

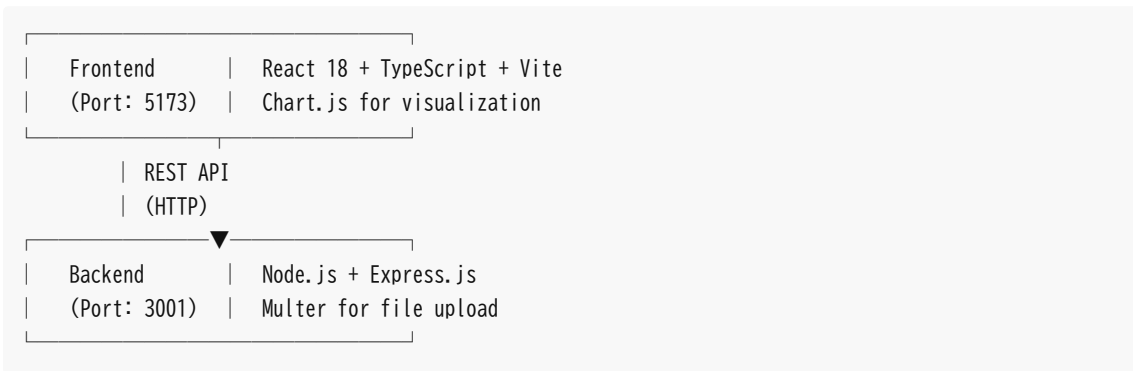
# 軌道復元システム - 技術仕様書

## システム概要

鉄道軌道の測定データをCSV形式で受け取り、可視化・統計分析・波形復元処理を行うWebアプリケーション。VB6で開発された既存システムのモダン化実装。

## システム構成

### アーキテクチャ



### 技術スタック

#### フロントエンド

- フレームワーク: React 18.2.0
- 言語: TypeScript 5.3.3
- ビルドツール: Vite 5.0.8
- グラフィブラリ: Chart.js 4.4.0 + react-chartjs-2 5.2.0
- スタイリング: CSS Modules

#### バックエンド

- ランタイム: Node.js (ES Modules)
- フレームワーク: Express.js 4.18.2
- ファイルアップロード: Multer 1.4.5-lts.1
- CORS: cors 2.8.5

## ディレクトリ構造

```
rail-track-app/  
├── backend/  
│   ├── src/  
│   │   └── server.js           # Expressサーバーメインファイル  
│   ├── uploads/               # 一時ファイルストレージ(処理後削除)  
│   ├── package.json  
│   └── node_modules/  
├── frontend/  
│   └── src/
```

```
| | | └── main.tsx # エントリーポイント
| | | └── App.tsx # メインアプリケーション
| | | └── components/
| | | | ├── FileUpload.tsx # ファイルアップロードコンポーネント
| | | | ├── ChartDisplay.tsx # グラフ表示コンポーネント
| | | | └── Statistics.tsx # 統計情報表示コンポーネント
| | | └── App.css
| | └── index.css
| └── index.html
| └── package.json
| └── tsconfig.json
| └── vite.config.ts
| └── node_modules/
└── sample-data.csv # サンプルデータファイル
```

## バックエンドAPI仕様

## エンドポイント一覧

## 1. ヘルスチェック

サーバーの稼働状態を確認

```
GET /api/health
```

Response:

```
{
  "status": "ok",
  "message": "Rail Track API is running"
}
```

## 2. CSVファイルアップロード

## 軌道測定データのアップロードと解析

```
POST /api/upload
Content-Type: multipart/form-data
```

**Request Body:**

- file : CSVファイル(multipart/form-data)

Response (成功時):

```
{
  "success": true,
  "filename": "sample-data.csv",
  "dataPoints": 41,
```

```
"data": [
  { "distance": 0.0, "irregularity": 2.5 },
  { "distance": 0.5, "irregularity": 2.8 },
  ...
],
"statistics": {
  "min": 2.2,
  "max": 6.7,
  "avg": 4.15,
  "stdDev": 1.42
}
}
```

Response (エラー時):

```
{
  "error": "ファイルがアップロードされていません"
}
```

実装箇所: [server.js:109-140](#)

---

### 3. 相関係数計算

2つのデータセット間の相関係数を算出

```
POST /api/calculate-correlation
Content-Type: application/json
```

