

# コンピュータビジョン

担当: 井尻 敬

## Contents 画像内の特定パターンを発見する手法

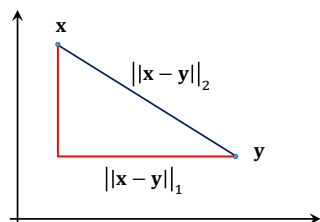
- テンプレートマッチング
- コーナー検出 (Harris corner detector/FAST)
- エッジ検出 (Canny edge detector)
- 直線の検出: Hough変換
- 特徴点と特徴ベクトル
  - SIFT特徴
  - 特徴点の対応付け

## 準備: ノルム(norm)

$d$ 次元空間のベクトル  $\mathbf{x} = (x_1, x_2, \dots, x_n)$  の  $p$ -ノルムは以下の通り定義される

$$\|\mathbf{x}\|_p = (|x_1|^p + |x_2|^p + \dots + |x_n|^p)^{\frac{1}{p}}$$

例  $d=2$  のとき



$p=2$  なら...

$$\|\mathbf{x} - \mathbf{y}\|_2 = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$$

これはよく知っているユークリッド空間の距離

$p=1$  なら...

$$\|\mathbf{x} - \mathbf{y}\|_1 = |x_1 - y_1| + |x_2 - y_2|$$

点  $\mathbf{x}$  から点  $\mathbf{y}$  へ、軸に沿った方向のみで移動した際の距離  
市街地における移動距離になぞらえて**市街地距離**や**マンハッタンノルム**と呼ばれる

左の画像から右の画像を探せ



※地味な例ですみません。。。

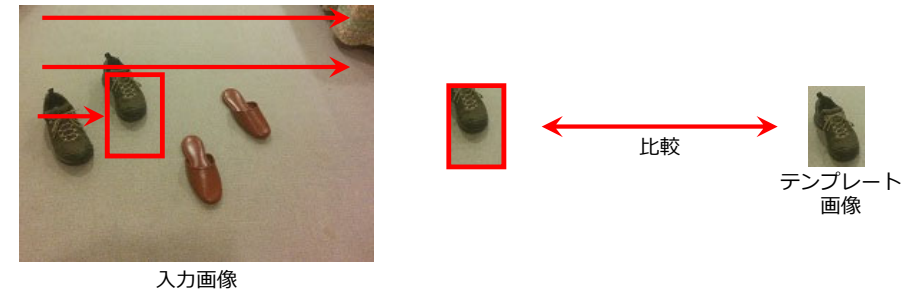
左の画像から右の画像を探せ



※地味な例ですみません。。

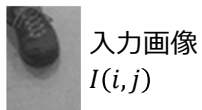
## テンプレート マッチング

TemplateMatching.py

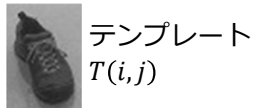


- 入力画像をラスタスキャンし、入力画像とテンプレートの類似度を比較
- 類似度が閾値より高い部分を出力する
- ※テンプレート：検索対象を表す標準画像
- ※ラスタスキャン：画像を左から右に、上から下に、一画素ずつ走査すること

## 類似度（相違度）の定義



入力画像  
 $I(i, j)$



テンプレート  
 $T(i, j)$

Grayscale化  
されている

- 相違度: **Sum of Square Distance**

$$R_{SSD} = \sum_{i,j} (I(i, j) - T(i, j))^2$$

- 相違度: **Sum of Absolute Distance**

$$R_{SAD} = \sum_{i,j} |I(i, j) - T(i, j)|$$

- 類似度: **Normalized Cross Correlation**(正規化相互相関)

$$R_{NCC} = \frac{\sum_{i,j} I(i, j)T(i, j)}{\sqrt{\sum_{i,j} I(i, j)^2 \sum_{i,j} T(i, j)^2}}$$

