QTc解析アプリ (qtc.html) 仕様書

> ver 1.5 : 2026/3/5 update 本仕様書はソースコード (`qtc.html`) の実装を正確に記述したものである。

1. アプリケーション概要

心電図（ECG）画像から QT 間隔を自動計測し、QTc（心拍補正 QT）を算出するブラウザベースの単一 HTML アプリケーションである。標準12誘導心電図（赤方眼紙上の黒波形）とホルター心電図（カラー波形）の両方に対応する。

1.1 基本仕様

項目	値
紙送り速度	25 mm/sec（固定）
最大処理幅	2000 px（超過時は自動リサイズ）
画像入力数	最大2枚（画像1: II誘導、画像2: V5誘導）
QTc補正式	Bazett (QTcB) および Fridericia (QTcF)
解析履歴	localStorage に最大20件保存
対応ブラウザ	モダンブラウザ（Canvas API, localStorage 使用）

1.2 QTc算出式

```
QTcB = QT / √(RR)     ... Bazett補正
QTcF = QT / ∛(RR)     ... Fridericia補正
```

ここで QT は ms 単位、RR は秒単位である。

2. キャリブレーション（自動計測計算方法）

2.1 概要

キャリブレーションとは、画像上のピクセル数と実際の時間（秒）の対応関係を決定する処理である。心電図の紙送り速度は 25 mm/sec であるため、画像上で「25 mm が何ピクセルに相当するか」を求めることが目標となる。最終的に `pxPerSec`（1秒あたりのピクセル数）として表現される。

2.2 プリセットシステム

3つのキャリブレーションプリセットが用意されている。

プリセット名	デフォルト値	用途
100%	94 px/sec	標準12誘導心電図（等倍表示）
200%	191 px/sec	標準12誘導心電図（2倍表示）
ホルター	126 px/sec	ホルター心電図

- プリセット値は `localStorage`（キー: `qtc_cal_presets`）に JSON 形式で保存される。
- アプリ起動時に `loadPresets()` が呼ばれ、保存済みの値が復元される。保存値がない場合はデフォルト値が使用される。
- ユーザーが「再設定」ボタンを押すと、現在の画像で使用中の `pxPerSec` 値が選択したプリセットに上書き保存される（`saveCurrentAsPreset()`）。
- 保存されたプリセット値は同一ブラウザ・同一オリジンであれば永続的に保持され、次回起動時に自動的にその数値が設定値として採用される。

2.3 自動キャリブレーション計測アルゴリズム (detectCalibration)

画像内の赤い方眼グリッド線の間隔を検出し、ピクセル/秒を算出する。

Step 1: 赤色列プロファイルの生成

画像の縦方向中央 40%（上端30%〜下端70%）の領域を走査し、各列（x座標）の「赤スコア」を算出する。

```
走査範囲:
  startY = height × 0.3
  endY   = height × 0.7
  yステップ = 2（1行おきにサンプリング）

赤ピクセル判定条件:
  R > 130 かつ G < 200 かつ B < 200 かつ R > G + 10

各列の赤スコア = 該当ピクセル数の合計
```

結果として幅 `width` の `colProfile` 配列（Float32Array）が生成される。

Step 2: 赤グリッド不足の判定

```
maxRedScore = colProfile の最大値
maxRedScore < 3 の場合 → 検出失敗（赤グリッド不足）
```

Step 3: ピーク検出

赤スコアの列プロファイルからローカルピーク（グリッド線の中心位置）を検出する。

```
閾値 = maxRedScore × 0.3

ピーク条件:
  colProfile[x] > 閾値
  colProfile[x] > colProfile[x-1]（左隣より大きい）
  colProfile[x] > colProfile[x+1]（右隣より大きい）
  前回ピークとの距離 > 3 px（重複防止）

ピーク数 < 2 の場合 → 検出失敗（ピーク不足）
```

Step 4: グリッド間隔の最頻値（モード）算出

隣接ピーク間の距離を算出し、2px単位に丸めてヒストグラムを生成する。

```
各ピーク間距離 d:
  rounded = round(d / 2) × 2   （2px単位に丸め）
  rounded > 2 のもののみカウント

modeDist = 最も頻度が高い丸め距離
modeDist = 0 の場合 → 検出失敗
```