Request Body:

```
{
  "data1": [
    { "distance": 0.0, "irregularity": 2.5 },
    ...
  ],
  "data2": [
    { "distance": 0.0, "irregularity": 2.3 },
    ...
  ]
}
```

Response:

```
{
  "success": true,
  "correlation": 0.85,
  "description": "強い正の相関"
}
```

相関の評価基準:

- $> 0.7$  : 強い正の相関
- $0.3 \sim 0.7$  : 中程度の正の相関
- $-0.3 \sim 0.3$  : 弱い相関
- $-0.7 \sim -0.3$  : 中程度の負の相関
- $< -0.7$  : 強い負の相関

実装箇所: [server.js:143-166](#)

---

#### 4. ピーク検出

軌道狂い量の極大値・極小値を検出

```
POST /api/detect-peaks
Content-Type: application/json
```

Request Body:

```
{
  "data": [
    { "distance": 0.0, "irregularity": 2.5 },
    ...
  ]
}
```

Response:

```
{
  "success": true,
  "maxima": [
    { "index": 5, "distance": 2.5, "value": 6.7, "type": "maximum" }
  ],
  "minima": [
    { "index": 2, "distance": 1.0, "value": 2.2, "type": "minimum" }
  ]
}
```

---

#### 5. 異常値検出

統計的手法による異常値の検出

```
POST /api/detect-outliers
Content-Type: application/json
```

Request Body:

```
{
  "data": [...],
  "sigmaMul": 2.0
}
```

**Parameters:**

- `sigmaMul` : 標準偏差の倍率 (デフォルト: 2.0、範囲: 1.0~5.0)

**Response:**

```
{
  "success": true,
  "outliers": [
    {
      "index": 10,
      "distance": 5.0,
      "value": 8.5,
      "deviation": 3.2,
      "type": "high"
    }
  ],
  "threshold": { "lower": 1.5, "upper": 7.0 }
}
```

---

## 6. FFTスペクトル解析

高速フーリエ変換による周波数解析

```
POST /api/analyze-spectrum
Content-Type: application/json
```

**Request Body:**

```
{
  "data": [...]
}
```

**Response:**

```
{
  "success": true,
  "spectrum": {
    "frequencies": [0.0, 0.1, 0.2, ...],
    "power": [12.5, 8.3, 5.2, ...]
  },
  "dominantFrequencies": [
    { "frequency": 0.15, "power": 8.3, "wavelength": 6.67 }
  ]
}
```

---

## 7. カント・スラック補正

曲線部の傾斜とレール間隔の補正

```
POST /api/apply-corrections
Content-Type: application/json
```

**Request Body:**

```
{
  "data": [...],
  "coefficients": {
    "cant": 1.0,
    "slack": 0.5
  }
}
```

**Response:**

```
{
  "success": true,
  "correctedData": [...],
  "statistics": {
    "before": { "min": 2.2, "max": 6.7, ... },
    "after": { "min": 2.0, "max": 6.2, ... }
  }
}
```

---

## 8. MTT軌道評価

BC値・CD値による軌道品質評価

```
POST /api/calculate-mtt
Content-Type: application/json
```

**Request Body:**

```
{
  "data": [...]
}
```

**Response:**

```
{
  "success": true,
  "mtt": {
    "bc": 3.45,
    "cd": 2.12,
    "evaluation": "良好"
  }
}
```

---

## 9. 波形復元処理

移動平均フィルタによるノイズ除去と波形平滑化

```
POST /api/restore-waveform
Content-Type: application/json
```

Request Body:

```
{
  "data": [
    { "distance": 0.0, "irregularity": 2.5 },
    ...
  ],
  "filterType": "simple"
}
```

Parameters:

- filterType : フィルタータイプ
  - "simple" : 3点移動平均フィルタ(デフォルト)

Response:

```
{
  "success": true,
  "original": {
    "data": [...],
    "statistics": { "min": 2.2, "max": 6.7, "avg": 4.15, "stdDev": 1.42 }
  },
  "restored": {
    "data": [...],
    "statistics": { "min": 2.3, "max": 6.5, "avg": 4.14, "stdDev": 1.35 }
  },
  "filterType": "simple"
}
```

実装箇所: [server.js:169-210](#)

