令和 6 年度　修士論文

プログラム読解における
視線運動のクラスタリング
Clustering of Eye Movements
in Program Reading

大阪公立大学大学院
情報学研究科　基幹情報学専攻
学籍番号　BGA23116
明石　拓也

2025 年 2 月 10 日

目次

第 1 部　はじめに

1.1　本研究の背景

情報活用能力が必要となった近年において，初等教育からプログラミングが必修化されるなどプログラミング教育の機会は増加している． 一方で，プログラミングに長けた指導者の不足が問題視されている．そのため，プログラミング学習の支援となるシステムの重要性が高まっている．学習システム開発のためには，プログラミングを理解している人の読解方法の傾向をつかむことが重要となっている．

これまで，ソースコード読解時の視線運動を対象とした検証が行われている．被験者にソースコード読解を必要とするタスクを課し， 読解時の視線運動をアイトラッカーでの計測によりディスプレイ画面上の視線座標という形で取得し，タスクの正誤との関係を調べる分析が主流となっている． これまでタスクに利用されてきたソースコードは，変数代入や四則演算のみで構成されるもの，条件分岐や繰り返し文を含むものであるが， いずれも単一のクラスを用いた手続き型のソースコードで，数行から十数行程度の短いものである．既存の研究では，クラスオブジェクト生成やポリモーフィズムなど， オブジェクト指向の概念を取り入れたソースコードでの検証はなされていない．

1.2　本研究の目的

本研究は，　本論文では，6 つのクラスで構成され，オブジェクト生成やメソッド呼び出しを含む百行以上にわたるソースコードを用いて実験を行い， クラス単位というよりマクロな視点での分析を行う．分析結果の検証のため，被験者ごとの各クラスへの注視時間割合を求め，視線運動の傾向を表すパラメータとする． 傾向の可視化のため，3 次元グラフを用いる．高次元データを 3 次元グラフで可視化するため，主成分分析で次元圧縮をする．

1.3　本論文の構成

本論文では，以下の構成に従って，複数のクラスで構成される Java ソースコードの読解者の視線運動に対する研究成果の報告を行う．

第 2 章では本研究に際し提案した手法の紹介をする．

第 3 章では本研究のため行った実験の準備や手順を説明する．

第 4 章では実験で得られたデータを加工し，分析する過程とその結果を説明する．

最後に，第 5 章で本研究のまとめを行う．

第 2 部　提案手法

2.1　既存の分析手法

本研究では，吉岡らが提案した視線移動のマッピング手法を流用する．吉岡らは，以下の構成で被験者の視線座標をソースコード中の行・列に変換している．

2.2　新たに提案する手法

第 3 部　実験

本章では，実験の概要，実験に際し行った準備と，当日の手順を示す．

3.1　実験の目的

本実験は，以下の目的で行う．
・Java の基礎知識を有する被験者が，複数のクラスで構成され 100 行以上にわたる比較的長大な Java ソースコードを読解する際の視線座標を計測する
・被験者の Java 理解度に応じた視線運動の差異の有無の検証

実験は，被験者 27 名を対象とし，1 つのプログラムに関する 5 つのタスクを課す． アイトラッカーと呼ばれる機器を用いて被験者のモニタ画面上の視線座標を取得する目的で行う．
本章の大まかな流れを以下に示す．

1. 実験準備
   (a) iTrace-Atom プラグインの開発
   (b) 各 PC の環境構築
2. 実験
   (a) 被験者入場
   (b) 解答用紙への氏名記入・同意書記入
   (c) キャリブレーション
   (d) 実験の流れの説明
   (e) 練習問題
   (f) 5 つのタスク
   (g) 被験者退場