Step 5: スケール変換（1mm / 5mm / 25mm の自動判別）

検出された `modeDist` が心電図グリッドのどのスケールに対応するかを自動判別する。

```
modeDist < 16 px  → 1mm 間隔と判定  → pxPer5mm = modeDist × 5
16 ≤ modeDist ≤ 50 → 5mm 間隔と判定  → pxPer5mm = modeDist（そのまま）
modeDist > 50 px  → 25mm 間隔と判定 → pxPer5mm = modeDist / 5

pxPerSec = round(pxPer5mm × 5)
```

心電図の紙送り速度 25 mm/sec において、5mm = 0.2秒 なので `pxPer5mm × 5` で1秒あたりのピクセル数に変換される。

2.4 プリセット自動選択（ver 1.5 新仕様）

自動キャリブレーション計測の後、検出値を直接使用するのではなく、3つのプリセットとの差分を比較して最も近いプリセット値にスナップする。

検出成功時の処理フロー

```
1. detectCalibration() で自動検出値 (detectedPx) を取得

2. 各プリセットとの差分を計算:
   diff_100    = |calPresets['100']   - detectedPx|
   diff_holter = |calPresets['holter'] - detectedPx|
   diff_200    = |calPresets['200']   - detectedPx|

3. 最小差分のプリセットを選択 (findClosestPreset):
   matchedPreset = argmin(diff_100, diff_holter, diff_200)

4. プリセット値を採用:
```

```
   pxPerSec = calPresets[matchedPreset]  ← 検出値ではなくプリセット値

5. 抽出モード決定:
   matchedPreset === 'holter' → extractionMode = 'holter'
   それ以外                   → extractionMode = 'standard'
```

画面には「自動検出: XXpx/s → 100%(YYpx/s)を適用 (差分: Zpx)」のように表示される。

検出失敗時のフォールバック

```
pxPerSec === 0（検出失敗）の場合:
  デフォルト値 94 px/sec に最も近いプリセットを選択
  （通常は 100% プリセットが選択される）
```

2.5 2枚目画像のキャリブレーション

画像2（V5誘導）が読み込まれた際、画像1（II誘導）が既に解析済みであれば、画像1と同一のキャリブレーション値（`pxPerSec`, `calMethod`, `extractionMode`）がそのまま引き継がれる。画像2に対しては`detectCalibration()`は実行されない。

2.6 手動キャリブレーション修正

キャンバス上に表示されるH型バー（オレンジ色）をドラッグすることで、手動でキャリブレーション値を調整できる。

```
H型バー:
  左端ハンドル: 基準位置の移動（表示位置のみ変更、pxPerSec は不変）
  右端ハンドル: pxPerSec を動的に変更
    newPxPerSec = max(20, round(マウスX座標 - calX))

手動修正時:
  calMethod = 'drag' に設定
  もう一方の画像にも同じ pxPerSec が同期される
(syncCalibrationToOther)
```

3. 心電図波形の認識方法

3.1 抽出モードの分岐

画像から1次元信号（各x座標における波形のy座標）を抽出する処理は、`extractionMode`に応じて2つのアルゴリズムに分岐する。

```
extractionMode === 'standard' → extractSignalStandard()
... 赤方眼紙上の黒波形
extractionMode === 'holter'   → extractSignalHolter()
... カラー波形（ホルター）
```

3.2 Standard モード: 赤方眼紙上の黒波形抽出 (extractSignalStandard)

標準12誘導心電図を想定したアルゴリズムで、赤い方眼紙上に印刷された黒い波形を検出する。

Pass 1: 各列の最小輝度の算出

```
走査範囲: y = 1 〜 height - 1（上下1pxをマージン）

各列(x)について:
  全ピクセルを走査し、以下の条件でフィルタリング:
    赤ピクセル除外条件: R > 100 かつ R > G + 30 かつ R > B + 30
    → この条件に該当するピクセル（赤方眼線）はスキップ

  フィルタ後のピクセルの輝度を算出:
    lum = 0.299 × R + 0.587 × G + 0.114 × B

  darkestLum = その列における最小輝度
```