---

## データ処理アルゴリズム

### 1. CSVパース処理

実装: [server.js:41-57](#)

```
function parseCSV(csvText) {
  const lines = csvText.split('\n').filter(line => line.trim());
  const data = [];

  for (const line of lines) {
```

```

const values = line.split(',').map(v => v.trim());
if (values.length >= 2) {
  const distance = parseFloat(values[0]);
  const irregularity = parseFloat(values[1]);
  if (!isNaN(distance) && !isNaN(irregularity)) {
    data.push({ distance, irregularity });
  }
}
}

return data;
}

```

処理内容:

- 改行で分割し、空行を除外
- カンマで列を分割
- 数値変換(parseFloat)
- 無効なデータ行をスキップ

## 2. 相関係数計算(ピアソンの積率相関係数)

実装: [server.js:60-84](#)

計算式:

$$r = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{[\sum (x_i - \bar{x})^2 \times \sum (y_i - \bar{y})^2]}}$$

または

$$r = \frac{(n \times \sum (x_i \times y_i) - \sum x_i \times \sum y_i)}{\sqrt{[(n \times \sum x_i^2 - (\sum x_i)^2) \times (n \times \sum y_i^2 - (\sum y_i)^2)]}}$$

実装ロジック:

```

function calculateCorrelation(data1, data2) {
  const n = Math.min(data1.length, data2.length);
  if (n === 0) return 0;

  let sumX = 0, sumY = 0, sumXX = 0, sumYY = 0, sumXY = 0;

  for (let i = 0; i < n; i++) {
    const x = data1[i].irregularity;
    const y = data2[i].irregularity;
    sumX += x;
    sumY += y;
    sumXX += x * x;
    sumYY += y * y;
    sumXY += x * y;
  }

  const avgX = sumX / n;
  const avgY = sumY / n;

```



```

const numerator = sumXY - n * avgX * avgY;
const denominator = Math.sqrt((sumXX - n * avgX * avgX) * (sumYY - n * avgY * avgY));

if (denominator === 0) return 0;
return numerator / denominator;
}

```

### 3. 統計計算

実装: [server.js:87-101](#)

計算項目:

- 最小値 (min): `Math.min(...values)`
- 最大値 (max): `Math.max(...values)`
- 平均値 (avg):  $\sum x_i / n$
- 標準偏差 (stdDev):  $\sqrt{(\sum (x_i - \bar{x})^2 / n)}$  (母集団標準偏差)

```

function calculateStatistics(data) {
  if (data.length === 0) {
    return { min: 0, max: 0, avg: 0, stdDev: 0 };
  }

  const values = data.map(d => d.irregularity);
  const min = Math.min(...values);
  const max = Math.max(...values);
  const avg = values.reduce((a, b) => a + b, 0) / values.length;

  const variance = values.reduce((sum, val) =>
    sum + Math.pow(val - avg, 2), 0) / values.length;
  const stdDev = Math.sqrt(variance);

  return { min, max, avg, stdDev };
}

```

### 4. 波形復元フィルタ

実装: [server.js:169-210](#)

Simple Moving Average Filter (3点移動平均)

計算式:

$$y'[i] = (y[i-1] + y[i] + y[i+1]) / 3$$

ただし、端点( $i=0$ ,  $i=n-1$ )は元の値を維持

実装コード:

```

if (filterType === 'simple') {
  restoredData = data.map((point, i) => {
    if (i === 0 || i === data.length - 1) {
      return point; // 端点はそのまま
    }
    const avgIrregularity = (
      data[i-1].irregularity +
      point.irregularity +
      data[i+1].irregularity
    ) / 3;
    return { ...point, irregularity: avgIrregularity };
  });
}

```

効果:

- 高周波ノイズの除去
- 波形の平滑化
- 測定誤差の影響を軽減

拡張性:

- `filterType` パラメータで異なるフィルタを追加可能
- 例: "gaussian", "median", "savitzky-golay" など

## フロントエンド仕様

### データ型定義

TypeScript Interfaces: [App.tsx:7-21](#)

```

export interface TrackData {
  distance: number // 距離 (m)
  irregularity: number // 軌道狂い量 (mm)
}

export interface DataSet {
  data: TrackData[] // データ配列
  statistics: {
    min: number // 最小値
    max: number // 最大値
    avg: number // 平均値
    stdDev: number // 標準偏差
  }
  filename?: string // ファイル名(オプション)
}

```

### 状態管理

React State: [App.tsx:24-26](#)

```
const [originalData, setOriginalData] = useState<DataSet | null>(null)
const [restoredData, setRestoredData] = useState<DataSet | null>(null)
const [loading, setLoading] = useState(false)
```