## テンプレートマッチングの結果

templateMatching.py

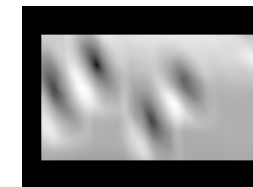


テンプレート

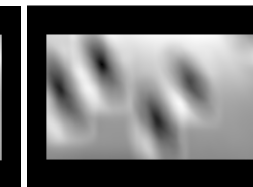


入力画像

SAD



SSD



NCC



$$\sum_{i,j} |I(i, j) - T(i, j)|$$

$$\sum_{i,j} (I(i, j) - T(i, j))^2$$

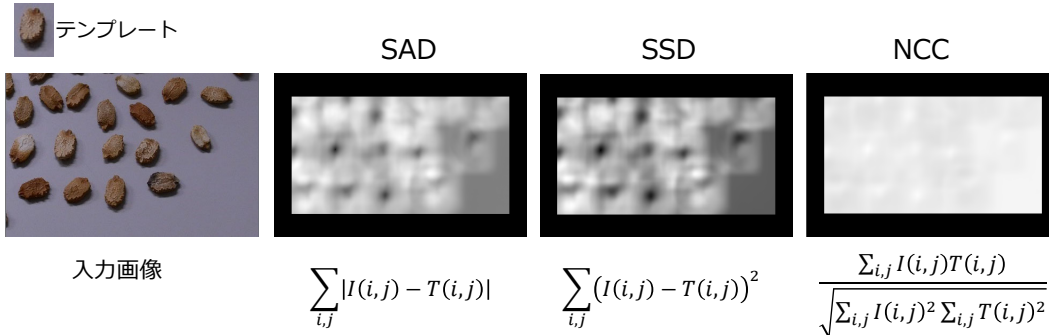
$$\frac{\sum_{i,j} I(i, j)T(i, j)}{\sqrt{\sum_{i,j} I(i, j)^2 \sum_{i,j} T(i, j)^2}}$$

SAD/SSDは相違度なので、近いところほど値が小さくなる

NCCは類似度なので近いところほど値が大きくなる

例えば、閾値以下の局所最小部を検出対象とすればよい

## テンプレートマッチングの結果



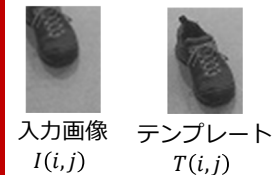
SAD/SSDは相違度なので、近いところほど値が小さくなる

NCCは類似度なので近いところほど値が大きくなる

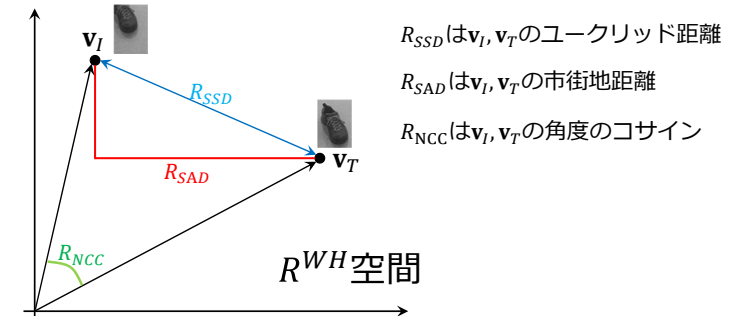
例えば、閾値以下の局所最小部を検出対象とすればよい

## 類似度・相違度の定性的理解

- 入力画像・テンプレートは  $W \times H$  グレースケール画像
- これを  $(WH)$ -次元ベクトルと考える



$\mathbf{v}_I, \mathbf{v}_T \in R^{WH}$



## Pythonコードの動かし方

- Github リポジトリにアクセス
  - <https://github.com/TakashiIjiri/PythonOpenCVPractice>
  - # 良い機会なのでアカウント作っても良いかも
- ソースコードをClone or ダウンロード
- anacondaをインストール + OpenCVをインストール
- コマンドプロンプトで動かす

## サブピクセル精度のテンプレートマッチング

- テンプレートマッチングは目的画像にテンプレート画像を重ね差分を評価するため発見できる位置は**ピクセル単位（離散値）**
- **サブピクセル（連続値）**精度で位置検出を行いたい
- 局所的に関数をフィッティングし、最小値を求める
  - 等角直線フィッティング
  - パラボラフィッティング

## テンプレートマッチングの高速化



$W \times H$



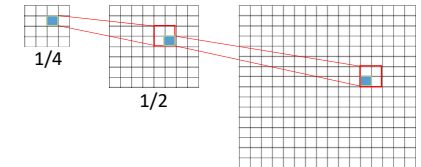
$w \times h$

対象画像全領域にテンプレートを重ね合わせて差分を計算する計算量は...



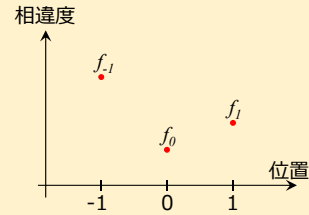
**残差逐次検定：** 目標画像をラスタスキャンしテンプレートとの差分計算をする際、現在の最小値よりも差分が大きくなったら計算を打ち切る

**粗密探索法：** ガウシアンピラミッドを生成。低解像度画像にてマッチングする画素を発見。ひとレベル高解像度画像に移動し、発見した画素に関する数画素のみに対してマッチングを計算する



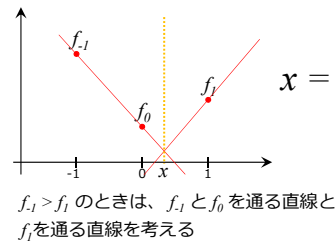
### 問題

- 相違度が最小の画素を原点( $x=0$ )にとる
  - $x=\pm 1$  の相違度も既知
  - 最小値を与える位置 $x$  (実数精度) はどこ?
- ※画像に適用する際は縦横を独立に扱えば良い



