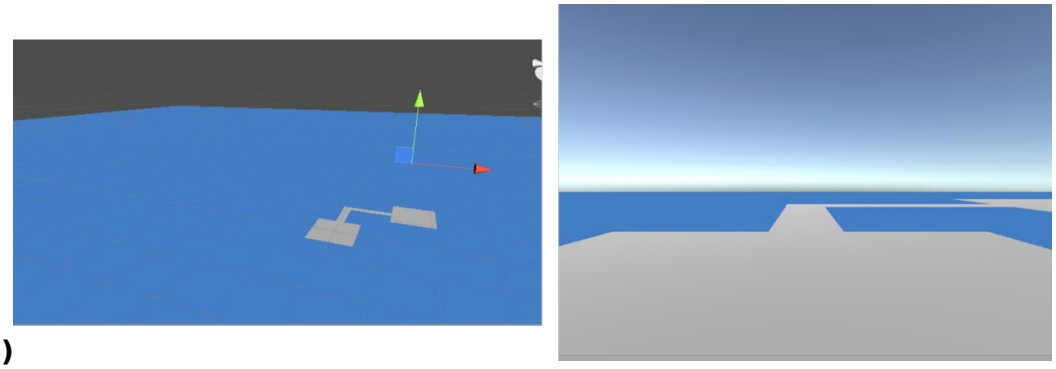
さらに、K-Means 法、Elbow 法などを用いて、その結果を自動クラスタ化して評価する。

視野変動状態でのゲーム形式実験（視線の過渡分析）においては、VR 酔いの前兆現象をつかもうと考え、視野が変動する中で細い通路を歩行する、ゲーム形式の実験を試みた。

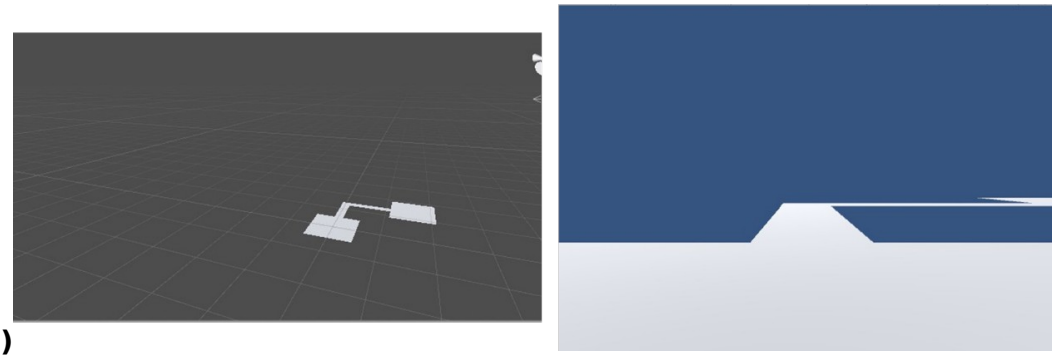
Stage 1-4 の 4 タイプのコンテンツを用意した。それぞれ、視野変動の有り無し、背景（水平線・構造物の有り無し）の違いがある（図 2.）。



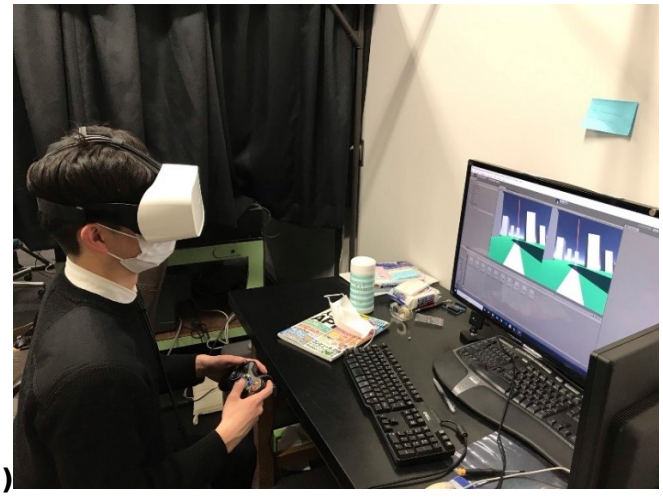
(a)



(b)



(c)



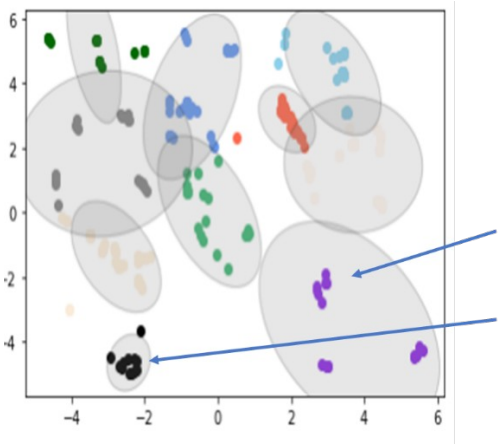
(d)

図 2.　(a)Stage-1 および-2　(b)Stage-3　(c)Stage-4　(d)実験の様子

このゲーム形式実験で得た視線データを、Fourier 変換ののち、Neural Network 技術を用いた解析を試みた。

【結果と考察】

クラスタリング後のデータについて、クラスタごとに標準偏差 σ とこれを基にした確率楕円を計算する。長径を $2\sigma_A$、短径を $2\sigma_B$ と定義し、確率楕円を表示したものが図 3.である。Target 位置の確率楕円は、他の位置の確率楕円に比較して明確に小さくなっており、Target を「凝視」する様子が見て取れる。



	中心座標	$2\sigma_A$	$2\sigma_B$
Searching	(3.59, -3.56)	7.01	4.26
Stare	(-2.43, -4.65)	1.78	1.15

図 3.　確率楕円の計算結果

ゲーム形式テストにおいて、被験者に「酔い」を感じたかを答えてもらった。被験者ごとの酔いを感じる度合い（5 段階評価）を示す。この結果、Stage-2、Stage-3 は、やや酔いやすいコンテンツである、という結果であった。

Neural Network 技術による解析結果において、Stage-2、Stage-3 は正答率が高かった事実と符合する。この点は大変興味深く、Neural Network 技術が、被験者に依存しない、VR 酔いに関連した共通の視線の動きを捉えたと考えることも可能であろう（図 4.）。

Examinee T, O, F and M

		Correction rate
Stg-1	[[555. 1022. 680. 1743.]	13.9%
Stg-2	[506. 1428. 1071. 995.]	35.7%
Stg-3	[580. 925. 1549. 946.]	38.7%
Stg-4	[437. 1388. 770. 1405.]]	35.1%
	accuracy: 30.9%	

Response of Examinees (Feeling of VR Sickness by Level 0-5)

Examinee	H	T	O	F	M	Ave.
Stg-1	0	0	0	0	0	0.0
Stg-2	3	1	2	2	2	2.0
Stg-3	2	2	3	2	1	2.0
Stg-4	1	2	1	2	1	1.4

図 4.　Neural Network 解析とアンケートの結果

第１章　序論

1.1.　研究の背景

近年、仮想現実（Virtual Reality: VR）技術の普及と発展が急速に進んでいる。それに伴い、色々なタイプのヘッドマウントディスプレイ（Head Mount Display: HMD）が提案されており[1]-[3]、VR 関連の市場規模も年々拡大してきている（図 1-1-1.）[4]。

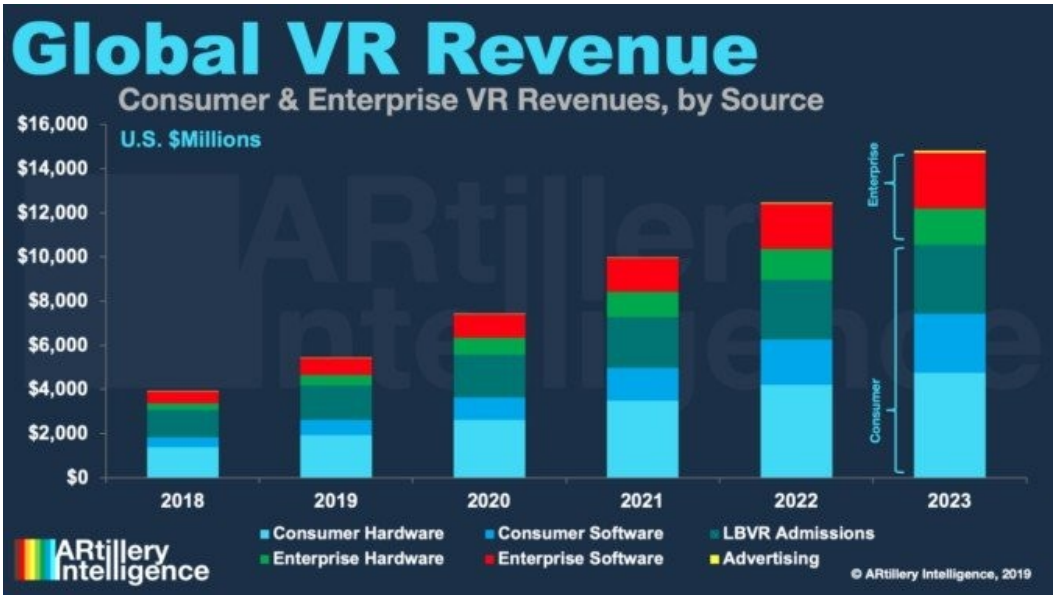


図 1-1-1.　世界の VR 関連市場規模

VR 技術を用いた、新たなビジネスも提案されている（図 1-1-2.）。

図 1-1-2.　今年 2020 年に提案されたバーチャルマーケット

さらに、今年は新型肺炎拡大の影響を受けて、「3密」回避のために、バーチャル・タウン開発の試みもなされた（図 1-1-3.）

図 1-1-3.　バーチャル渋谷

しかしながら、以下に示すように、現状では未解決の課題は少なくない。

（1）　解像度の粗さ解決
（2）　VR 酔いの解消
（3）　自然な操作を可能にするインターフェイス実現
（4）　リアル世界の高速・高度モデリング技術の確立
（5）　利用者の高速・高度モデリング技術の確立
（6）　HMD、VR グラスを装着した人の顔が分からない問題の解決
（7）　VR 空間内での画像・音源などの連動

これらの課題は、いずれも容易に解決できるものではないが、（2）（3）については、ユーザの視線の動きを解析することによって、解決へのきっかけが得られる可能性がある。ユーザがどこを注視しているのか、あるいは、何かを探索して迷っているのか、さらにはランダムウォークのような動きがあるのかを解析することによって、VR 酔いの前兆現象を捉えたり、視線による選択や入力の実現につなげられたりする可能性がある。

一方で、視線トラッキング（Eye Gaze-Tracking）技術の開発も進んできている。これにより、ユーザがどこを注視しているのかを把握することができ、注視情報を応用したインタラクション技術の開発が可能となってきている[5]。従来の研究では、PC のスクリーンを対象とした視線追跡が主

であったが、今では視線トラッカーが搭載された Head Mounted Display （HMD）もリリースされている。例えば、「StarVR one[6]」「Tobii Pro VR[7]」「FOVE 0[8]」がそれである。HMD にアイトラッカーが搭載されることで、視野の中心付近だけを高解像度で見せる「フォービエイテッド・レンダリング」が可能になるなど、HMD においても視線を利用する技術や研究が報告されている[9]。

本研究ではこのような背景を鑑み、VR 環境下での視線トラッキングデータの解析を試みることとした。

1.2. 研究の目的

本研究では、FOVE0 VR システム（図 1-2-1.）を用いて、VR 環境下における視線トラッキングデータを取得し、視線の動きを数理的に解析し、その特徴づけを行なうことを目的とする。そのために数理モデルを提案し、「凝視」「探索」などの視線の空間的動きと過渡的動きについて自動解析し、特徴づけることを試みた。さらに、近年注目されている機械学習の手法を導入し、「VR 酔い」に関連する視線の動きを解析・分類することも併せて試みた。

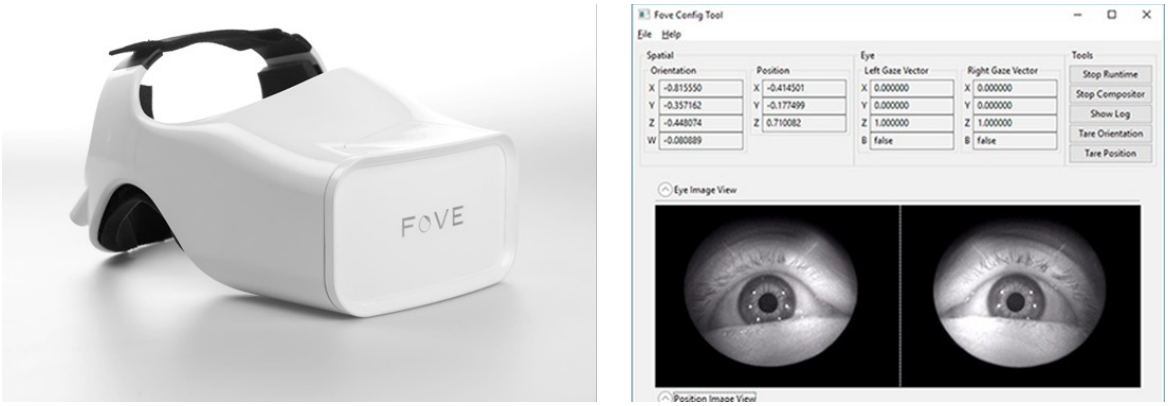


図 1-2-1. FOVE0 と視線トラッキングシステム

1.3. バーチャルリアリティ（VR）について

バーチャルリアリティ（VR）とは、一般に現物・実物（オリジナル）ではないが機能としての本質は同じであるような環境を、ユーザの五感を含む感覚を刺激することにより理工学的に作り出す技術およびその体系のことを言い、日本語では「仮想現実」などと訳される。