Pass 2: 最暗ピクセルの中から最も極端な位置を選択

```
darkestLum < 250 の場合（信号が存在する列）:
  lumThresh = darkestLum + 30（許容範囲）
  centerY = height / 2（画像中央）

  輝度が lumThresh 以下のピクセルの中から:
    画像中央（centerY）から最も遠い y 座標を選択

  signal[x] = height - bestY（y軸を反転して上が正）
```

この「中央から最も遠い」選択戦略により、R波のような大きな振れ幅を持つ波形成分において、にじみや太い線による複数候補がある場合でも、波形のピーク端に近いピクセルが優先的に選択される。

補間処理

信号が取得できなかった列（NaN）に対して線形補間を実施する。

```
NaN の列 i に対して:
  左側の最も近い有効値 l と右側の最も近い有効値 r を探索
  両方存在: 線形補間 signal[i] = signal[l] + (signal[r] -
signal[l]) × (i - l) / (r - l)
  左のみ: signal[i] = signal[l]
  右のみ: signal[i] = signal[r]
  両方なし: signal[i] = height / 2
```

3.3 Holter モード: カラーチャンネル優先抽出 (extractSignalHolter)

ホルター心電図を想定したアルゴリズムで、黒・緑・赤の3チャンネルに対応する。

走査範囲

```
yMargin = max(2, floor(height × 0.02))  ← 画像端のアーティファクト除外
yStart = yMargin
yEnd = height − yMargin

    黒チャンネル(channel === 'black')

Pass 1: 各列の最小輝度を算出
  minLum = min(0.299×R + 0.587×G + 0.114×B)  ← 全ピクセル対象

Pass 2: 重み付き重心で波形位置を算出
  threshold = min(minLum + 50, 200)

  判定条件:
    lum ≤ threshold（十分に暗い）
    sat < 40（彩度が低い = 無彩色 = 黒系）
    ※ sat = max(R,G,B) − min(R,G,B)

  重み w = threshold − lum + 1（暗いほど重い）
  signal[x] = height − (Σ(y × w) / Σw)  ← 重み付き重心

    緑チャンネル(channel === 'green')

判定条件:
  G > 40（緑成分が一定以上）
  G > R + 15（赤より十分に緑が強い）
  G > B + 15（青より十分に緑が強い）

重み w = max(1, G − max(R, B))（緑の優位度）
signal[x] = height − (Σ(y × w) / Σw)

    赤チャンネル(channel === 'red')

判定条件:
  R > 40（赤成分が一定以上）
  R > G + 15（緑より十分に赤が強い）

重み w = max(1, R − G)（赤の優位度）
signal[x] = height − (Σ(y × w) / Σw)

    補間処理
```

Standard モードと同一のロジックで NaN 列を線形補間する。

3.4 平滑化処理

抽出された信号に対して3点移動平均による平滑化を適用する。

```
smooth[0] = signal[0]
smooth[width-1] = signal[width-1]
smooth[i] = (signal[i-1] + signal[i] + signal[i+1]) / 3   (i = 1 〜 width-2)
```

R波の振幅保持を考慮し、意図的に窓幅を最小（3点）に設定している。

3.5 R波検出アルゴリズム (detectRPeaks)

Step 1: スコア信号の生成

2つの特徴量を組み合わせた複合スコアでR波候補を評価する。

```
(A) 勾配（slope）:
  slope[i] = |smooth[i+1] − smooth[i−1]|   （中央差分の絶対値）

(B) 信号中央値（エッジマージン 15px 除外）:
  interiorSmooth = smooth[edgeMargin .. width−edgeMargin] をソート
  signalMedian = 中央値

(C) 正規化振幅偏差:
  maxAbsDev = max(|smooth[i] − signalMedian|)   （エッジ除外内部領域）
  normDev[i] = |smooth[i] − signalMedian| / maxAbsDev

(D) 正規化勾配:
  slopeMax = max(slope[i])   （エッジ除外内部領域）
  normSlope[i] = slope[i] / slopeMax

(E) 複合スコア:
  scoreSignal[i] = normDev[i] + normSlope[i] × 3.0
  → 勾配に3倍の重みを付与（QRSの急峻な立ち上がり/下がりを重視）
```