状態遷移:

1. 初期状態: `originalData = null` , `restoredData = null`
2. アップロード後: `originalData = {...}` , `restoredData = null`
3. 復元処理後: `originalData = {...}` , `restoredData = {...}`
4. 再アップロード: `restoredData` をクリア

---

## 主要コンポーネント

### 1. FileUpload

ファイルアップロードUI

Props:

- `onFileUpload: (file: File) => void` - ファイル選択時のコールバック
- `loading: boolean` - ローディング状態

### 2. ChartDisplay

Chart.jsによるグラフ表示

Props:

- `originalData: TrackData[]` - 元データ
- `restoredData?: TrackData[]` - 復元データ(オプション)

グラフ設定:

- タイプ: Line Chart
- X軸: 距離 (m)
- Y軸: 軌道狂い量 (mm)
- 色: 元データ(青), 復元データ(赤)

### 3. Statistics

統計情報表示カード

Props:

- `title: string` - カードタイトル
- `statistics: { min, max, avg, stdDev }` - 統計値

### 4. SpectrumAnalysis

FFTスペクトル解析コンポーネント

Props:

- `data: TrackData[]` - 解析対象データ

機能:

- スペクトル解析の実行

- パワースペクトルグラフ表示
- 支配的周波数の表示

## 5. CorrectionSettings

カント・スラック補正設定コンポーネント

Props:

- `data: TrackData[]` - 補正対象データ
- `onCorrectionApplied: (data: TrackData[]) => void` - 補正完了コールバック

機能:

- 補正係数の入力フォーム
- 補正前後の比較グラフ
- 補正効果の統計表示

## 6. OutlierDetection

異常値検出コンポーネント

Props:

- `data: TrackData[]` - 検査対象データ
- `onOutliersDetected: (outliers: Outlier[]) => void` - 検出完了コールバック

機能:

- シグマ倍率スライダー (1.0~5.0)
- 異常値テーブル表示
- 重症度別の色分け表示

## 7. FFTFilterSettings

FFTフィルター詳細設定コンポーネント

Props:

- `onSettingsChange: (settings: FFTSettings) => void` - 設定変更コールバック
- `disabled: boolean` - 無効化フラグ

機能:

- フィルタータイプ選択
- カットオフ周波数スライダー
- 帯域幅設定 (bandpass時)

---

## API通信処理

ファイルアップロード

実装: [App.tsx:28-57](#)

```
const handleFileUpload = async (file: File) => {
  setLoading(true)
  const formData = new FormData()
  formData.append('file', file)
```

```

try {
  const response = await fetch('/api/upload', {
    method: 'POST',
    body: formData,
  })

  const result = await response.json()

  if (result.success) {
    setOriginalData({
      data: result.data,
      statistics: result.statistics,
      filename: result.filename
    })
    setRestoredData(null)
  } else {
    alert('エラー: ' + (result.error || '不明なエラー'))
  }
} catch (error) {
  console.error('Upload error:', error)
  alert('アップロードエラーが発生しました')
} finally {
  setLoading(false)
}
}

```

## 波形復元処理

実装: [App.tsx:59-92](#)

```

const handleRestoreWaveform = async () => {
  if (!originalData) return

  setLoading(true)
  try {
    const response = await fetch('/api/restore-waveform', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({
        data: originalData.data,
        filterType: 'simple'
      }),
    })
  }

  const result = await response.json()

  if (result.success) {
    setRestoredData({
      data: result.restored.data,
      statistics: result.restored.statistics,

```

```
        filename: originalData.filename + ' (復元)'
      })
    } else {
      alert('エラー: ' + (result.error || '不明なエラー'))
    }
  } catch (error) {
    console.error('Restore error:', error)
    alert('波形復元エラーが発生しました')
  } finally {
    setLoading(false)
  }
}
```

## セキュリティ考慮事項

### ファイルアップロード

- MIMEタイプ検証: text/csv のみ許可
- 拡張子検証: .csv のみ許可
- ファイルサイズ制限: Multerのデフォルト設定に依存
- 一時ファイル削除: 処理完了後即座に削除

実装: [server.js:29-38](#)

```
const upload = multer({
  storage: storage,
  fileFilter: (req, file, cb) => {
    if (file.mimetype === 'text/csv' || file.originalname.endsWith('.csv')) {
      cb(null, true);
    } else {
      cb(new Error('CSVファイルのみアップロード可能です'));
    }
  }
});
```