近年では、ゲーム、シミュレーション、トレーニング、アトラクションなどに、主に視覚刺激による VR 環境が提供されてきている。

本研究では、HMD によって VR 環境を実現しているが、主に両眼に異なる画像を入力することによって立体感を提供し、さらに頭の動きを検知する加速度センサを用いて入力する画像を制御することによって、より現実に近い視覚的刺激を提供している。

一方で、音源が立体画像の変化に応じて変化するような、精密な制御はできていない。

言うまでもないが、加速度、温度、圧力（風圧など）や匂いは、被験者に対して全く提供されない。

このように、本研究では主に視覚に限定したバーチャルリアリティ環境における、視線トラッキングデータを扱うことになる。

第２章　先行研究と本研究の比較

2.1.　先行研究事例

広島市立大学　黒田ら[10]は、大学における講義の理解度によって受講生の凝視点の分布がどのように異なるか、分析を試みている（図 2-1-1、2-1-2）。

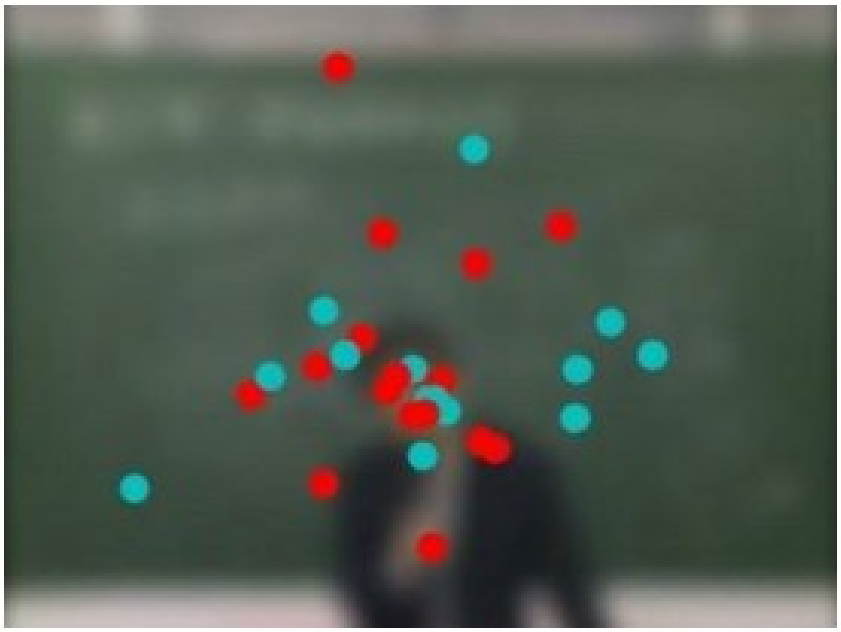


図 2-1-1.　講義中の凝視点分布の例[10]

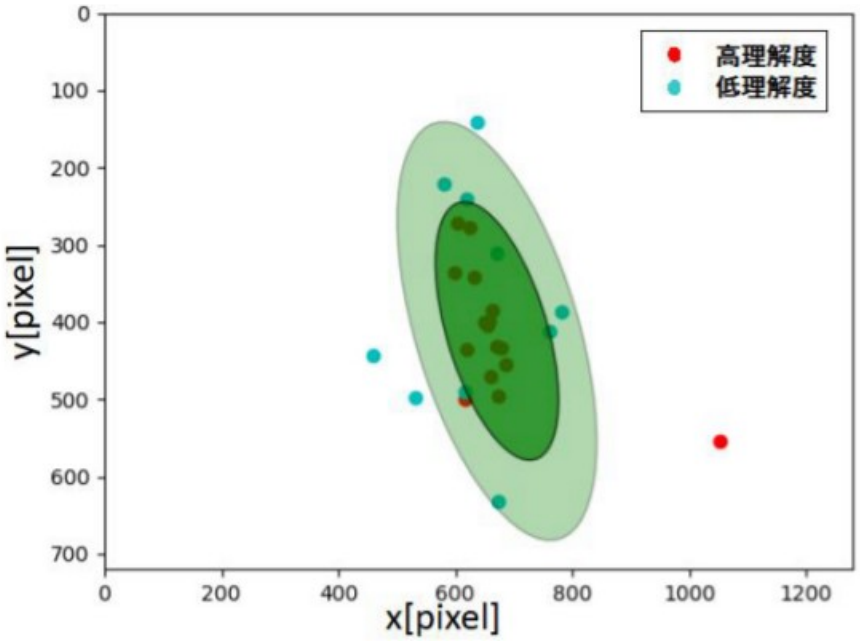


図 2-1-2.　凝視点の正規分布解析の例[10]

黒田らは、FOVE0 視線トラッキングシステムを用いて視線の空間分布データを取得し、凝視点群を正規分布と仮定している。高理解度の注視点群のほうが、低理解度の注視点群より正規分布楕円の面積が小さく、視点の散らばりが小さいことを示した。

一方、Tobii Technology（スウェーデン）は、視線トラッキングシステムと視線分布の解析ソフトウエア（Tobii Pro）を提供している[11]。Tobii Pro は、実像の上に注視点群をプロットし（図 2-1-3.）、脳波などの生体情報との関連性を取得する（図 2-1-4.）ことに主眼が置かれている。

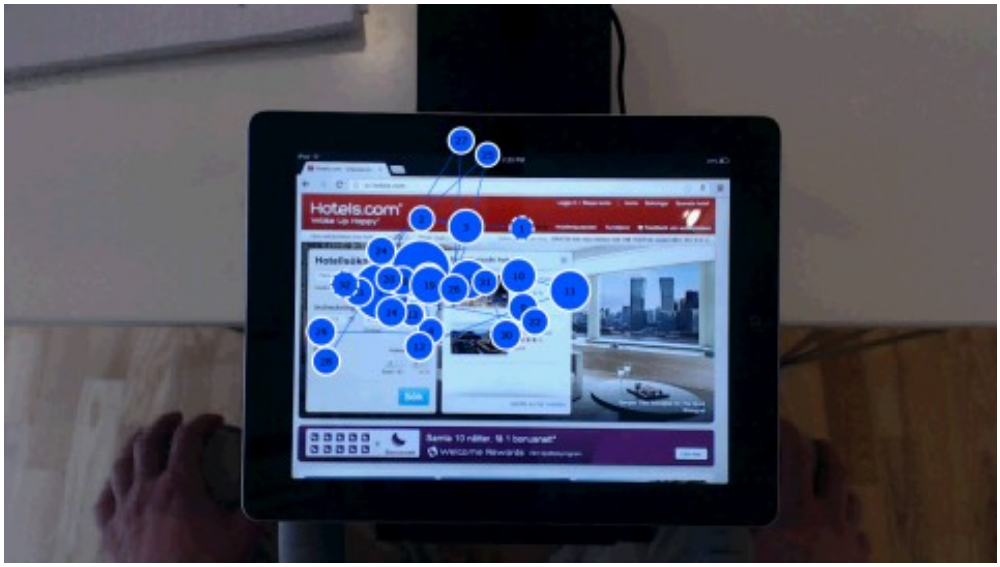


図 2-1-3.　実像と凝視点群の重ね合わせの例[11]

図 2-1-4.　凝視点と脳波の関連性の取得を試みる例[11]

また、Australian National University（ANU）の K. Naqshbandi らは[12]、被験者が絵画

を見たとき、視線分布を K-Means 法によってクラスタリングする試みを行なっている。この研究によって、人の顔周辺に視線が集中する傾向があることを示している（図 2-1-5.）。

図 2-1-5.　K-Means 法による視線のクラスタリングを試みる例[12]

2.2.　先行研究における課題

広島市立大の黒田らの先行例では、視点の空間分布について正規分布楕円を用いて表現しており、さらに時間経過とともに視点の乖離度を評価する試みを行なっている。しかし、複数の注視点群を分類する試みや、正規分布楕円を定量的に定義することは行なわれていない。このため、定量的に凝視点の密集度を評価することはできていない。また、視点の過渡的な振る舞いについても、定量的な解析は行なわれていない。

Tobii Pro では、視点の動きを実画面上に表示したり、他の生体情報と関係づけたりすることに重点が置かれており、注視点の数理的解析に関しては限定的である。

ANU の K. Naqshbandi らの先行例では、K-Means 法による視線のクラスタリングを試みた点において優れているが、「凝視」「探索」などの視線の動きを分類しておらず、また視線密度などの定量的評価も行なっていない。

このように、先行研究事例において、視線データの数理的な解析は充分に行なわれておらず、「凝視」や「探索」「VR 酔い」と視線データがどのように関連しているか、定量的・数理的に特徴づける試みは行なわれていない。

2.3. 先行研究と本研究の相違点

本研究では、先行研究では行われていない、「凝視」や「探索」「VR 酔い」と視線データがどのように関連しているか、数理的に特徴づける試みを行なう。

このため、複数の凝視点群（クラスタ）の数を「Elbow 法」によって確定し、「K-Means 法」を用いて「自動クラスタリング」する手法を提案した。さらに、各クラスタを「確率楕円」を用いて表現し、確率楕円と注視点数から「視点密度」を計算する方法を考案した。これらの手法によって、凝視点群（クラスタ）を自動解析し、視点の空間分布を定量的に評価することができるようになった。

さらに、視点の過渡的振る舞いを評価するため、注視点群についてフーリエ変換を行ない、周波数（時定数）成分を抽出する解析法を提案した。この手法によって、視点の過渡的な振る舞いを定量的に評価することができる。

このように、視点データ・凝視点群（クラスタ）を数理的に解析することによって、定量的に空間的・過渡的に解析しようとする点が、先行研究と大きく異なる点であり、本研究の特徴となっている。

第３章　実験概要

3.1. Unity3D について

本研究で被験者が体験する3次元空間は Unity3D ゲームプラットフォームによって作られる。Unity3D は主にコンピュータや携帯機器向けのゲームを開発するためのプラットフォームで、プログラムは C 言語で書かれる。

スクリプト言語としては、C#、JavaScript、Boo の 3 種類のプログラミング言語によって制御することができるが、本研究の実験環境では C#を使用している。Unity3D プラットフォームを実験に使用した理由は、その空間づくりの手軽さにある。

本研究では、異なるパターンの 3 次元空間を複数用意する必要があり、さらにその空間情報を分析するためにデータを記録する必要がある。Unity3D では、プログラムにより複数のタスクを簡易的に行えるほか、HMD の動きも同時に記録することができるという利点がある。

3.2. 実験環境

3.2.1. 使用機材

FOVE0；FOVE 0 は世界初の視線追跡（視線トラッキング）機能搭載型の VR ヘッドセットである（図 1-2-1.）。視線追跡の精度の高さと柔軟なデータ出力機能から、世界中の企業や研究機関で使われている。

Unity 3D；FOVE 0 の実験を作る際に使用したソフトウェアである。3D オブジェクトの作成や入力、描画、当たり判定の計算などの様々なライブラリを含む便利な 3D ゲームの統合開発環境である。

3.2.2. プログラミング言語

C#；Windows の開発したオブジェクト指向のプログラミング言語である。Unity は主に Windows 環境で動くので Unity のオブジェクトやそのコンポーネントの処理に用いられる。

Python；数理解析に用いたオブジェクト指向言語である。機械学習などのライブラリを多く含むだけでなく、Jupyter Notebook などの表現や編集が便利で使いやすい実行環境が存在する。

3.3. 実験方法

視覚探索とゲーム形式テストのそれぞれの実験において頭の向き、頭の位置、キャラクターの位置、キャラクターの位置、右目左目それぞれの視線の向きをリアルタイムにデータファイルとして記録する。

図 3-3.　FOVE0 と視線トラッキングシステムを用いた実験の様子

3.3.1. 視覚探索

被験者は、ターゲットとなるパターンを覚え、与えられたテストパターンの中から、ターゲットを探す。この一連の視線の動きを自動的に取得する（図 3-3-1.）。

実験の手順は以下のとおりである。

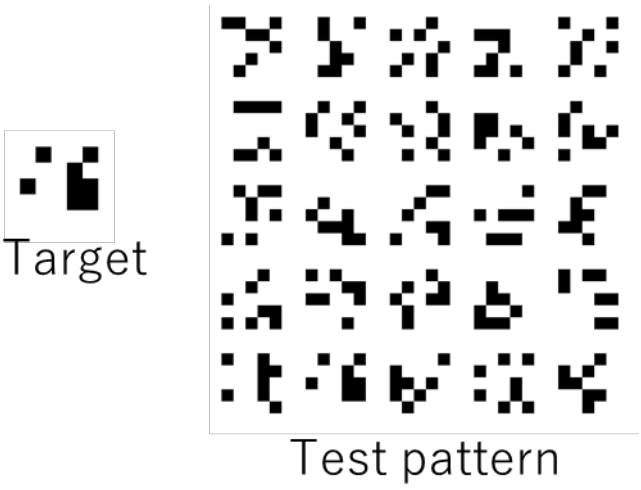
a. VR ヘッドセットを装着する

b. 表示された画像を覚えたら、Space Key を押す。

c. 覚えた画像と同じ画像を選択し Space Key を押す。覚えた画像がない、もしくはわからないときは何も選択せずに Space Key を押す。

d. 2. 3.を 50 回繰り返す。

この結果、被験者 7 人に対し各 50 回分の視線の座標データを取得するとともに、ターゲット発見までの時間（Reaction Time）を記録することができた。



(a)

(b)