Step 2: グローバル閾値によるR波候補検出（Pass 1）

```
globalThreshold = interiorMaxScore × 0.40   （内部領域最大スコアの40%）
minRDist = pxPerSec × 0.45   （最小RR間隔: 450ms → HR 133bpm相当）

ピーク条件:
  scoreSignal[i] > scoreSignal[i−1]（左隣より大きい）
  scoreSignal[i] > scoreSignal[i+1]（右隣より大きい）
  scoreSignal[i] > globalThreshold
  前回ピークとの距離 > minRDist
  距離 ≤ minRDist の場合はスコアが高い方を残す
```

Step 3: テンプレートマッチングによるギャップ補填（Pass 2）

検出済み R-peak が2個以上の場合に実施する。

```
(A) RR間隔の中央値算出:
  medianRR = 全RR間隔のメディアン
  gapThresholdRR = medianRR × 1.5   （1.5倍以上はギャップと判定）

(B) QRSテンプレート生成:
  halfW = max(5, floor(pxPerSec × 0.04))  ← ±40ms の窓幅
  templateLen = halfW × 2 + 1
  template = 全検出R-peak周辺の平均波形

(C) テンプレートの統計量:
```

```
  tMean = テンプレートの平均値
  tStd = テンプレートの標準偏差（√分散）

(D) ギャップ内で正規化相互相関（NCC）を算出:
  探索範囲: 前のR-peak + minRDist ～ 次のR-peak - minRDist

  各位置 pos について:
    局所区間 [pos-halfW, pos+halfW] の信号を取得
    sMean = 局所区間の平均
    sStd = 局所区間の標準偏差
    NCC = Σ((signal - sMean)(template - tMean)) / (sStd × tStd)

  NCC > 0.4（相関閾値）かつ最大の位置をR候補として挿入
```

Step 4: QRS極性の判定

```
各R-peak位置の信号値と signalMedian を比較:
  signalMedian より上: aboveCount++
  signalMedian より下: belowCount++

belowCount > aboveCount → isInvertedQRS = true（陰性QRS = QSパターン）
```

Step 5: ローカル極値へのスナップ

```
陽性QRS: 各R-peak 周辺 ±10px で信号最大値の位置に移動
陰性QRS: 各R-peak 周辺 ±20px で信号最小値の位置に移動
```

Step 6: RR間隔整合性フィルタ

```
メディアンRR間隔の65%未満の間隔が存在する場合:
  その間隔を構成する2つのピークのうち、
  scoreSignal が低い方を除去

これを短すぎる間隔がなくなるまで繰り返す
（ただし R-peak が3個未満になったら停止）
```

4. 接線法によるQT間隔計測

4.1 概要

QT間隔は Q-onset（Q波の開始点）から T-end（T波の終了点）までの時間であり、本アプリでは接線法（tangent method）により T-end を決定する。接線法は、T波の下行脚（陽性T波の場合）または上行脚（陰性T波の場合）における最急勾配点に接線を引き、その接線と基線の交点を T-end とする方法である。

4.2 ビート解析の処理フロー (analyzeBeats)

各ビートについて、以下の順序で処理される。

```
Phase 1: 粗Q-onset推定（基線なし）
Phase 2: Baseline（基線）算出
Phase 3: 精密Q-onset検出（基線あり）
Phase 4: T波極性判定
Phase 5: T-peak検出
Phase 6: 最急勾配点の探索と接線法によるT-end決定
Phase 7: QT/QTc算出
```

4.3 Phase 1: 粗Q-onset推定

R-peak から時間的に後方（左方向）に辿り、QRS群の開始点を粗く推定する。

陽性QRS（通常）の場合

```
探索範囲: R-peak から後方 80ms（= 0.08 × pxPerSec）以内
          ただし prevR + 10 より左には探索しない

R-peak位置から左へ1pxずつ辿る:
  smooth[k] ≤ smooth[roughQ] → roughQ = k（より低い点を更新）
  smooth[k] > smooth[roughQ] + 3 → 探索終了（急な上昇で打ち切り）
```