3.2　使用するソースコード

ソースコードは Java で記述し，Main クラスを含む計 6 つのクラスで構成されるものを準備する．Main クラス中に 5 つのタスクにまつわる記述を設けている．
以下に，使用するソースコードの全体像を示す．

```java
class Object {
    private String name;

    public void setName(String name) {
        this.name = name;
    }
}

class Animal {
    private String name;

    public void setName(String name) {
        this.name = name;
    }
    public String getName() {
        return this.name;
    }

    public void speak() {
        System.out.println("モーモー！");
    }
}

class Robot {
    private String os;
    private int x;
    private int y;
    private String name;

    public void setName(String name) {
        this.name = name;
    }
    public String getName() {
        return this.name;
    }

    public void setOs(String os) {
        this.os = os;
    }
    public void setNumber(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public void process() {
        int out = x * y;
        System.out.println(this.os + "出力：" + out);
    }
}

class Person extends Animal {
    private int age;

    public void setAge(int age) {
        this.age = age;
    }
    public int getAge() {
        return age;
    }

    public void speak() {
        int futureAge = age + 5;
        System.out.println("人間です！");
    }
}

class Cat extends Animal {
    private int age;

    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }

    public void speak() {
        System.out.println("ニャー！");
    }
}

public class Main {
    public static void main(String[] args) {
//=============================================================================
        //1問目（1分）
        int x = 3;
        if(x > 5){
            x = 0;
            if(x < 2){
                System.out.println("A");
            }
            else{
                System.out.println("B");
            }
        }
        else if(x <= 5){
            x = 10;
            if(x > 5){
                x = 7;
                System.out.println("C");
            }
            else if(x == 7){
                System.out.println("D");
            }
            else{
                System.out.println("E");
            }
        }

//=============================================================================
        //2問目（1分）
        int[] indexes = {51, 3, 4, 6};
        int[] array = {8, 3, 9, 4, 2, 4, 6, 8, 1, 5};

        int sum = 0;
        for(int i = 0; i < indexes.length; i++){
            sum += array[indexes[i]];
        }

        System.out.println(sum);

//=============================================================================
        //3問目(2分)
        Robot robot = new Robot();
        robot.setOs("Windows");
        robot.setNumber(2, 3);
        robot.process();

//=============================================================================
        //4問目(2分)
        Person person = new Person();

        person.setName("Taro");
        person.setAge(25);

//=============================================================================
        //5問目(2分)
        Animal animal1 = new Person();
        Animal animal2 = new Cat();
        Animal animal3 = new Animal();

        animal1.speak();
        animal2.speak();
        animal3.speak();

//=============================================================================
    }
}
```

図 1: 使用したソースコード

3.3 被験者に課すタスク

タスクは全 5 問用意する．うち 2 問がオブジェクト指向の概念を伴わない Main クラス内で完結するタスク，うち 3 問がオブジェクト指向の概念を伴うタスクである． 以下にタスクの内容を示す．

タスク 1:変数代入と if 文による条件分岐を組み合わせたタスク．標準出力の出力結果を答えさせる．

```
//1問目（1分）
int x = 3;
if(x > 5){
    x = 0;
    if(x < 2){
        System.out.println("A");
    }
    else{
        System.out.println("B");
    }
}
else if(x <= 5){
    x = 10;
    if(x > 5){
        x = 7;
        System.out.println("C");
    }
    else if(x == 7){
        System.out.println("D");
    }
    else{
        System.out.println("E");
    }
}
```

図 2: タスク 1 のソースコード

タスク 2:配列の参照を使用したタスク．標準出力の出力結果を答えさせる．

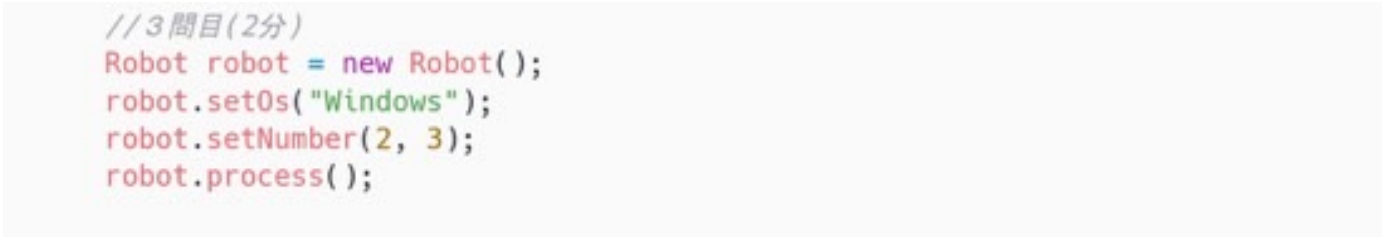
```
//2問目（1分）
int[] indexes = {51, 3, 4, 6};
int[] array = {8, 3, 9, 4, 2, 4, 6, 8, 1, 5};

int sum = 0;
for(int i = 0; i < indexes.length; i++){
    sum += array[indexes[i]];
}

System.out.println(sum);
```

図 3: タスク 2 のソースコード

タスク 3:1 種類のクラスを使用し，オブジェクト生成と外部からのメソッド実行を含むタスク．標準出力の出力結果を答えさせる．



```
//3問目(2分)
Robot robot = new Robot();
robot.setOs("Windows");
robot.setNumber(2, 3);
robot.process();
```