図 3-3-1.　(a)ターゲットとテストパターンの例　(b)視覚探索実験の様子

3.3.2. 視野変動状態でのゲーム形式実験（視線の過渡分析）

VR 酔いの前兆現象をつかもうと考え、視野が変動する中で細い通路を歩行する、ゲーム形式の実験を試みた。

Stage 1-4 の 4 タイプのコンテンツを用意した。それぞれ、視野変動の有り無し、背景（水平線・構造物の有り無し）の違いがある（図 3-3-2.）。

Stage-1　水平線あり・構造物あり　視野変動なし

Stage-2　水平線あり・構造物あり　視野変動あり

Stage-3　水平線あり・構造物なし　視野変動あり

Stage-4　水平線なし・構造物なし　視野変動あり

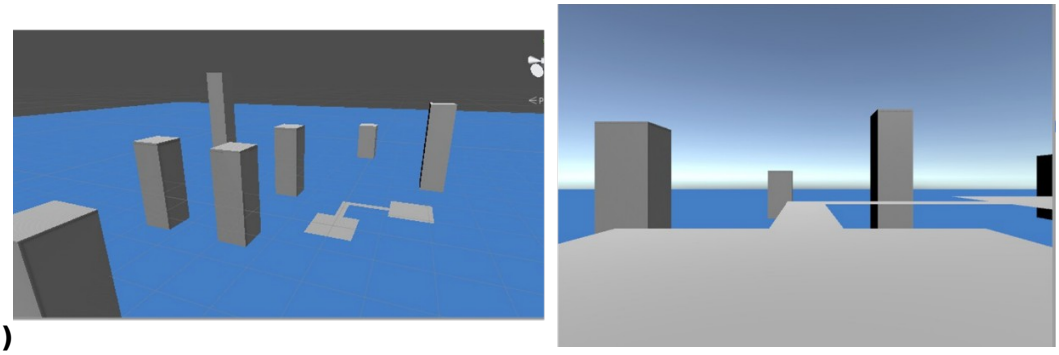
実験の手順は以下のとおりである。

a. VR ヘッドセットを装着する。

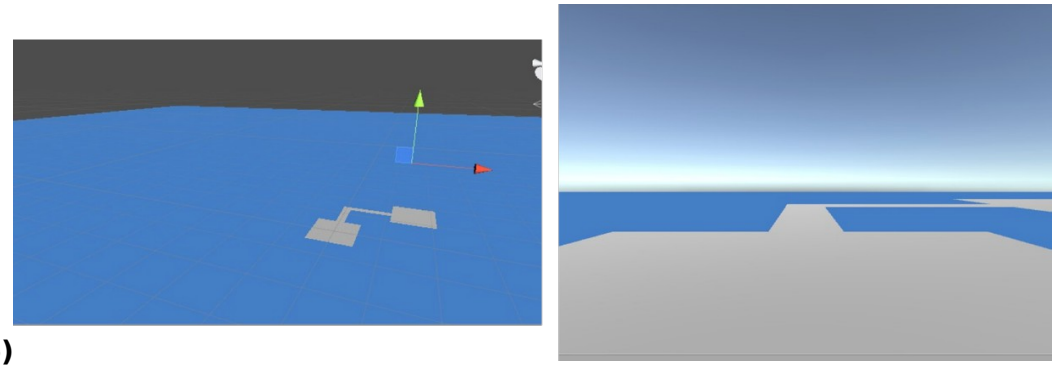
b. コントローラーを操作してゴールを目指す。

c. b.を Stage4 まで繰り返す。

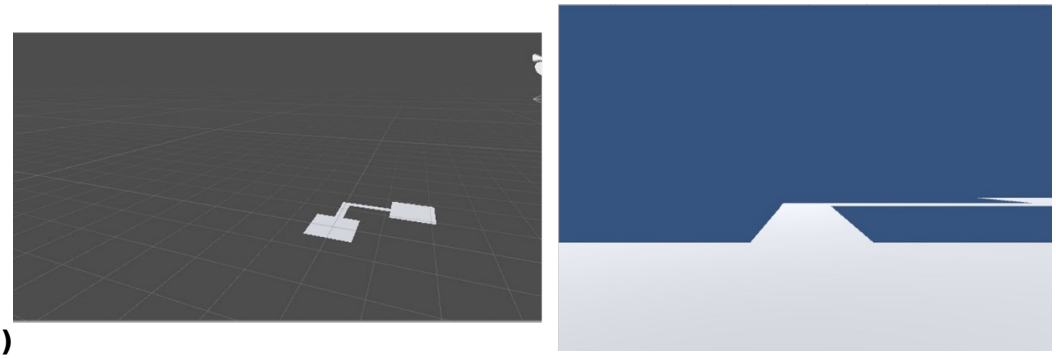
この結果、被験者 6 人に対し各1回分の視線の座標データを取得することができた。



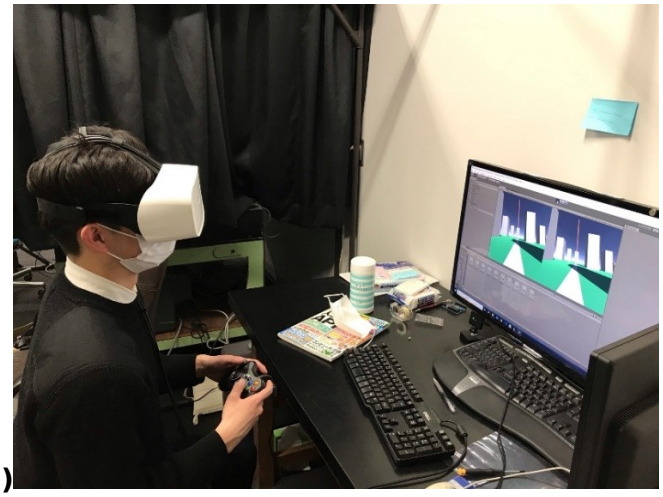
(a)



(b)



(c)



(d)

図 3-3-2.　(a)Stage-1 および-2　(b)Stage-3　(c)Stage-4　(d)実験の様子

3.4. 数理解析方法

3.4.1. Elbow 法

Elbow 法は、2 次元や 3 次元空間に分散したデータを、いくつかのデータ群（クラスタ）に分ける（クラスタリング）際、最適なクラスタ数を求める手法の1つである。同一のクラスタは、データ空間中において近くに集まる（＝歪（Distortion）が小さい）という性質を用いる。クラスタ数を少しずつ増やしてゆき、クラスタの歪み（Distortion）があまり改善しなくなる点を、クラスタ数の最適値とする（Elbow Point）。

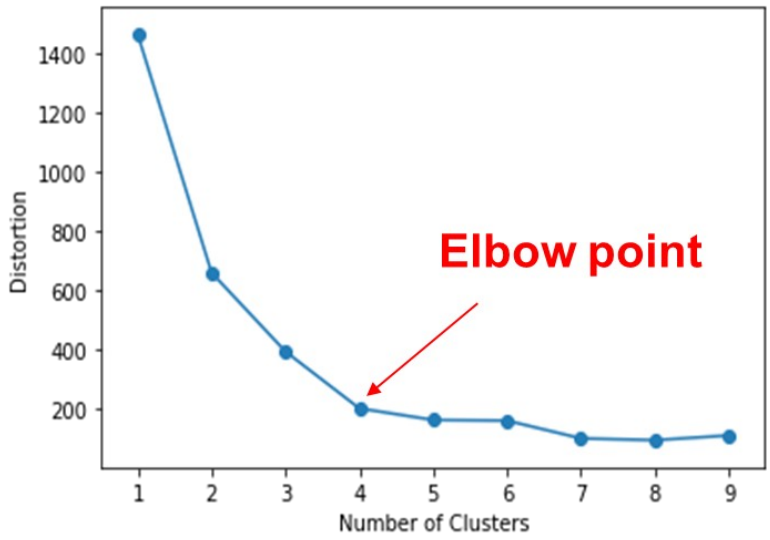


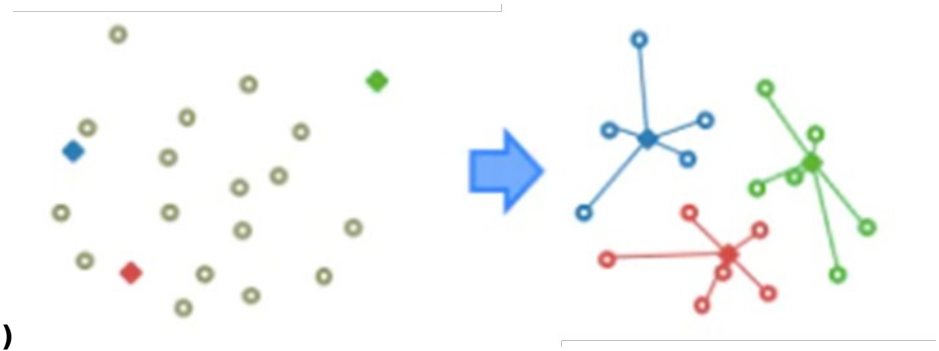
図 3-4-1. Elbow 法計算結果の例

図 3-4-1.は、Elbow 法計算結果の一例である。クラスタ数を増やしてゆくと、Distortion が徐々に減ってゆくが、クラスタ数 4 以上に増やしてもあまり改善しない。そこで、クラスタ数 4 を最適値とする。

本研究では、Python プログラムコードを作成して、視線の空間データについてクラスタ数を求めることにした。

3.4.2. K-Means 法

K-Means 法[13]は、あらかじめクラスタの数を決めておき、各データをクラスタに振り分けてゆく方法である。各クラスタに含まれる各データとそのクラスタの重心点との距離が、他のクラスタの重心点より小さくなるように振り分ける。結果的に、クラスタ間の距離は大きくクラスタ内のデータ間の距離は小さくなるように振り分けられる。図 3-4-2.は、K-Means 法によるクラスタリングのイメージと評価関数を示している。ここで、K はクラスタの数である。



(a)

$$\sum_{i=1}^{K} \sum_{x \in X_i} \left\| x - \mu_i \right\|^2$$

(b)

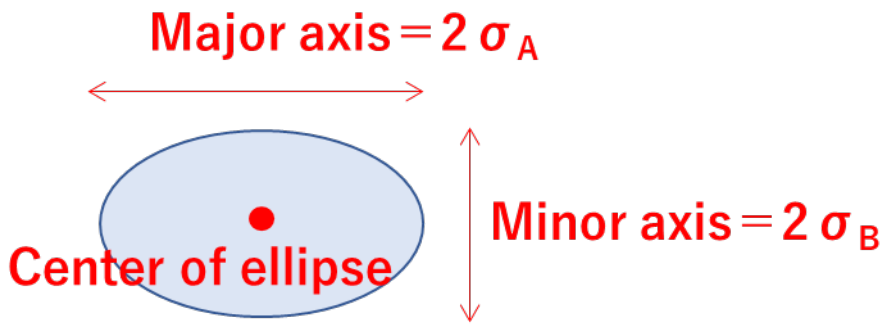
図 3-4-2.　（a）K-Means 法によるクラスタリングのイメージ　（b）評価関数

計算方法としては、クラスタへのデータ配分を評価するための評価関数を定め、その評価関数が最適条件を得るように、クラスタへの配分を決めることになる。より具体的には、K このクラスタに分割し、クラスタの重心を代表点として選び、代表点とデータ点の距離を計算し、評価関数が最小となるまで計算を繰り返すのである。

3.4.3.　確率楕円と視線密度

各クラスタを評価するために、「確率楕円」と「視線密度」を定義することにした。

1つのクラスタが含むデータ点の分布が正規分布すると仮定し、短径、長径をそれぞれ標準偏差 σ の2倍として確率楕円を定義する。さらに、このクラスタに含まれる視点の数を確率楕円の面積で割ったものを視線密度として定義する（図 3-4-3.）。



(a)

$$\rho = \frac{\text{Gaze Points}}{\pi\ \sigma_A \sigma_B}$$

(b)

図 3-4-3.　（a）確率楕円のイメージ　（b）視線密度

3.4.4.　Fourier 変換

一般に、独立変数の次元を逆数に移す。例えば下記に示す方程式のように、変換前の独立変数 t[s]が時間の次元を持つとき、返還後の独立変数 ω[Hz or 1/s]は周波数の次元を持つ。

$$F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-2\pi i \omega t} dt$$

視線データの場合は、k∈Nの時、t_k、x_kとして得られるから、任意の周波数ωの Fourier 変換は、以下のような方程式で表わされる。

$$F(\omega) = \frac{1}{N} \sum_{k}^{N} x_k e^{-2\pi i \omega}$$

さらに、下記の方程式のようにこの Fourier 変換の絶対値を計算する。

$$|F(\omega)| = \frac{1}{N} \sqrt{\left(\sum_{k}^{N} x_k \cos(-2\pi\omega) \right)^2 + \left(\sum_{k}^{N} x_k \sin(-2\pi\omega) \right)^2}$$

本研究では、視野変動状態でのゲーム形式実験で得た視線データに関して、過渡解析（どのような周波数成分を持つか）を行なう際にこれらを適用した。

3.4.5.　Neural Network 技術

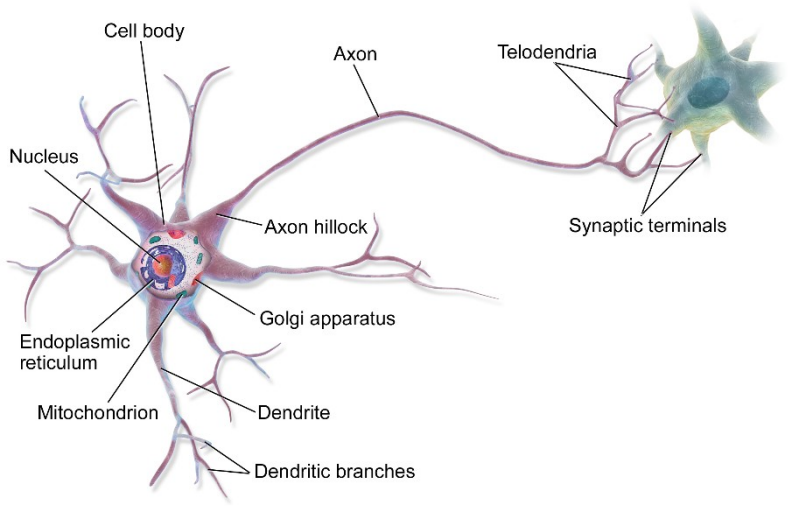
Neural Network 技術は、生物の神経回路網から着想されたもので（図 3-4-5.(a)）、本間ら[14]が音素の認識に適用し、その後文字や画像の認識に用いられるようになり、ディープラーニング分野の形成に至った。