陰性QRS（QSパターン）の場合

```
探索範囲: 同上

R-peak（信号の谷 = nadir）から左へ辿り、
信号が上昇してフラットになる点を検出:
  smooth[k] ≥ maxVal → maxVal更新, roughQ = k
  smooth[k] < maxVal − 3 → 探索終了
```

4.4 Phase 2: Baseline（基線）算出

陽性QRSの場合

```
PR segment 領域を使用:
  baseStart = rLoc − 200ms (= 0.2 × pxPerSec)
  baseEnd   = rLoc −  60ms (= 0.06 × pxPerSec)

baseline = smooth[baseStart .. baseEnd] のメディアン
```

陰性QRS（QSパターン）の場合

```
粗Q-onset 直前の30px区間（PR segment）を使用:
  preQStart = max(prevR + 10, roughQ − 30)
  preQEnd   = roughQ

baseline = preQVals のメディアン（50パーセンタイル）
```

手動基線

ユーザーが基線（青い水平線）を上下にドラッグした場合、`manualBaselines[beatId]` に保存され、自動算出を上書きする。

4.5 Phase 3: 精密Q-onset検出

基線が確定した後、より精密な Q-onset を決定する。

陽性QRSの場合

```
探索範囲: R-peak から後方 40ms 以内（prevR + 10 以降）

R-peak位置から左へ辿り:
  smooth[k] ≤ qMinVal → qMinVal更新, qTrough = k
  smooth[k] > qMinVal + 3 → 探索終了（Q波の谷を通過）
```

陰性QRS（QSパターン）の場合

```
探索範囲: R-peak（nadir）から後方 80ms 以内

基線の95%閾値（baseThreshold = baseline × 0.95）を設定
R-peak位置から左へ辿り:
  smooth[k] ≥ baseThreshold → qTrough = k（基線に到達）

基線未到達の場合:
  粗Q-onset推定と同様のロジックでフォールバック
```

4.6 Phase 4: T波極性の自動判定

```
T波検索範囲:
  tStart = rLoc + max(120ms, 20px)
  tLimit = min(
    nextR − max(150ms, 20px),      ← 次のR波前のP波を除外
    rLoc + max(800ms, RR×0.75)     ← 長いQTに対応
  )

T波領域内の最大値（tMaxVal）と最小値（tMinVal）を取得:
  |tMinVal − baseline| > |tMaxVal − baseline| → 陰性T波
(isNegativeT = true)
  それ以外 → 陽性T波
```

4.7 Phase 5: T-peak検出

```
陽性T波: tStart〜tLimit 内で smooth の最大値の位置
陰性T波: tStart〜tLimit 内で smooth の最小値の位置

T-peak振幅フィルタ:
  |smooth[tPeak] − baseline| < 3 → ノイズとして棄却
```

4.8 Phase 6: 接線法によるT-end決定

これが本アプリの核心となるアルゴリズムである。

Step 1: 最急勾配点の探索

T-peak から T波検索範囲の終端（tLimit）まで、1px刻みで勾配を算出する。

勾配: s = smooth[k+1] − smooth[k]　（1pxあたりの信号変化量）

陽性T波: 最も急な下降勾配（s が最も負の値）の位置を slopeIdx とする
  bestSlope = min(s)　（最大の負の勾配）

陰性T波: 最も急な上昇勾配（s が最も正の値）の位置を slopeIdx とする
  bestSlope = max(s)　（最大の正の勾配）

Step 2: 接線とbaseline の交点計算

最急勾配点 (slopeIdx, smooth[slopeIdx]) を通り、勾配 bestSlope を持つ直線と、水平線 y = baseline の交点が T-end となる。

接線の方程式:
  y = bestSlope × (x − slopeIdx) + smooth[slopeIdx]

基線との交点:
  baseline = bestSlope × (tEnd − slopeIdx) + smooth[slopeIdx]

  tEnd = (baseline − smooth[slopeIdx]) / bestSlope + slopeIdx

Step 3: フォールバック処理

勾配が有効でない場合（slopeIdx が見つからない、または勾配の方向が不正な場合）:

フォールバック 1:
  T-peak の2px後から、信号が基線の ±2px 以内に戻る最初の点を T-end とする

フォールバック 2（フォールバック1も失敗）:
  T-peak の2px後から、基線との距離が最も小さい点を T-end とする

フォールバック 3（フォールバック2も失敗）:
  そのビートは棄却される（beats 配列に追加されない）

4.9 Phase 7: QT/QTc算出と妥当性検証

QT (ms) = (tEnd − qTrough) / pxPerSec × 1000
RR (sec) = (rLoc − prevR) / pxPerSec
HR (bpm) = 60 / RR

QTcB = QT / √RR
QTcF = QT / ∛RR

妥当性チェック:
  200 ≤ QT ≤ 700 ms の範囲外のビートは棄却

4.10 手動修正による接線の再計算

ユーザーが赤マーカー（接線の接触点）をドラッグした場合、接触点の x 座標が manualTEnds[beatId] に保存される。

```
手動接触点 cx での処理:
  cSlope = (smooth[cx+1] - smooth[cx-1]) / 2  ← 中央差分で局所勾配を算出
  cy = smooth[cx]
  tEnd = (baseline - cy) / cSlope + cx  ← 接線と基線の交点
  （cSlope === 0 の場合: tEnd = cx + 20 にフォールバック）
```

4.11 統計量の算出

全ビートの計測値の算術平均が最終結果として表示される。

```
HR   = mean(全ビートの hr)
QT   = mean(全ビートの qt)
QTcB = mean(全ビートの qtcB)
QTcF = mean(全ビートの qtcF)
```

5. ホルター心電図の判定方法

5.1 ホルターモードへの切り替え条件

ホルターモード（`extractionMode = 'holter'`）が適用される条件は以下の通りである。

```
条件 1: 自動キャリブレーション検出値が、ホルタープリセットに最も近い場合
  → findClosestPreset() がホルターを返す
  → extractionMode = 'holter' が自動設定

条件 2: ユーザーが手動で「ホルター」プリセットボタンを選択した場合
  → selectPreset('holter') が呼ばれる
  → extractionMode = 'holter' が設定される
```

5.2 ホルターモードと標準モードの違い

処理	標準モード (standard)	ホルターモード (holter)
波形抽出関数	`extractSignalStandard()`	`extractSignalHolter()`
赤方眼フィルタ	あり（赤ピクセルを除外）	なし
波形色	黒のみ	黒/緑/赤（チャンネル選択）
信号取得方式	最暗ピクセル追跡（中央から最遠）	重み付き重心
チャンネル選択UI	非表示	表示
画像端マージン	上下各1px	上下各2%（最低2px）

5.3 最適チャンネル自動検出 (detectBestChannel)

画像読み込み時にホルターモードが選択された場合、最適な波形チャンネルが自動判定される。

サンプリング方法

```
サンプリング間隔: step = max(1, floor(width / 200))
→ 画像全体から約200列をサンプリング（計算量削減）
```

各チャンネルのスコア算出

```
黒スコア（blackScore）:
  lum < 80 かつ sat < 40
  → 暗くて彩度が低いピクセル

緑スコア（greenScore）:
  G > 40 かつ G > R + 15 かつ G > B + 15
  → 緑が支配的なピクセル

赤スコア（redScore）:
  R > 40 かつ R > G + 15
  → 赤が支配的なピクセル
```

選択ロジック

```
colorThreshold = blackScore × 0.3

(1) 緑が赤より多く、かつ colorThreshold を超え、
    かつ blackScore の50%以上 → 'green' を選択

(2) 赤が緑より多く、かつ colorThreshold を超え、
    かつ blackScore の50%以上 → 'red' を選択

(3) それ以外 → 'black' を選択（デフォルト）
```

この選択ロジックにより、明確なカラー波形がある場合のみカラーチャンネルが選択され、カラー成分が少ない場合は安全に黒チャンネルにフォールバックする。

5.4 手動チャンネル切り替え

ユーザーがチャンネルバー上の「⬛ 黒」「🟩 緑」「🟥 赤」ボタンをクリックすると、`switchChannel()` が呼ばれ、波形抽出からやり直される。

```
switchChannel(index, channel):
  state.manualChannelSelected = true   ← 以降の自動検出を抑制
  state.channel = channel
  performSignalExtraction(state)       ← 波形抽出→R波検出→ビート解析を再実行
```