図 4: タスク 3 のソースコード

タスク 4:2 種類のクラスを使用し，オブジェクト生成とゲッター・セッターの呼び出しを含むタスク．
2 種類のクラスは片方がもう片方を継承する関係にある．ソースコード内で，それぞれのオブジェクトのセッターが定義された行を答えさせる．

```
//4問目(2分)
Person person = new Person();

person.setName("Taro");
person.setAge(25);
```

図 5: タスク 4 のソースコード

タスク 5:3 種類のクラスを使用し，メソッドのオーバーライドを含む．標準出力の出力結果を答えさせる．

```
//5問目(2分)
Animal animal1 = new Person();
Animal animal2 = new Cat();
Animal animal3 = new Animal();

animal1.speak();
animal2.speak();
animal3.speak();
```

図 6: タスク 5 のソースコード

3.4 実験に使用する物品

実験に際して，以下の物が必要となる．

1. 被験者の視線座標取得に用いるアイトラッカー (Tobii Pro Spark)
2. アイトラッカーの設定・キャリブレーションに用いるソフトウェア (Tobii Pro Eye Tracker Manager)

3. アイトラッカーの処理を担当するソフトウェア (iTrace)
4. タスクに使用するソースコードを表示するエディタ (Atom)
5. アイトラッカーで取得した画面上での座標を，ソースコードの行・列に変換する為のプラグイン
6. ソースコードとタスク

以下，これらの物を解説する．

3.4.1 アイトラッカー

視線運動をディスプレイ画面上の時系列座標データとして取得するため，アイトラッカーと呼ばれる装置を使用する. 本研究では,3 台のディスプレイで同時に測定するため，計 3 台のアイトラッカーを準備する． 内訳として，Tobii Pro Spark[2]2 台と Tobii Pro Nano[4]1 台を使用する．
Tobii Pro Spark，Tobii Pro Nano ともにディスプレイ装置の下部に固定することで, ディスプレイ画面上における被験者の注視点の座標を計測できる．サンプリングレートはいずれも 60Hz である. 測定の際, 各被験者ごとにキャリブレーションする．

3.4.2 Tobii Pro Eye Tracker Manager

アイトラッカーのパラメータ設定，キャリブレーションには Tobii 公式で提供されている Tobii Pro Eye Tracker Manager というソフトウェアを使用する． 本ソフトウェアでは,

3.4.3 アイトラッカーの処理を担当するソフトウェア

アイトラッカーの測定開始・終了処理や，出力データの成型などを担当するソフトウェアとして iTrace[5] を使用する．

3.4.4 タスクに使用するソースコードを表示するエディタ (Atom)

エディタは，iTrace プラグインとの互換性のため，Atom[6] を使用する．Atom は,

3.4.5 iTrace-Atom プラグイン

iTrace の出力座標データとエディタの画面表示情報を結びつけ，座標からプログラム中の行・列に対応させるため，Atom エディタのプラグインが開発されている．
本研究では，Github 上で公開されている iTrace-Atom プラグインに機能を追加したものを用いる．

3.4.6　ディスプレイ

3.4.7　コンピュータ

3.5　実験準備

実験で使用する

3.6　実験手順

3.7　実験

本章では，実験の流れを示す．本実験により，データは問 1〜問 5 で　人分得られた．

近畿大学工業高等専門学校情報科 4 年生の学生 27 名を対象として行う．対象者全員がオブジェクト指向を含む Java プログラミングの授業を受講した経験がある．

実験では，オブジェクト指向の概念を含まない問題 2 問，含む問題 3 問の計 5 つの問題に取り組んでもらった．なお，対象者全員に計測するデータを研究以外の用途に使用しない旨を伝え，承諾書にサインしてもらった． 3 台の実験台を用い，3 人ずつ同時並行で測定する．実験の流れを以下に示す．

承諾書にサインしてもらう解答用紙に，出席番号と名前を書いてもらう 3 台の実験台それぞれに座ってもらい，キャリブレーションを行う． 実験の流れの説明． ソースコードを読んで問題を解くこと，その際の視線データを計測すること，スクロール操作のみ行い，コードへの書き込みを行わないこと，回答がわかり次第挙手で合図し，解答用紙に書き込むこと，