一例として、3 層構造の Neural Network 技術について説明する。入力層（input）データを o_k、隠れ層（hidden）データを x_k、o_k、出力層データ（output）を x_k というように表記し、重み行列 W_{ji} とシグモイド関数よって伝搬（行列計算）する（順伝搬、図 3-4-5.(b)(c)）。

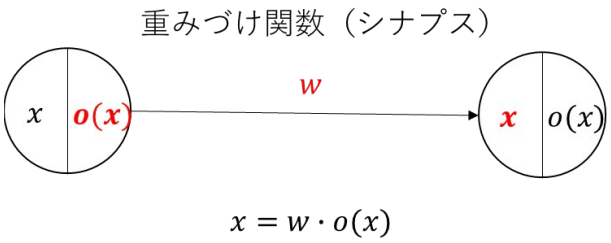
さらに損失関数 E を定義し（図 3-4-5.(d)）、E の W による微分を作り（逆伝搬を意味する図 3-4-5.(f)）、これが充分小さくなるように W の更新を繰り返す（図 3-4-5.(d)(e)）。このとき、η は学習効率（learning rate）を示し、あらかじめ定数で指定する必要がある。このようなプロセスが Neural Network 技術による機械学習を意味する。

本研究では、Python を用いて上述のような 3 層構造の Neural Network プログラムを作成した。第一段階として、手書きの数字を認識させるための機械学習とそのテストを行なった（Neural Network 予備実験の章を参照）。第二段階として、視線データを Fourier 変換した後のデータを Neural Network プログラムによって解析を行なった。

脳神経細胞（ニューロン）の図

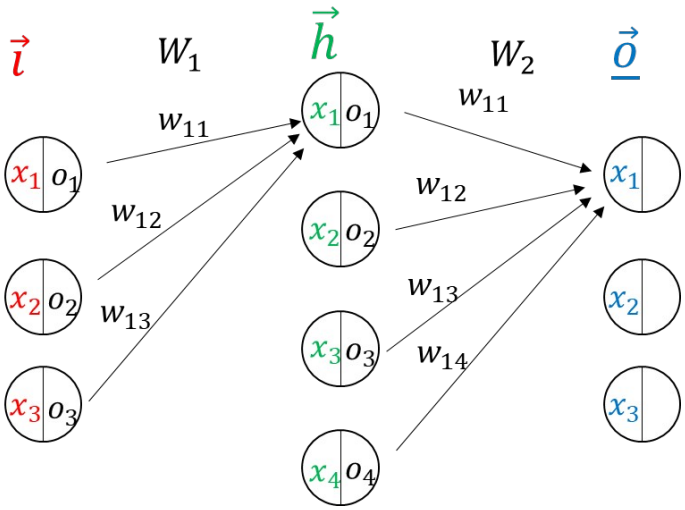


ニューロンのモデル化



$$x = w \cdot o(x)$$

図 3-4-5.　（a）脳神経細胞（ニューロン）とそのモデル化



(b)

$$\vec{h} = W_1 o(\vec{i})$$
$$\vec{o} = W_2 o(\vec{h})$$

$$o(x) = \frac{1}{1+e^{-x}}$$

(c) $$o'(x) = o(x)\big(1 - o(x)\big)$$

図 3-4-5.　（b）Neural Network 順伝搬のイメージ　(c)重み行列とシグモイド関数

(d) $$E = \frac{1}{2}\left|\vec{t} - o(\vec{o})\right|^2$$

$$W \leftarrow W + \eta \frac{\partial E}{\partial W}$$

(e)

図 3-4-5.　（d）損失関数　(e)重み行列の更新

出力層から隠れ層

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_i} \cdot \frac{\partial o_i}{\partial x_i} \cdot \frac{\partial x_i}{\partial w_{ij}}$$
$$= \frac{\partial E}{\partial o_i} \cdot o'_i \cdot o_j$$
$$= (t_i - o_i) \cdot o'_i \cdot o_j$$
$$= \delta_i \cdot o_j$$

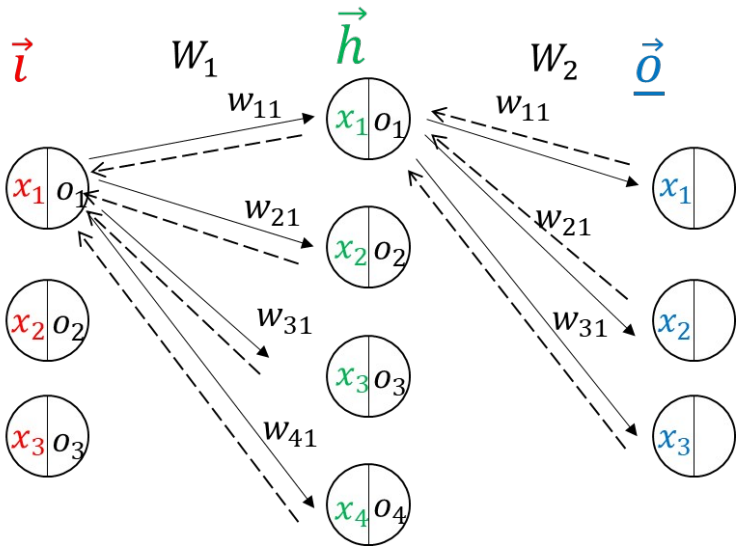
隠れ層から入力層

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial x_i} \cdot \frac{\partial x_i}{\partial w_{ij}}$$
$$= \left(\sum_k \frac{\partial E}{\partial x_k^{next}} \cdot \frac{\partial x_k^{next}}{\partial x_i} \right) \cdot o_j$$
$$= \left(\sum_k \delta_k^{next} \cdot \frac{\partial (w_{ki}^{next} o_i)}{\partial x_i} \right) \cdot o_j$$
$$= \left(\sum_k \delta_k^{next} \cdot w_{ki}^{next} \cdot o'_i \right) \cdot o_j$$
$$= \delta_i \cdot o_j$$

$$\delta_i = \frac{\partial E}{\partial x_i}$$

(f)

図 3-4-5.　（f）誤差逆伝搬



(g)

図 3-4-5.　（g）順伝搬•逆伝搬のイメージ

プログラミングをする際は主に逆伝搬の部分が重要になる。行列で表記すると以下のようになる。「∘」はアダマール積である。

$$\frac{\partial E}{\partial W} = \vec{\delta}\,\vec{o}^{T}$$
$$= \left(W^{next^{T}}\,\overrightarrow{\delta^{next}} \circ \overrightarrow{o'}\right)\vec{o}^{T}$$

詳しい処理はプログラミングに載せている。

第４章　実験結果

4.1.　K-Means 法と Elbow 法を用いた解析

第3章で述べたように、K-Means 法により視線データをクラスタリングするには、あらかじめクラスタ数を指定する必要がある。クラスタ数を決定するために、Elbow 法を用いてデータ解析を行なう。

被験者は FOVE 0 に表示された Target を覚え、次に表示された Test Pattern の中から Target を探す（図 4-1-1.）。取得した視線データを Elbow 法プログラムに取り込み、評価関数の値とクラスタ数の関係を計算する（図 4-1-2.）。この時、クラスタ数の最適値（Elbow Pont）を見積もるために、視線の移動距離を元に閾値を計算した（視点間距離の 2 乗の合計値）。図 4-1-2. 中の赤点線がこの計算された閾値を示しており、この場合クラスタ数 10 が最適値となる。

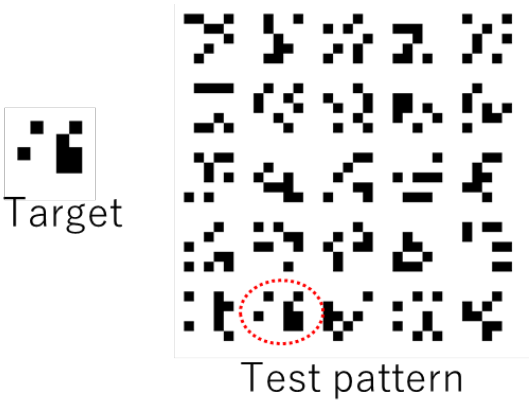


図 4-1-1.　Target と Test Pattern（赤丸が Target の位置）

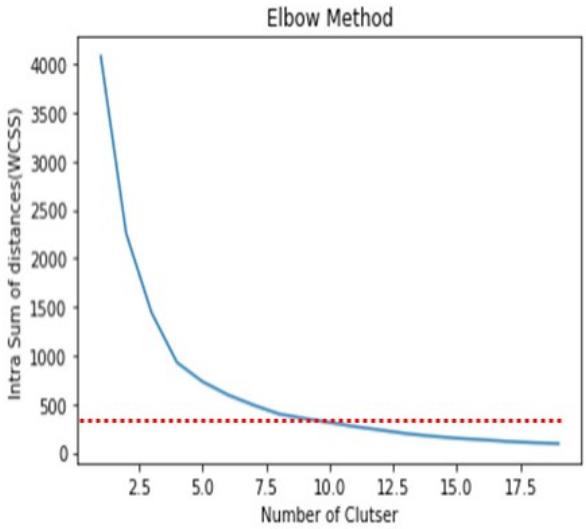


図 4-1-2.　Elbow 法計算結果の例（赤色点線は閾値を示す）

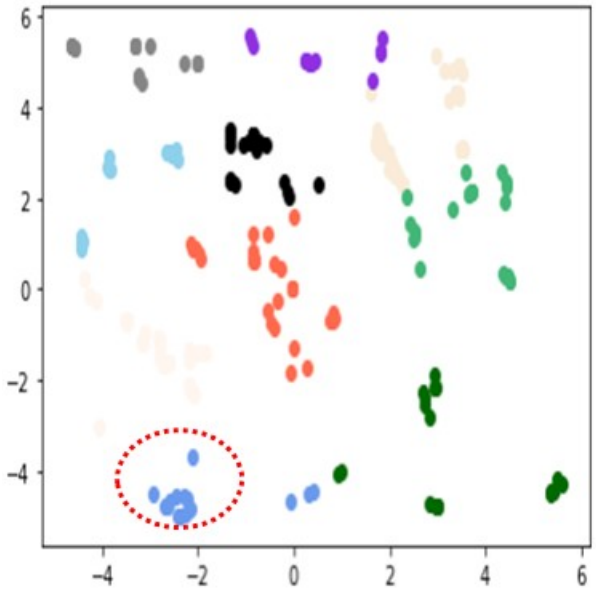
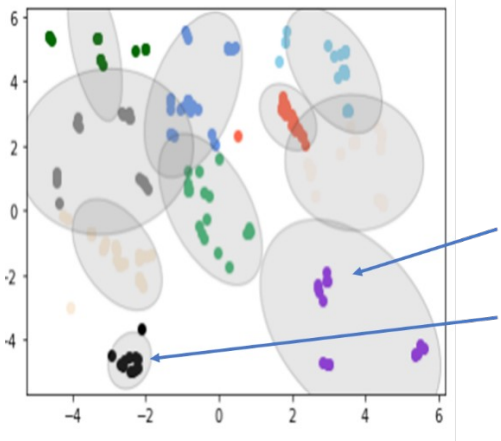


図 4-1-3.　クラスタリングの結果（赤丸は Target 位置を示す）

上記 Elbow 法により計算されたクラスタ数 10 を用いて、クラスタリングを試みた結果が図 4-1-3.である。Target 位置（赤丸で示す）に視点が集中している様子が見て取れる。

4.2.　確率楕円を用いた視線の空間分布解析

クラスタリング後のデータについて、クラスタごとに標準偏差 σ とこれを基にした確率楕円を計算する。長径を $2\sigma_A$、短径を $2\sigma_B$ と定義し、確率楕円を表示したものが図 4-2-1.である。Target 位置の確率楕円は、他の位置の確率楕円に比較して明確に小さくなっており、Target を「凝視」する様子が見て取れる。一方、他の確率楕円は大きく広がる傾向にあり、「探索」状態と対応すると考えられる。なお、Target 発見までの時間（Reaction Time）は 16.1sec.であった



	中心座標	$2\sigma_A$	$2\sigma_B$
Searching	(3.59, -3.56)	7.01	4.26
Stare	(-2.43, -4.65)	1.78	1.15

図 4-2-1.　確率楕円の計算結果

次に、被験者 7 人の実験データについて同様の解析を行ない、クラスタごとの視線密度を計算

した。この結果をまとめたのが図 4-2-2.である。7 人の被験者のうち、5 人において注視点の視点密度が「探索」状態の視点密度より高くなっており、平均的には 1 桁視線密度が上昇している。うち 4 人は、Target 位置を注視しこの位置で最も視線密度が高くなっている。一方、2 人については、視点密度上昇が見あたらず、Target を発見できなかった可能性がある。