`manualChannelSelected = true` が設定されると、`performSignalExtraction()` 内で `detectBestChannel()` がスキップされ、ユーザーが選択したチャンネルが維持される。

5.5 ホルターモードの波形抽出の特徴

標準モードとの最大の違いは、信号の取得方式にある。

標準モード：
「最暗ピクセル → 画像中央から最も遠いピクセル」を選択
→ 1px の離散的な位置が返される
→ R波のような大きな振れ幅に対して先端側を優先

ホルターモード：
「条件を満たす全ピクセルの重み付き重心」を算出
→ サブピクセル精度の連続値が返される
→ 波形の幅の中心を追跡するため、安定した信号が得られる
→ 太い波形線でも正確な追跡が可能

6. ユーザーインタラクション

6.1 マーカー操作

操作	対象	動作
クリック	R-peak（青丸）	そのR-peakを削除
ドラッグ	R-peak（青丸）	R-peak位置を水平移動
Shift+クリック	キャンバス任意位置	その位置にR-peakを追加
ドラッグ	赤マーカー（接線接触点）	接線の接触点を移動→接線とT-endを再計算
クリック	赤マーカー	手動修正をリセット（自動検出に戻す）
ドラッグ	金色マーカー（T-peak）	T-peakを移動→接線を再計算
ドラッグ	緑マーカー（Q-onset）	Q-onset位置を移動
クリック	緑マーカー	Q-onsetの手動修正をリセット
ドラッグ	青い水平線（基線）	基線を上下に移動
クリック	青い水平線	基線の手動修正をリセット
ドラッグ	H型バー右端	キャリブレーション値を変更
ドラッグ	H型バー左端	キャリブレーション表示位置を移動

6.2 データ永続化（localStorage）

キー	内容	形式
`qtc_cal_presets`	キャリブレーションプリセット値	`{"100": 94, "200": 191, "holter": 126}`
`qtc_analysis_history`	解析履歴（最大20件）	JSON配列

7. 処理パイプライン全体図

```
画像読み込み（handleFile）
  ↓
リサイズ判定（幅 > 2000px なら縮小）
  ↓
runAnalysis()
  ├── detectCalibration()          ← 赤グリッド間隔から pxPerSec を自動検出
  ├── findClosestPreset()          ← 検出値に最も近いプリセットを選択
  ├── プリセット値にスナップ        ← pxPerSec = calPresets[matchedPreset]
  ├── extractionMode 決定          ← 'standard' or 'holter'
  └── performSignalExtraction()
        ├── [holter] detectBestChannel()     ← 最適チャンネル自動判定
        ├── [holter] extractSignalHolter()   ← カラーチャンネル重み付き重心
        ├── [standard] extractSignalStandard() ← 最暗ピクセル追跡
        ├── 3点平滑化
        ├── detectRPeaks()
        │     ├── スコア信号生成（振幅偏差 + 勾配×3）
        │     ├── Pass 1: グローバル閾値検出
        │     ├── Pass 2: テンプレートマッチング（NCC）ギャップ補填
        │     ├── QRS極性判定
        │     ├── ローカル極値スナップ
        │     └── RR整合性フィルタ
        ├── analyzeBeats()
        │     ├── Phase 1: 粗Q-onset推定
        │     ├── Phase 2: Baseline算出
        │     ├── Phase 3: 精密Q-onset検出
        │     ├── Phase 4: T波極性判定
        │     ├── Phase 5: T-peak検出
        │     ├── Phase 6: 接線法 T-end決定
        │     └── Phase 7: QT/QTc算出 + 妥当性検証(200-700ms)
        ├── renderCanvas()           ← マーカー・接線・基線を描画
        └── updateTable()            ← 結果テーブル更新
```

本仕様書は qtc.html ver 1.5 (2026/3/5) のソースコードに基づいて作成された。