### CORS設定

- すべてのオリジンを許可(開発環境)
- 本番環境では特定オリジンのみ許可するよう変更推奨

```
app.use(cors()); // 開発環境
// 本番環境推奨設定:
// app.use(cors({ origin: 'https://your-domain.com' }));
```

## パフォーマンス考慮事項

### データ量の制限

- 現状: データ点数に制限なし
- 推奨: 大規模データ(10,000点以上)の場合:

- サーバサイドでページネーション実装
- フロントエンドでデータ間引き表示
- Web Workerで非同期処理

## 処理の同期性

- 現状: すべて同期処理
- 改善案:
  - 重い計算処理を非同期化
  - ストリーム処理でメモリ効率向上

---

## 拡張性

### フィルタアルゴリズムの追加

現在のフィルタ:

- `simple`: 3点移動平均

追加可能なフィルタ例:

```
// Gaussian filter
if (filterType === 'gaussian') {
  const kernel = [0.27, 0.46, 0.27]; // ガウシアンカーネル
  // 実装...
}

// Median filter
if (filterType === 'median') {
  restoredData = data.map((point, i) => {
    const window = data.slice(Math.max(0, i-1), i+2);
    const sorted = window.map(p => p.irregularity).sort((a, b) => a - b);
    const median = sorted[Math.floor(sorted.length / 2)];
    return { ...point, irregularity: median };
  });
}
```

### 新規エンドポイントの追加

例: データエクスポート機能

```
app.get('/api/export/:format', (req, res) => {
  const { format } = req.params;
  // CSV, JSON, Excel等への変換処理
});
```

---

## 開発・デバッグ情報

### ログ出力

サーバ起動時:

Rail Track API server is running on `http://localhost:3001`  
Health check: `http://localhost:3001/api/health`

#### エラーログ:

- `console.error` でスタックトレースを出力
- クライアントには簡潔なエラーメッセージを返す

#### デバッグ用エンドポイント

```
# ヘルスチェック
curl http://localhost:3001/api/health

# アップロードテスト
curl -X POST -F "file=@sample-data.csv" http://localhost:3001/api/upload
```

## 依存パッケージ詳細

#### バックエンド (backend/package.json)

```
{
  "dependencies": {
    "express": "^4.18.2",      // Webフレームワーク
    "cors": "^2.8.5",         // CORS対応
    "multer": "^1.4.5-lts.1"  // ファイルアップロード
  }
}
```

#### フロントエンド (frontend/package.json)

```
{
  "dependencies": {
    "react": "^18.2.0",      // UIライブラリ
    "react-dom": "^18.2.0",  // React DOM操作
    "chart.js": "^4.4.0",     // グラフ描画
    "react-chartjs-2": "^5.2.0" // Chart.js Reactラッパー
  },
  "devDependencies": {
    "@types/react": "^18.2.43", // React型定義
    "@types/react-dom": "^18.2.17", // React DOM型定義
    "@vitejs/plugin-react": "^4.2.1", // Vite Reactプラグイン
    "typescript": "^5.3.3", // TypeScriptコンパイラ
    "vite": "^5.0.8" // ビルドツール
  }
}
```

## 既知の制限事項



1. **ファイルサイズ**: 大容量CSVファイル(>50MB)は処理が遅延する可能性
  2. **ブラウザ互換性**: モダンブラウザ(Chrome, Firefox, Edge, Safari最新版)を推奨
  3. **同時接続数**: Express.jsのデフォルト設定に依存
  4. **データ永続化**: データベース未実装(すべてメモリ上で処理)
  5. **認証・認可**: 未実装(必要に応じて追加)
- 

## 将来の拡張計画

- ☒ ピーク検出機能 ☒ v2.0で実装済み
- ☒ 異常値検出機能 ☒ v2.0で実装済み
- ☒ FFTスペクトル解析 ☒ v2.0で実装済み
- ☒ カント・スラック補正 ☒ v2.0で実装済み
- ☒ MTT軌道評価 ☒ v2.0で実装済み
- ☒ 高度なFFTフィルター ☒ v2.0で実装済み
- ☐ 複数ファイルの一括処理
- ☐ データベース統合(履歴管理)
- ☐ PDFレポート出力機能
- ☐ ユーザー認証機能
- ☐ リアルタイムデータストリーミング対応
- ☐ DCP/LABOCSフォーマット対応
- ☐ Oracle DB連携