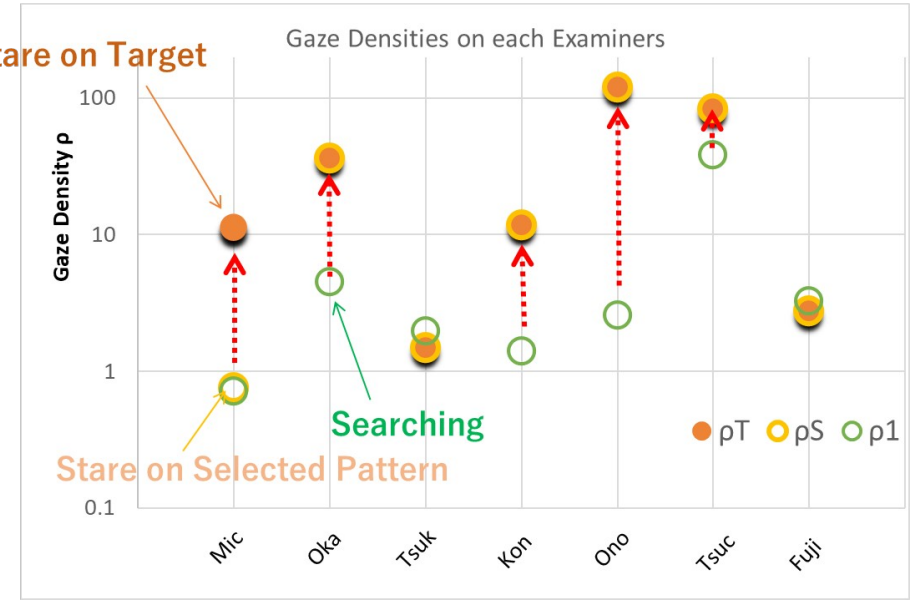


図 4-2-2.　被験者ごとの視線密度（ρT；Target 位置の視線密度、ρS；凝視点（Target 以外）位置の視線密度、ρ1；「探索」状態の視線密度）

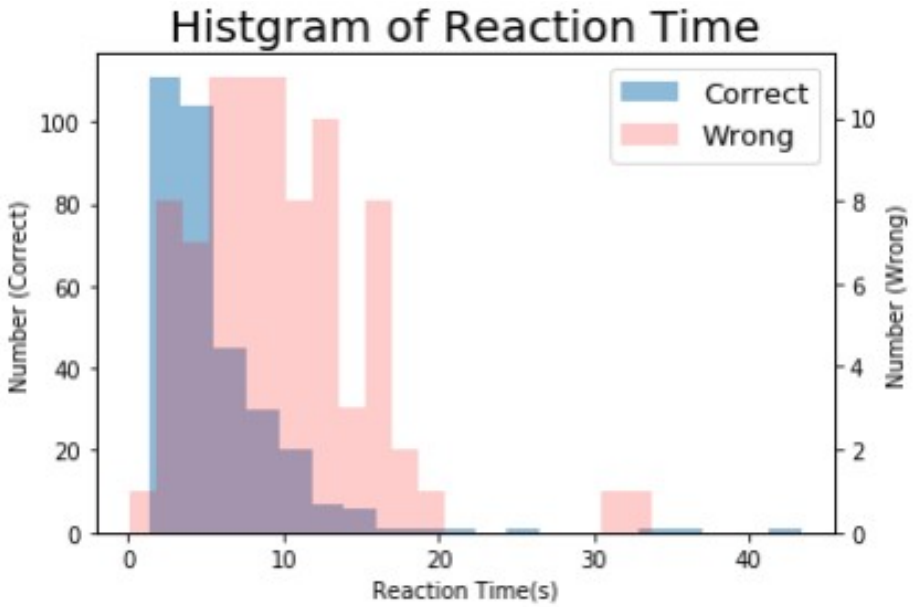


図 4-2-3.　Target を正しく発見した場合と誤った場合の Reaction Time 比較

さらに、全ての実験における Reaction Time をまとめた（図 4-2-3.）。正しく Target を発見した場合、Reaction Time の最頻値は 2-3 秒でその付近に集中しており、誤った場合の最頻値は

6-10 秒および広く分布しており、明らかな差が観測できた。

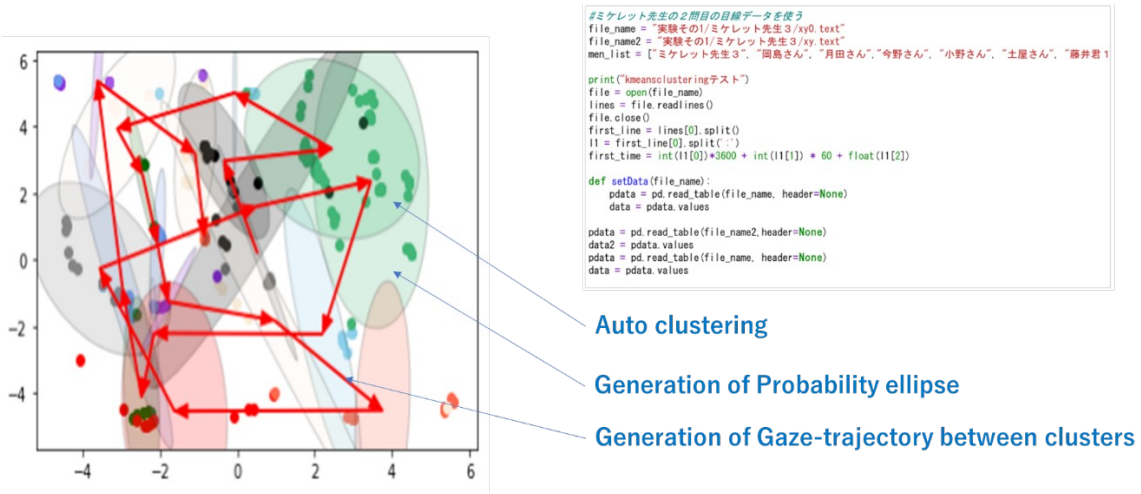


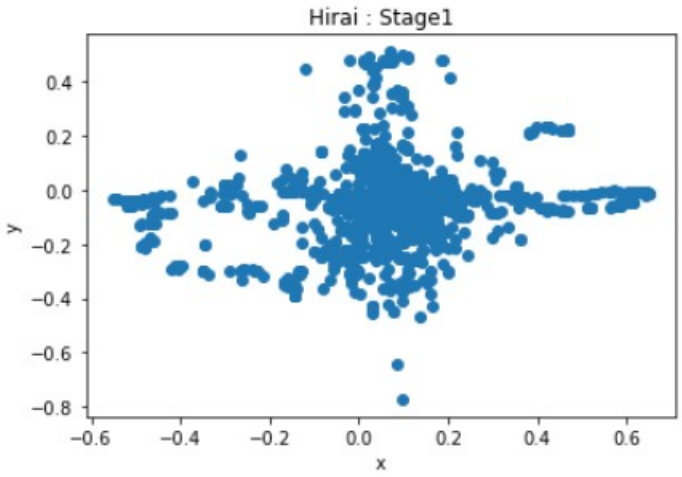
図 4-2-4.　視線データの自動解析

一方、視線の空間分布解析手法を時系列の解析に適用する試みを行なった。図 4-2-4.は本研究における、クラスタリング、確率楕円の生成、視線密度の計算を、時系列解析に底要した例を示したものである。クラスタ間の視線移動を自動計算し、矢印で示した。確率楕円は重なり合ってしまい、適切なクラスタリングができているとは思えない。「凝視」状態の視線が他の領域にジャンプしてから戻ってくるような動きが頻発しており、この視線の空間解析方法が、時系列、過渡的解析には適切でない、と言えるかもしれない。

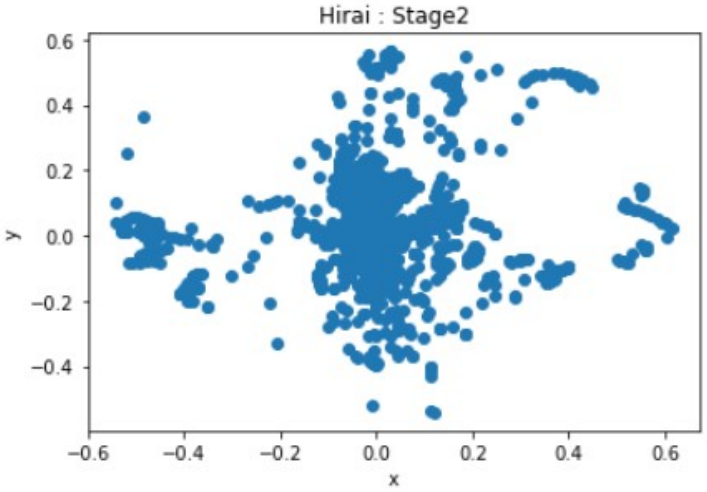
4.3.　Fourier 変換を用いた過渡分布解析

前述のように、視線の空間解析方法では適切な過渡的解析ができない、という観点から、過渡的解析により適した、Fourier 変換を用いたかと分布解析を提案することにした。

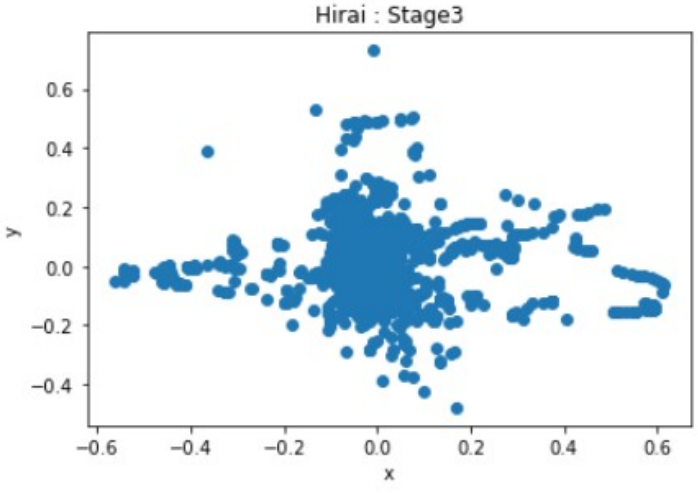
まず、前記 3.3.2.視野変動状態でのゲーム形式実験（視線の過渡解析）で得た、視線データを元に Fourier 変換を用いた過渡解析を行なう。視線データは、被験者の正面を仮想的に原点として、時間と視線座標がセットで記録される。この視線データを 2D 空間にプロットしたものが図 4-3-1.である。(a)-(d)は、それぞれ Stage-1～4 を示している。おおむね、視野中心付近に視点が集中しているが、Stage によって傾向が異なるように見える。


Hirai : Stage1
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
y
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6
x

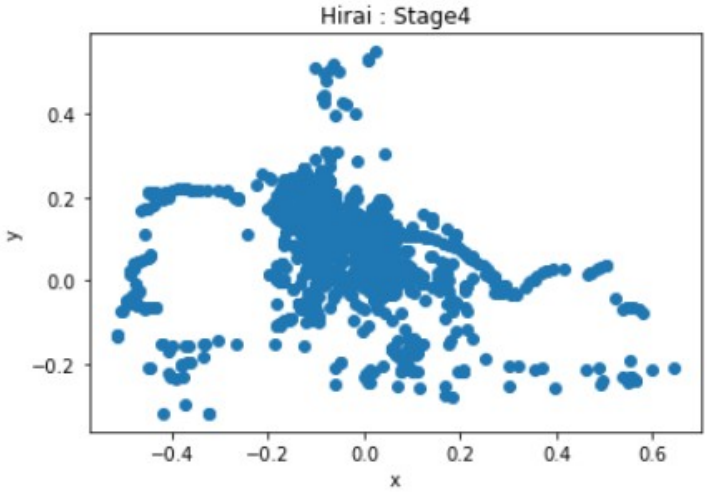
(a)


Hirai : Stage2
0.6
0.4
0.2
0.0
−0.2
−0.4
y
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6
x

(b)


Hirai : Stage3
0.6
0.4
0.2
0.0
−0.2
−0.4
y
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6
x

(c)

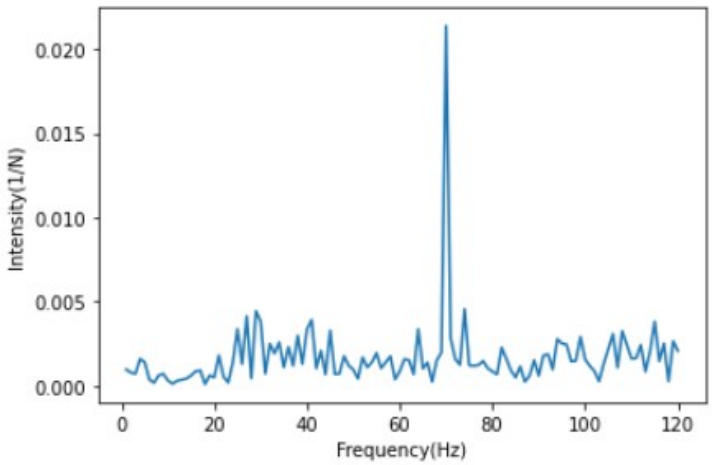


(d)

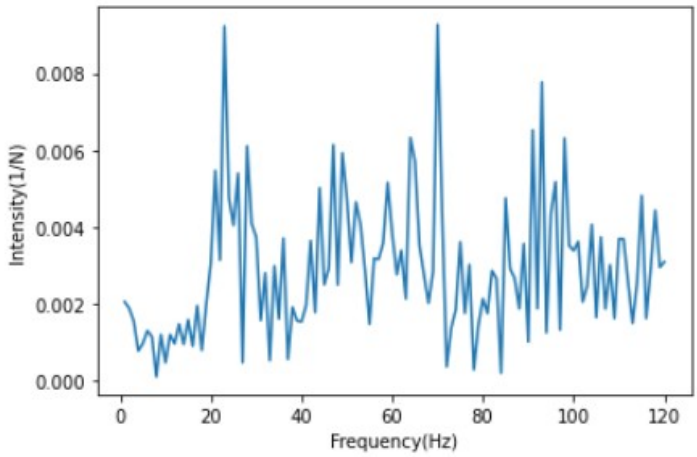
図 4-3-1.　視野変動状態でのゲーム形式実験データ　(a)Stage-1、(b)Stage-2、(c)Stage-3、(d)Stage-4