図 7: 実験部屋の様子

第 4 部　データの分析

本章では，実験で得られたデータを加工し，分析する一連の流れを示す．

4.1　はじめに

実験により得られた生データは xml 形式で保存される．以下に，生データが持つパラメータを記す．

表 1: 生データに含まれるデータ一覧

パラメータ名	値の意味	値の例
screen_width	ディスプレイ画面の横 px 数	1920
screen_height	ディスプレイ画面の縦 px 数	1200
plugin_type	プラグインの種類	ATOM
gaze	視線情報	後述

gaze パラメータはアイトラッカーのサンプリング数だけ存在し，それぞれ以下のパラメータを持つ．

表 2: 生データに含まれる視線情報

パラメータ名	値の意味	値の例
event_id	2	133784518694431017
plugin_time	サンプリングの UNIX 時間 [ms]	1733978269442
x	ディスプレイ画面上の x 座標 [px]	480
y	ディスプレイ画面上の y 座標 [px]	418
source_file_line	ソースコード中の行	2
source_file_col	ソースコード中の列	9
word	視線が位置していた単語	int
gaze_target	視線が位置していたファイルの名前	Problem1.java
gaze_target_type	対象ファイルの拡張子	java
source_file_path	対象ファイルのファイルパス	Problem1.java
editor_line_height	エディタの行の高さの設定値	40
editor_font_height	エディタのフォントサイズの設定値	120

上述の生データから特定の情報を抜き出し，プログラムで扱いやすいよう csv に加工する． csv から，クラスごとの注視時間割合を算出注視時間割合を主成分分析で 3 次元まで落とし，正誤で色分けして 3D プロットで可視化可視化したデータを観察する．