次に、この視線データを Fourier 変換した結果が図 4-3-2.である。X 方向の視線の動きに対応する周波数成分ごとの強度をプロットしてある（Y 方向の視線の動きについては反映していない）。

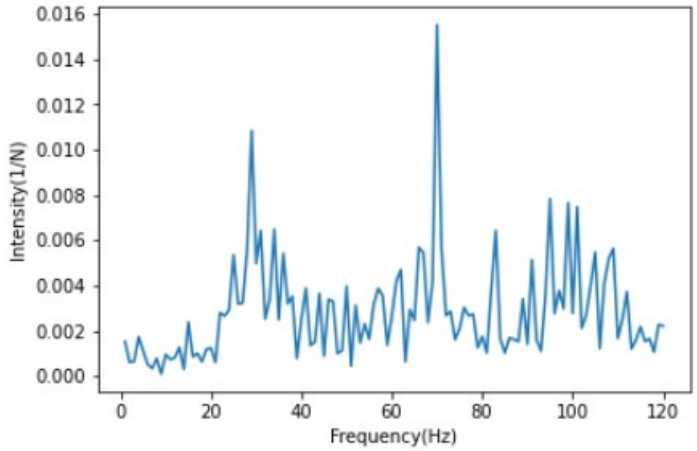
視線データは、およそ 30point/sec.で取得されているが、視点が素早く動くと 30Hz より速い周波数でフィッティングされると考えられるので、高周波数の成分も有効と考えている。



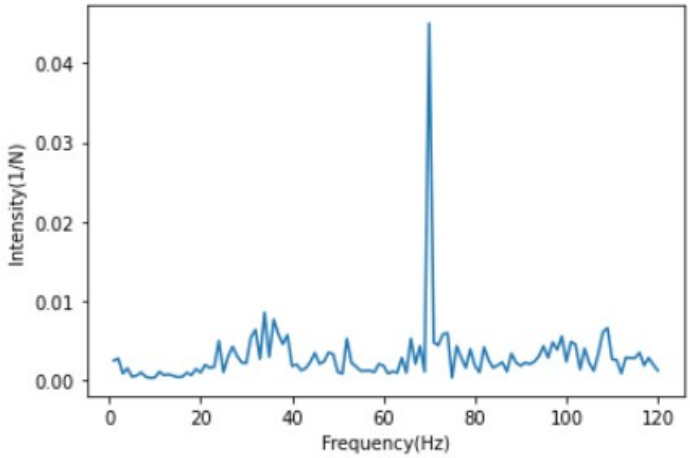
(a)



(b)



(c)



(d)

図 4-3-2.　Fourier 変換後の視野変動状態でのゲーム形式実験データ　(a)Stage-1、(b)Stage-2、(c)Stage-3、(d)Stage-4

各ステージ間のグラフの形はそれぞれ異なっているように見える。目視によって視線分布の過渡的傾向を見出すのは困難と考え、本研究では、このグラフの形を人工知能的処理によって判定させる試みを行なうことにした。

4.4. Neural Network 技術を用いた解析

4.3.で行なった Fourier 変換したデータに関して、Neural Network 技術を用いた自動判別を試みる。

準備として、図 4-3-2.のデータのうち、どの程度のデータサイズを切り出し判別するのが妥当か、を検討することにした。図 4-4-1.は、データサイズとそのデータ（Intensity）の分散の合計の関係を示したものである。この結果を見ると、データサイズおよそ 800 以上になると、分散の合計値はあまり変化しなくなる。このことから、データサイズ 800 ぶん（約 27 秒分のデータに相当する）を、切り取って Neural Network システムに読み込ませることにする。

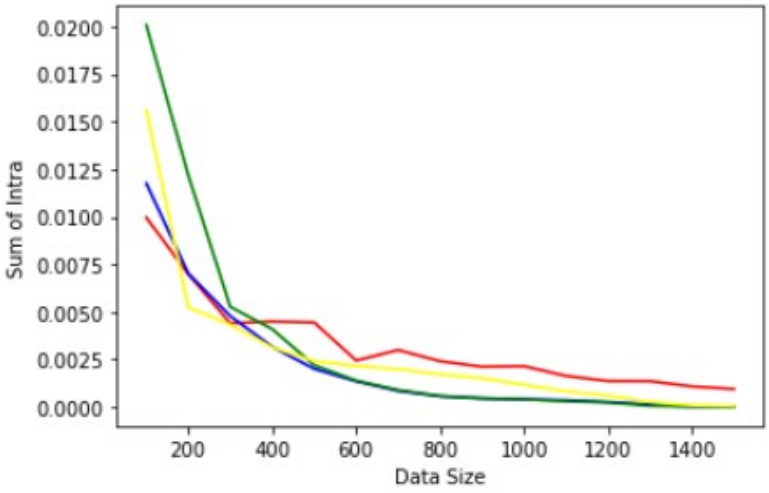


図 4-4-1. データサイズ最適化の試み（最適値 800）

被験者 H における Stage-1～4 のデータから、データサイズ 800 ぶんをランダムに切り取り、1,000 個ずつ Neural Network システムに読み込ませて学習させ、重み行列を最適化した。その後、再びデータサイズ 800 ぶんをシステムに読み込ませ、正しく判定できるかをテストした結果が図 4-4-2.である。例えば、Stage-1 のデータについては、1,000 このデータすべてが正しく Stage-1 と判定され、Stage-2 のデータについては、1,000 個のうち 997 個を正しく Stage-2 と判定されたことを示している。

このことから、この Neural Network システムは、99.9%の確率で、同一集団の Stage-1～4 のデータを正しく判断できたことになる。

```
[[1000.    0.    0.    0.]   ←Stage-1のデータ解析結果
 [   3.  997.    0.    0.]   ←Stage-2のデータ解析結果
 [   0.    0. 1000.    0.]   ←Stage-3のデータ解析結果
 [   0.    0.    0. 1000.]] ←Stage-4のデータ解析結果
accuracy: 99.9%
```

図 4-4-2.　Neural Network システムの学習検証結果（Confusion Matrix）

次に、被験者 H における最適化結果を用いたシステムに、他の被験者 4 人の Fourier 変換後の視線データを判定させる試みを行なった。図 4-4-3.の下段がその結果を示しているが、全体平均で正答率 30.9%と大幅に正答率が落ちている。最適化集団と異なるデータを評価した結果なので、正答率が低下するのは当然であるが、Stage-2、Stage-3 の視線データにおいては、それぞれ Stage-2、Stage-3 と判別したデータ数が、他のステージと判定したものより少なくとも 40～60%多く、最も高い水準となっている。このことは、Stage-2 および Stage-3 においては、異なる被験者に共通の（時定数や周波数依存の）視線の動きが含まれることを示唆している、と考えられる。

```
        x, y = detecVR.test(vec)
        confusion_matrix[j, np.argmax(y.reshape(1, -1)[0])] += 1.0
a = 0.0
for i in range(4):
    a += confusion_matrix[i, i]
print(confusion_matrix)
print("accuracy: " + str(format(a / (epoch * 4) * 100.0 ,'.1f')) + "%")

#Testing in Others
confusion_matrix = np.zeros((4, 4))
n = len(directions)
for num in range(1, n):
    for i in range(epoch):
        for j in range(4):
            temp = data[num][j]
            N = len(temp[:, 0])
            rand = np.random.randint(N - data_length)
            vec = rfourier(temp[rand:rand + data_length, 0], temp[rand:rand + data_length, 1], 120) * scale
            ans = np.zeros(4) + 0.1
            ans[j] = 0.9
            x, y = detecVR.test(vec)
            confusion_matrix[j, np.argmax(y.reshape(1, -1)[0])] += 1.0
a = 0.0
for i in range(4):
    a += confusion_matrix[i, i]
print(confusion_matrix)
print("accuracy: " + str(format(a / (epoch * 4 *(n - 1)) * 100.0 ,'.1f')) + "%")
```

```
[[944.   0.  39.  17.]          ← Stage-1の正答率  94.4%
 [  0. 974.  16.  10.]          ← Stage-2の正答率  97.4%
 [  0.   0. 988.  12.]          ← Stage-3の正答率  98.8%
 [  1.   5.  20. 974.]]         ← Stage-4の正答率  97.4%
accuracy: 97.0%
[[ 555. 1022.  680. 1743.]      ← Stage-1の正答率  13.9%
 [ 506. 1428. 1071.  995.]      ← Stage-2の正答率  35.7%
 [ 580.  925. 1549.  946.]      ← Stage-3の正答率  38.7%
 [ 437. 1388.  770. 1405.]]     ← Stage-4の正答率  35.1%
accuracy: 30.9%
```

被験者Hの判定結果

Hを除く被験者の判定結果

図 4-4-3.　Neural Network システムの学習検証結果（多数被験者）

さらに、被験者 H における最適化結果を用いたシステムに、他の被験者の Fourier 変換後の視線データを「個別に」判定させる試みを行なった。図 4-4-4.がその結果を示しているが、被験者によって正答率が大きく異なっている。最適化集団と異なるデータを評価したにもかかわらず、高い正答率を示す被験者もいる（被験者 M）。Stage-1 については、総じて正答率が低い点も興味深い。被験者によって、（時定数や周波数依存の）視線の動きが類似しているケースと異なるケースが存在している、ということを示している。

```
#Testing in Others
n = len(directions)
for num in range(1, n):
    confusion_matrix = np.zeros((4, 4))
    for i in range(epoch):
        for j in range(4):
            temp = data[num][j]
            N = len(temp[:, 0])
            rand = np.random.randint(N - data_length)
            vec = rfourier(temp[rand:rand + data_length, 0], temp[rand:rand + data_length, 1], 120)
            ans = np.zeros(4) + 0.1
            ans[j] = 0.9
            x, y = detecVR.test(vec)
            confusion_matrix[j, np.argmax(y.reshape(1, -1)[0])] += 1.0
    a = 0.0
    for i in range(4):
        a += confusion_matrix[i, i]
    print("Testing in " + directions[num] +" data")
    print(confusion_matrix)
    print("accuracy: " + str(format(a / (epoch * 4) * 100.0 ,'.1f')) + "%")
```

```
Testing in 被験者F data
[[  4. 565. 364.  67.]
 [ 39. 163. 701.  97.]
 [ 64. 187. 488. 261.]
 [  0. 521. 449.  30.]]
accuracy: 17.1%
Testing in 被験者O  data
[[ 60. 633. 284.  23.]
 [148. 636. 211.   5.]
 [166. 720.  48.  66.]
 [116. 658.  92. 134.]]
accuracy: 21.9%
Testing in 被験者T  data
[[ 36. 216. 314. 434.]
 [117. 421. 422.  40.]
 [ 25.  73. 902.   0.]
 [  0. 614. 371.  15.]]
accuracy: 34.4%
Testing in 被験者M  data
[[245.  38. 182. 535.]
 [  0. 524. 174. 302.]
 [  3.  44. 934.  19.]
 [  1.  90.  63. 846.]]
accuracy: 63.7%
```

図 4-4-4.　Neural Network システムの学習検証結果（個別被験者）

第5章　考察

5.1.　数理解析結果に関する考察

5.1.1.　K-Means 法と Elbow 法を用いた解析

視覚探索に関して、被験者がどこを凝視しているかといった解析に際し、K-Means 法および Elbow 法を用いてクラスタリングを行なうことを提案し、それが妥当であることを示した。

この結果、先行研究の解析に比べて、高速かつ機械的に分析することが可能なうえ、だれが見てもわかりやすい表現が可能となった。

一方、クラスタ数を決定する手法として、今回 Elbow 法を用いたが、シルエットプロットなどを用いた解析なども存在するので、比較検討などを行なう必要があるかもしれない。

5.1.2.　確率楕円を用いた視線の空間分布解析

視点の範囲を限定し、視線密度を算出した。凝視点においては視線密度が上昇していることがわかるようになった。

本研究により、どのあたりを重点的に凝視しているか、定量的に判定することが可能となった。

また、さらにクラスタリング技術を進めることができれば、凝視点の指定などをせず、一般的な環境での（任意の VR 機器も使用可能な）応用が可能となると思われる。

さらに、問題の出題を工夫することで視線入力デバイスへの応用も可能である。

視点の分散がそもそもガウス分布に沿うものであるかどうかは議論の余地がある。視線が遷移する過程においては特に別のモデルを適応される必要があると思う。

5.1.3.　Fourier 変換を用いた過渡分布解析

本研究では、ゲーム形式テストにおける視線データの Fourier 変換による過渡分析法を提案・実行し、視線の動きの周波数に依存した成分の抽出に成功した。

今回用いた Fourier 変換は考慮するパラメータが波の位置だけでなく、時間も含んでいるため仮に得られたデータの時間がバラバラであっても Fourier 変換することができる。

また、Fourier 関数の絶対値をとることで位相差も考慮されている。

本研究における Fourier 変換では、Excel などの高速 Fourier 変換などより高い周波数帯で Fourier 変換を行ったので、低周波帯の Fourier 変換などとの比較検証を今後する必要があると思われる。

5.2.　Neural Network 解析に関する考察

Neural Network 技術を用いて、Fourier 変換後の視線データの判別を試みる方法を提案し

た。その結果、ゲーム形式テストにおける各 Stage を判別することができた。この結果は、本研究において非常に重大な結果であると思われる。当初揺れこそが最も重要なファクターであると思われていたが、解析の結果見ている物体や情景などがそれと同じかそれ以上の要因を与えていることがわかる。

本来、科学的アプローチであれば、根拠を示すために仮に識別するようなモデルがあったとしても多角的な検証が必要になるが、今回用いた Neural Network は精度を高めることを目的としているので、問題を設定すればそれに応じた実用的かつ制度の高い解答が可能である。

仮に酔いやすい傾向、ステージなどを個別に設定すれば瞬時に実用レベルの導入が可能である。

本研究において、Neural Network 技術の課題も明らかになった。

一つは、判別に必要なキーパラメータを、一般に明らかにできない、という点である。Neural Network 技術はいわゆるデータサイエンスの領域であり、「相関性」を明らかにしようとするものである。それゆえ、「相関理由」「因果関係」を人に説明するといったことが大変難しいように感じた。データサイエンスの目的は精度を上げることであるが、科学は様々な分野から多面的な検討をするので、データサイエンスと科学は異なる分野であることが Neural Network 技術を用いたことに関する閾だと感じた。

もう一つは、苦手な解析分野が存在する、という点である。例えば、以下のような問題を設定するとする。

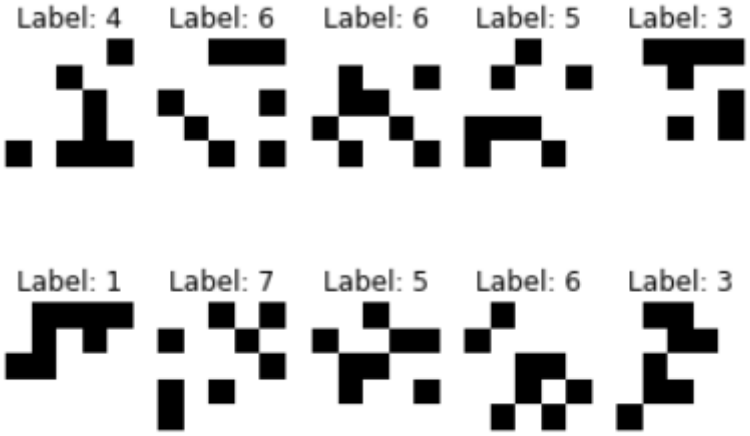


図 5-2-1.　Neural Network システムの苦手分野

この例では、分割された画素の数をラベルとして Neural Network に学習させるわけであるが、これは位置との相関がないために Neural Network 解析では良い結果を得ることができない。

このようなことから、Neural Network においては位置との相関が重要であり視線の情報には判別できるような相関が存在することがわかる。

5.3.　VR 酔いに関する考察

ゲーム形式テストにおいて、被験者に「酔い」を感じたかを答えてもらった。図 5-3-1.に、被験者ごとの酔いを感じる度合い（5 段階評価）を示す。

この結果、Stage-2、Stage-3 は、やや酔いやすいコンテンツである、という結果であった。4.4.項の Neural Network 技術による解析結果において、Stage-2、Stage-3 は正答率が高かった事実と符合する。この点は大変興味深く、Neural Network 技術が、被験者に依存しない、VR 酔いに関連した共通の視線の動きを捉えたと考えることも可能であろう。Stage-2、Stage-3 は、視界の揺れがあり、なおかつ背景表示があるコンテンツである（一方、Stage-1 は揺れが無く、Stage-4 は背景表示が無い）。現時点ではっきりと特定はできないが、Stage-2、Stage-3 では、精神的緊張の中で視線の低周波領域での動きが顕著になるのではないか、と想像している。その視線の特徴的な動きを特定するためには、重みづけ行列 W の中身を解析する必要があるであろう。

	Stage1	Stage2	Stage3	Stage4	
被験者 H	0	3	2	1	
被験者 T	0	1	2	2	
被験者 O	0	2	3	1	
被験者 F	0	2	2	2	
被験者 M		0	2	1	1

図 5-3-1.　被験者ごとの「酔い」レベル（5 段階評価）

また、ニューラルネットワークに以下のような周波数帯に注目して、識別していると思われる。これはフーリエ変換の特性、ニューラルネットワークの分析結果がアンケートと一致している。このことから実際に導入する際は 20Hz～80Hz 付近の成分に注目して識別することが適切だと思われる（図 5-3-2.）。

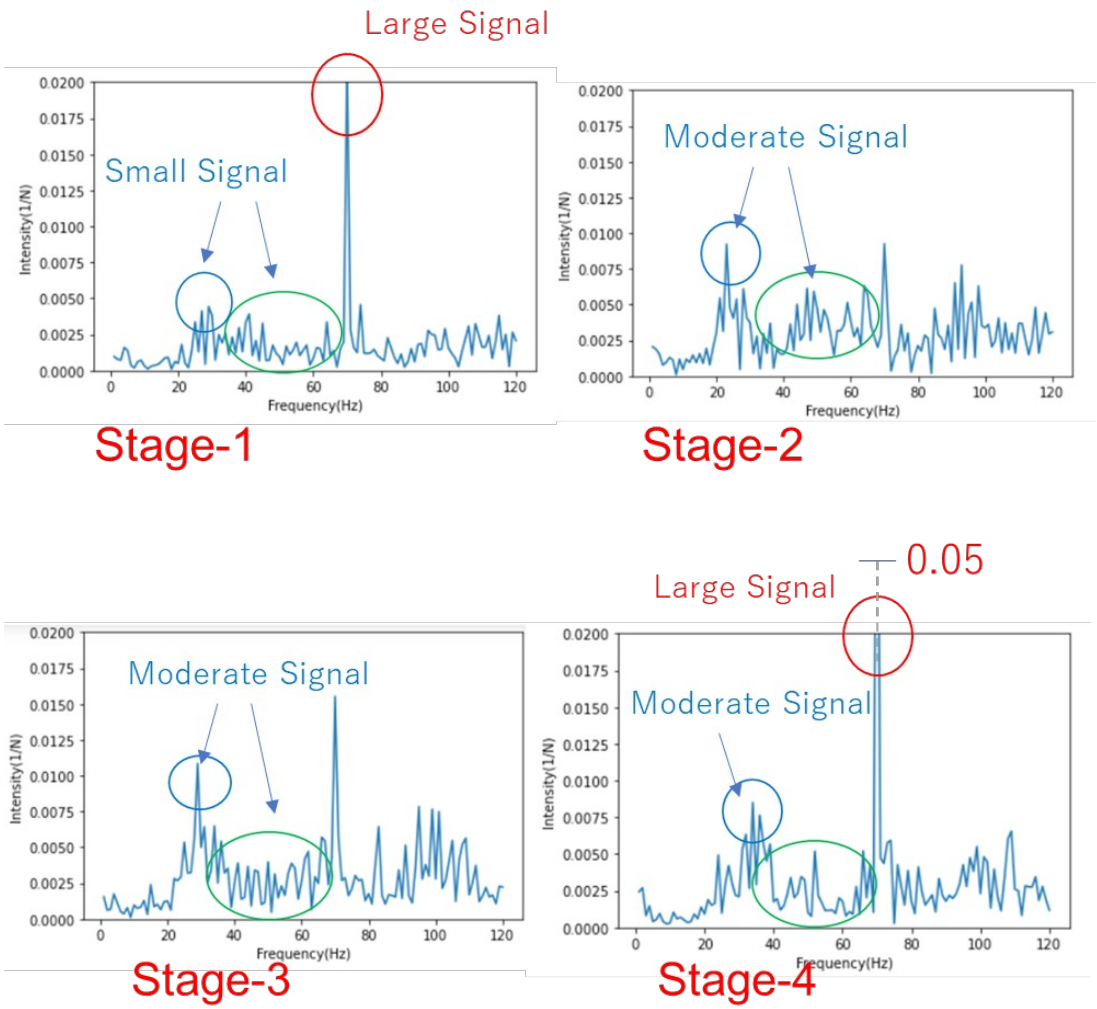


図 5-3-2.　Fourier 変換後の各ステージスペクトル

一方、VR 酔いの感じ方はやはり個人差が大きいという背景は依然として問題である。図 4-4-4. や、アンケート結果のばらつきもこれを反映している可能性がある。

また VR 酔いは避けられない問題である可能性があるため、解析によっては被験者の VR に対する耐性を検出できる可能性もある。

また個人差があることは実際には指紋認証やサインのように、生体認証やそれに類する認証に利用ができる可能性が存在する。

また実用に関してであるが、VR 機器に気持ち悪くなった時の目線の動きやステージの特徴などを登録しておくことにより、VR 酔いを検知し、アラートなどを鳴らす仕組みなどを作る際にこの研究結果は大変有用であると考えられる。

第６章　まとめ

これまで述べてきたように、本研究において、視覚探索と VR 酔いに関する視線の分析を試み、それぞれの特徴を得ることができた。そして、（K-Means 法や Neural Network 技術のような）機械学習を用いた分析に関しては、本研究が初めての試みである。

しかし、視線の動きを解析する手段として、数理解析や Neural Network 技術が有効であることは示せたが、科学的根拠を充分に示すことは出来ていないと考えている。今後の進展を望みたい。

また、VR 研究というものは認知してもらうということが最も重要な点である。いまだにほとんどの家庭において VR 機器は存在しないうえに、VR 酔いが何かといったことに関してもほとんど認知されていない。

しかし、コロナウイルスの感染拡大に際し、密を避ける取り組みとして VR を用いたイベントや開発がなされるようになってきた。

また、医療現場における手術支援や遠隔による手術などが、VR 技術のアプリケーションとして試みられるようになった。

海外においては、軍用のシミュレータや作戦支援の場に VR 技術が積極的に用いられるようになっている。

今後は、遠隔操作やコミュニケーションツールとして、または自動運転技術への適用など、さらなる試みがなされると思われる。

このようなことは VR 業界において大変喜ばしいことであり、一層の発展を期待する。

また VR 研究の問題点としては VR だけでは研究の対象とはなりえない点にある。

データを取ることは比較的容易にはなってきているが、何を目的に、どのような解析をするかが最も重要な点である。

視点の動きである「探索」「凝視」はその一端であるが、顔認識などのセキュリティ関連への適用もなされている。今後、どのような方向性を持って解析に当たるか、大変興味のある点である。

一方機械学習は、膨大なデータを効率的に解析する方法として試みられてきた技術であり、今後 VR だけではなくほかの科学的分野にも適用され、発展することを期待している。

しかしながら、機械学習技術では、データの相関を明確にするには優れているものの、因果関係を発見することは極めて難しい。このあたりの問題をきちんと理解しながら、機械学習技術を発展させてゆく必要があるだろう。

さらに、認知情報科学やその他の分野に関してもアンケート中心などではないデータサイエンス、数理解析的な手法を用いた発展を期待している。

以上を持って今回の研究のまとめとさせていただきたいと思う。

謝辞

本研究を進めるにあたり、研究テーマの設定、実験方法、データ分析方法などすべてにおいてご指導賜りましたルジェロ・ミケレット教授には心から感謝申し上げます。また、視覚情報研究室のメンバーの方々には研究のご協力、意見なども多くいただきました。この場を借りて感謝申し上げます。

また本研究において、被験者を快く引き受けてくださった皆様に感謝いたします。

最後に、本論文執筆に関わってくださった全ての方々に感謝の意を示し、謝辞といたします。

文献

[1] Goldman, David (2012 年 4 月 4 日). “Google unveils 'Project Glass' virtual-reality glasses”. Money (CNN)
[2] “Oculus Rift virtual reality headset gets Kickstarter cash”. BBC (2012 年 8 月 1 日)
[3] "Augmented Reality edges closer to the mainstream - CNET News"
[4] https://artillry.co/artillry-intelligence/vr-global-revenue-forecast-2018-2023/
[5] R.J.K. Jacob. What you look at is what you get: eye movement based interaction techniques. CHI 1990, pp.11-18, 1990.
[6] “StarVR one”. https://www.starvr.com/products/
[7] “Tobii Pro VR Integration”.
https://www.tobiipro.com/ja/product-listing/vr-integration/
[8] “FOVE Rye Tracking Virtual Reality Headset: Home”. https://www.getfove.com/
[9] 古柳俊佑, 蔡東生. 没入型 VR における動的被写界深度効果の実装と評価. 情報処理学会研究報告, 2012, vol. 2012-CG-146, no.26, p. 1-6
[10]第 18 回情報科学技術フォーラム（FIT2019）予稿集　N-012、No.4　P.333
[11] https://www.tobiipro.com/ja/
[12] K. Naqshbandi, T. Gedeon and U. A. Abdulla, (2016) “Automatic clustering of eye gaze data for machine learning”, Proc. IEEE Int. Conf. on Sys., Man and Cyber. (SMC 2016)
[13] Steinhaus, H. (1957). “Sur la division des corps matériels en parties” (French). Bull. Acad. Polon. Sci. 4 (12): P.801–804
[14] Homma, T., Les, A., Robert, M. II (1988) “An Artificial Neural Network for Spatio-Temporal Bipolar Patters: Application to Phoneme Classification”. Advances in Neural Information Processing Systems 1: P.31–40.