4.2 3D プロットの観察

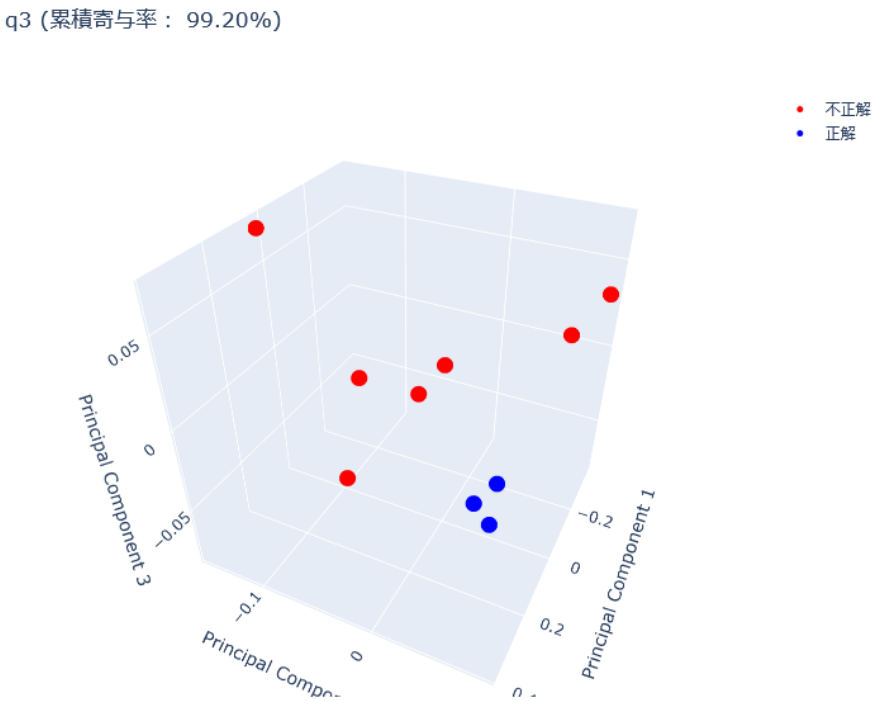


図 8: タスク 3 の分布

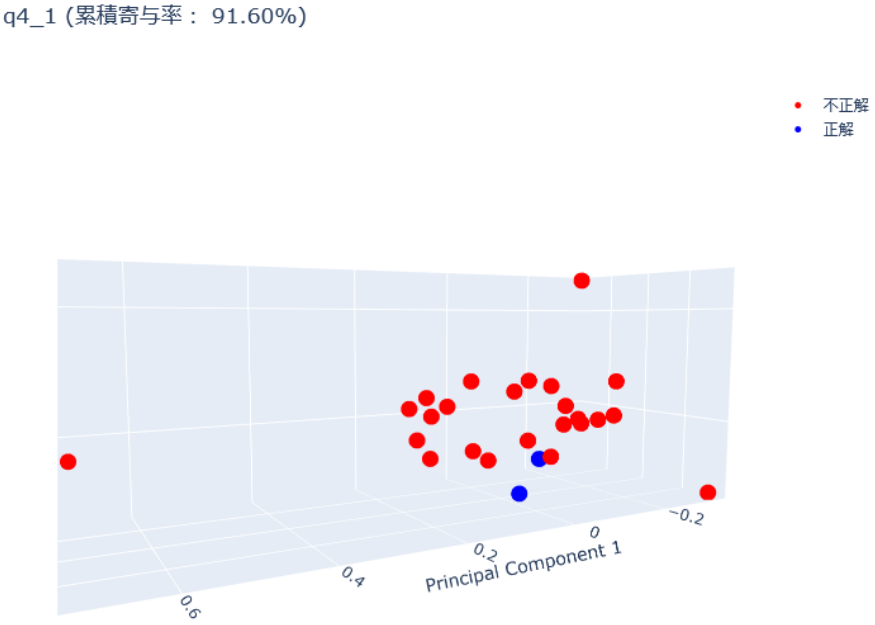


図 9: タスク 4-1 の分布

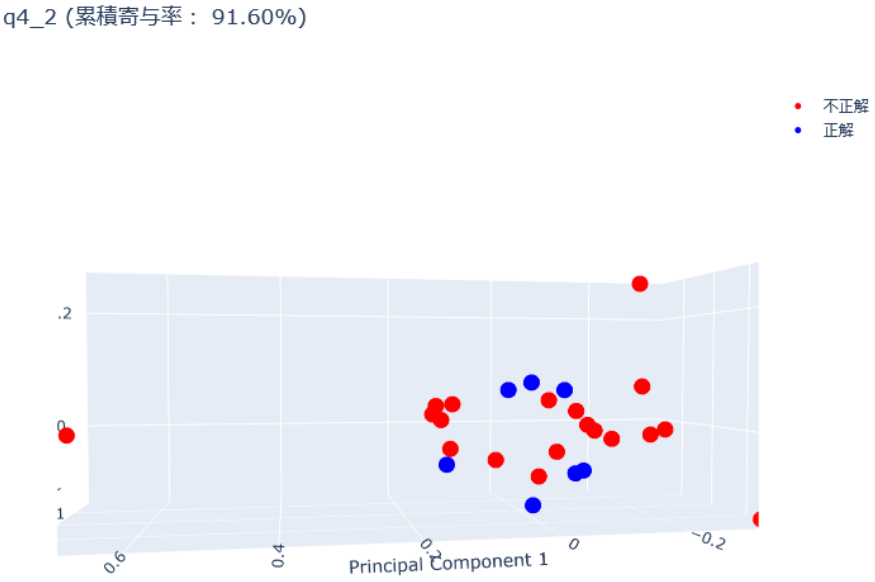


図 10: タスク 4-2 の分布

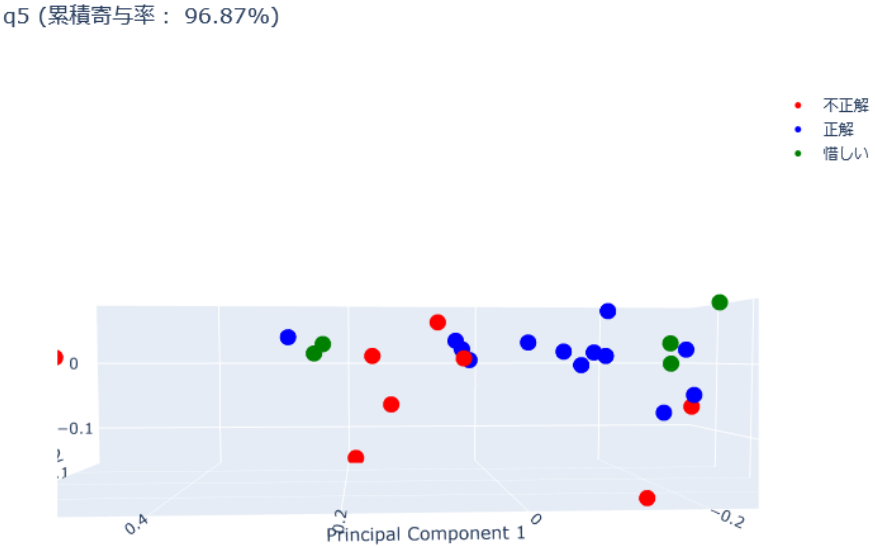


図 11: タスク 5 の分布

第 5 部　結論

参考文献

[1] 構文木と視線移動の自動マッピング手法を用いたプログラム理解過程の分析, 吉岡春彦, 上野秀剛,2023

[2] tobii pro spark https://www.tobii.com/ja/products/eye-trackers/screen-based/tobii-pro-spark

[3] Tobii Pro Eye Tracker Manager https://connect.tobii.com/s/etm-download

[4] https://www.tobii.com/ja/products/discontinued/tobii-pro-nano

[5] iTrace https://www.i-trace.org/

[6] ATOM https://atom-editor.cc/