学会報告実績

日本視覚学会 2021　冬季大会　2021 年 1 月 20 日～22 日

3o07　“Visual search using VR environment and mathematical analysis of eye-gaze tracking data”　平井亮、　ルジェロ-ミケレット（横浜市立大学）

Neural Network 予備実験

前記 3.4.6.項で行なう Neural Network 解析に先立って、機械学習プログラムが機能するかどうかを確かめるために予備実験を行なった。

図 S.1.のような、機械学習用プログラムを作成した。

```
import numpy as np
class Neural_Network:
    def f(self, x):
        return 1.0 / (1.0 + np.exp(-x))

    def df(self, x):
        return x * (1.0 - x)

    def __init__(self, i, h, o, lr, mu):
        self.w1 = np.random.normal(0.0, 1.0, (h, i))
        self.w2 = np.random.normal(0.0, 1.0, (o, h))
        self.m1 = np.zeros((h, i))
        self.m2 = np.zeros((o, h))
        self.lr = lr
        self.mu = mu

    def test(self, i):
        i_data = i.reshape(-1, 1)
        x = self.f(np.dot(self.w1, i_data))
        y = self.f(np.dot(self.w2, x))
        return x, y

    def train(self, i, t):
        i_data = i.reshape(-1, 1)
        t_data = t.reshape(-1, 1)
        x, y = self.test(i_data)

        d2 = (t_data - y) * self.df(y)
        self.w2 += self.lr * np.dot(d2, x.reshape(1, -1)) + self.mu*self.m2
        self.m2 = self.lr * np.dot(d2, x.reshape(1, -1))

        d1 = np.dot(self.w2.T, d2) * self.df(x)
        self.w1 += self.lr * np.dot(d1, i_data.reshape(1, -1)) + self.mu*self.m1
        self.m1 = self.lr * np.dot(d1, i_data.reshape(1, -1))
```

図 S.1.　Neural Network 機械学習用テストプログラム

この機械学習プログラムを評価するために、2つの予備実験を行なった。

1つ目は、XNOR ゲートとしての評価である（図 S.2.）。学習回数を 30,000 回実施した後、プログラムに XNOR ゲートとしての判断をさせたところ、ほぼ間違いなしに判断できることを示している。

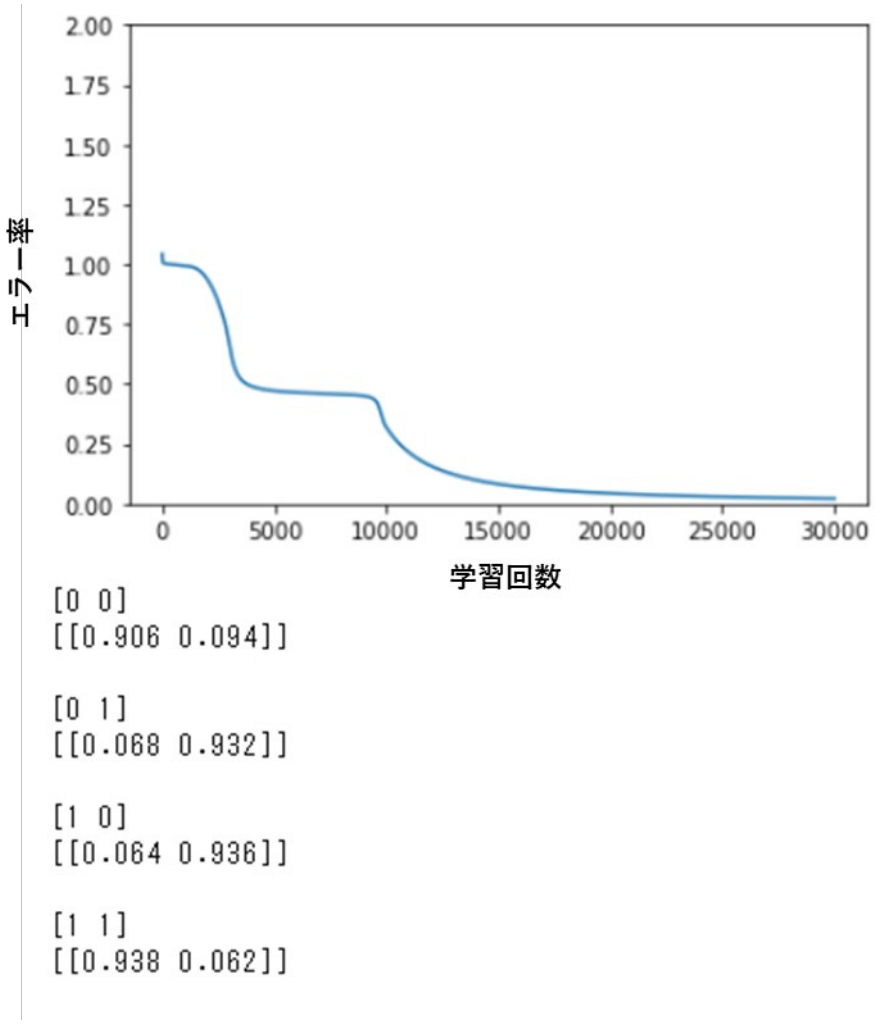
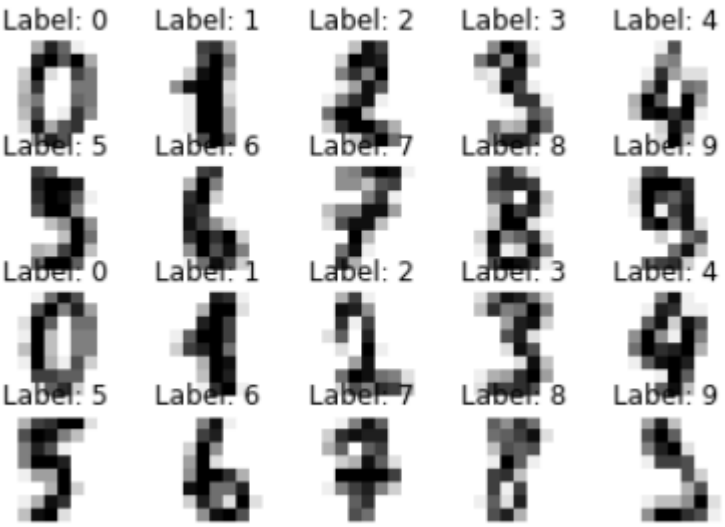


図 S.2.　機械学習用プログラムによる XNOR ゲート評価結果

2つ目は、手書き数字の判定評価である。

図 S.3.に、手書き数字の例と評価結果を示す。180,000 回の学習後、手書き数字の判定実験を行なったところ、99.6%の破堤率を得ることができた。

```
(1797, 8, 8)
```



(a)

```
[[178.   0.   0.   0.   0.   0.   0.   0.   0.   0.]
 [  0. 182.   0.   0.   0.   0.   0.   0.   0.   0.]
 [  0.   0. 177.   0.   0.   0.   0.   0.   0.   0.]
 [  0.   0.   0. 183.   0.   0.   0.   0.   0.   0.]
 [  0.   0.   0.   0. 178.   0.   0.   0.   1.   2.]
 [  1.   0.   0.   0.   0. 180.   0.   0.   0.   1.]
 [  0.   0.   0.   0.   0.   0. 181.   0.   0.   0.]
 [  0.   0.   0.   0.   0.   0.   0. 179.   0.   0.]
 [  0.   1.   0.   0.   0.   0.   0.   0. 173.   0.]
 [  0.   0.   0.   1.   0.   0.   0.   0.   0. 179.]]
accuracy: 99.6%
```

(b)

図 S.3.　機械学習用プログラムによる手書き数字の評価結果

(a)手書き数字の例　(b)評価結果

プログラミングコード

K-Means 法

```
class Machine:
    def __init__(self, file_name, number):
        self.file_name = file_name
        self.number = number
        self.result = pd.read_table(file_name+'/testResult.text')
        temp = ''
        pdata = pd.read_table(file_name+'/xy'+str(number)+'.text', header=None)
        self.xydata = pdata.values
        if number > 0:
            temp = str(number-1)
        pdata = pd.read_table(file_name+'/p25_'+temp+'.text', header=None)
        self.pdata = pdata
        #エルボーの閾値を決める
        self.elbow_threshold = 0
        for j in range(len(self.xydata)):
            if j < len(self.xydata) - 1:
                x_sq = pow(self.xydata[j+1][1]-self.xydata[j][1],2)
                y_sq = pow(self.xydata[j+1][2]-self.xydata[j][2],2)
                self.elbow_threshold += (x_sq + y_sq)
        #nを決める
        max_cluster = 25
        clusters_ = range(1, max_cluster)
        self.intra_sum_of_square_list = []
        n_array = []
        for k in clusters_:
            km = KMeans(n_clusters=k, init='k-means++', n_init=10, max_iter=300)
            km.fit(self.xydata[:,[1,2]])
            self.intra_sum_of_square_list.append(km.inertia_)
            if km.inertia_ < self.elbow_threshold:
                n_array.append(k)
        n_clusters = 1
        if len(n_array) > 0:
            n_clusters = min(n_array)
        self.n_clusters = n_clusters
        #k-meansの適応
        km = KMeans(n_clusters=n_clusters, init='k-means++', n_init=10, max_iter=300)
        km.fit(self.xydata[:,[1,2]])
        self.cluster_labels = km.predict(self.xydata[:,[1,2]])

    def elbow_plot(self):
        #エルボープロット
        max_cluster = 25
        clusters_ = range(1, max_cluster)
        fig = plt.figure()
        ax = fig.add_subplot(1, 1, 1)
        ax.set_title('Elbow Method')
        ax.set_xlabel('Number of Clutser')
        ax.set_ylabel('Intra Sum of distances(WCSS)')
        plt.plot(clusters_, self.intra_sum_of_square_list)
        ax.axhline(y=self.elbow_threshold, color='red', linestyle='--')
```

```
def cluster_plot(self):
    #楕円プロット
    fig = plt.figure()
    ax = fig.add_subplot(1,1,1)
    colorlist = ['tomato', 'antiquewhite', 'blueviolet', 'cornflowerblue', 'darkgreen',
    cluster_ids = list(set(self.cluster_labels))
    for cluster_id in cluster_ids:
        label_ = 'cluster = %d' % cluster_id
        data_by_cluster = self.xydata[self.cluster_labels == cluster_id]
        #data_by_cluster = data[cluster_labels == cluster_id]
        color = colorlist[cluster_id%len(colorlist)]
        ax.scatter(data_by_cluster[:,1],data_by_cluster[:,2], c=color, label = label_)
        tempdata = data_by_cluster[:,[1,2]]
        el = ConfidenceEllipse(tempdata, p=0.95)
        ax.add_artist(el.get_patch(face_color=color, alpha=0.2))
```

確率楕円

```
from matplotlib.patches import Ellipse

class ConfidenceEllipse:
    def __init__(self, data, p=0.95):
        self.data = data
        self.p = p

        self.means = np.mean(data, axis=0)

        x = []
        y = []
        for i in data:
            x.append(i[0])
            y.append(i[1])

        self.cov = np.cov(x,y)
        #self.cov = np.cov(data[:,0], data[:,1])

        lambdas, vecs = np.linalg.eigh(self.cov)
        order = lambdas.argsort()[::-1]
        lambdas, vecs = lambdas[order], vecs[:,order]

        c = np.sqrt(chi2.ppf(self.p, 2))
        self.w, self.h = 2 * c * np.sqrt(lambdas)
        self.theta = np.degrees(np.arctan(
            ((lambdas[0] - lambdas[1])/self.cov[0,1])))

    def get_params(self):
        return self.means, self.w, self.h, self.theta

    def get_patch(self, line_color="black", face_color="none", alpha=0):
        el = Ellipse(xy=self.means,
                     width=self.w, height=self.h,
                     angle=self.theta, color=line_color, alpha=alpha)
        el.set_facecolor(face_color)
        return el
```

ニューラルネットワーク

```
import numpy as np
class Neural_Network:
    def f(self, x):
        return 1.0 / (1.0 + np.exp(-x))

    def df(self, x):
        return x * (1.0 - x)

    def __init__(self, i, h, o, lr, mu):
        self.w1 = np.random.normal(0.0, 1.0, (h, i))
        self.w2 = np.random.normal(0.0, 1.0, (o, h))
        self.m1 = np.zeros((h, i))
        self.m2 = np.zeros((o, h))
        self.lr = lr
        self.mu = mu

    def test(self, i):
        i_data = i.reshape(-1, 1)
        x = self.f(np.dot(self.w1, i_data))
        y = self.f(np.dot(self.w2, x))
        return x, y

    def train(self, i, t):
        i_data = i.reshape(-1, 1)
        t_data = t.reshape(-1, 1)
        x, y = self.test(i_data)

        d2 = (t_data - y) * self.df(y)
        self.w2 += self.lr * np.dot(d2, x.reshape(1, -1)) + self.mu*self.m2
        self.m2 = self.lr * np.dot(d2, x.reshape(1, -1))

        d1 = np.dot(self.w2.T, d2) * self.df(x)
        self.w1 += self.lr * np.dot(d1, i_data.reshape(1, -1)) + self.mu*self.m1
        self.m1 = self.lr * np.dot(d1, i_data.reshape(1, -1))
```

Fourier 変換

```
import numpy as np
from matplotlib import pyplot as plt
def fourier(t, x, heltz):
    N = len(t)
    F = np.zeros(heltz)
    for w in range(heltz):
        Re = np.sum(x * np.cos(-2 * np.pi * (w+1) * t))
        Im = np.sum(x * np.sin(-2 * np.pi * (w+1) * t))
        F[w] = 1/N * np.sqrt(np.power(Re, 2) + np.power(Im, 2))
    plt.plot(np.arange(heltz) + 1, F)
    plt.xlabel("Frequency(Hz)")
    plt.ylabel("Intensity(1/N)")
    plt.show()
def rfourier(t, x, heltz):
    N = len(t)
    F = np.zeros(heltz)
    for w in range(heltz):
        Re = np.sum(x * np.cos(-2 * np.pi * (w+1) * t))
        Im = np.sum(x * np.sin(-2 * np.pi * (w+1) * t))
        F[w] = 1/N * np.sqrt(np.power(Re, 2) + np.power(Im, 2))
    return F
```