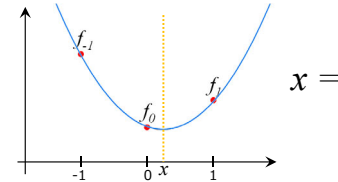
### 等角直線フィッティング

下図の通り傾きが-1倍の2本の直線の交点を利用



### パラボラフィッティング

二次関数で相違度を補間し相違度の最小位置を求める



## まとめ：テンプレートマッチング

- 入力画像から物体を検出するための手法
- 検出対象の画像 (テンプレート) を用意し、入力画像をラスタスキャンし相違度を評価
- 相違度が閾値以下の領域を出力する
- 相違(類似)度：SAD, SSD, NCCなど



入力画像



テンプレート

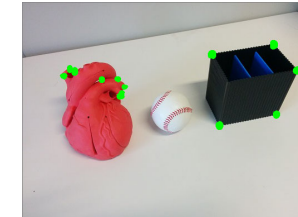
### サブピクセル精度で検出するための関数フィッティング

高速化のための残差逐次検定・粗密(coarse to fine)探索・chamfer matching

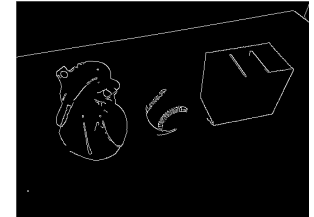
## コーナー、輪郭線の検出

HarrisCorner.py  
CannyEdge.py

物体認識・物体追跡・位置あわせなど、より高度な画像処理に利用するため画像から『コーナー』や『輪郭線』といった特徴的な点・曲線を検出する



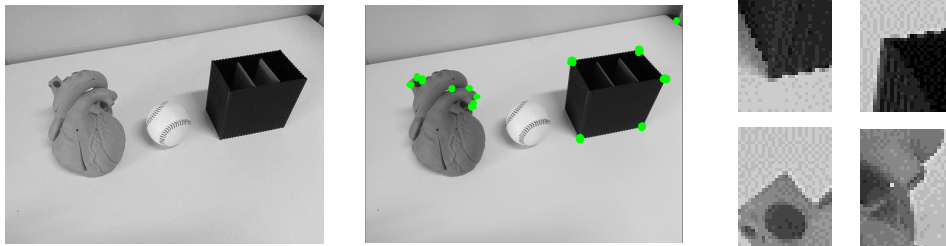
コーナー検出  
(Harris Corner Detector)



輪郭検出  
(Canny Edge detector)

## Harrisのコーナー検出アルゴリズム

[C. Harris & M. Stephens (1988). "A Combined Corner and Edge Detector". Proc. of the 4th ALVEY Vision Conference. pp. 147-151.]



• 入力：グレースケール画像

• 出力：コーナー画素群

• 手法の概要

Harris行列（又はStructure tensor matrixと呼ばれる）を定義し、この固有値固有ベクトルを用いて、局所領域の輝度変化方向と変化量を検出する  
局所領域の輝度変化が、直交する2方向について大きくなる部分をコーナーと定義

## Structure tensor matrix (1/3)

画像上の点 $(x, y)$ の輝度値を $I(x, y)$ と表す

点 $(x, y)$ における**Structure tensor matrix**は以下の通り定義される

$$\mathbf{A}(x, y) = \sum_{u, v} G(u, v) \begin{pmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{pmatrix}$$

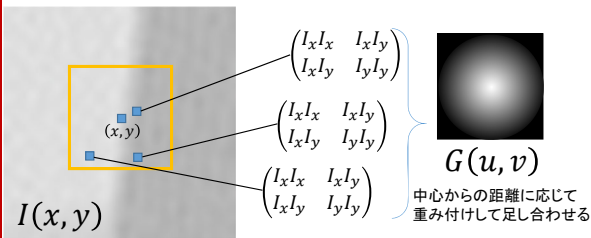
ただし、 $I_y = I_y(x + u, y + v)$ ,  $I_x = I_x(x + u, y + v)$  と省略したもの  
 $I_x$ と $I_y$ は、x方向・y方向の微分画像（sobel filter）  
また、 $G(u, v)$ は重み関数（ガウシアンを用いる）

※教科書の式11.6 ~ 11.9に対応する

## Structure tensor matrix (2/3)

実際の計算手順

$$\mathbf{A}(x, y) = \sum_{u, v} G(u, v) \begin{pmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{pmatrix}$$



Structure Tensorの性質

- 固有値を $\lambda_1, \lambda_2$  とする ( $\lambda_1 > \lambda_2$ )
- 固有ベクトルを  $\mathbf{v}_1, \mathbf{v}_2$ とする
- 対称行列 → 固有値は実数
- 対称行列 → 固有ベクトルは直交
- 半正定置 →  $\lambda_1 \geq 0, \lambda_2 \geq 0$ 
  - 半正定置行列の和なのでStructure tensorは半正定値になる

- $\mathbf{v}_1$ は輝度値変化の最も大きな方向
- $\lambda_1$ は $\mathbf{v}_1$ 方向の輝度値変化の大きさ
- $\lambda_2$ は $\mathbf{v}_2$ 方向の輝度値変化の大きさ

## Harrisのコーナー検出アルゴリズム

グレースケール画像からコーナーを検出

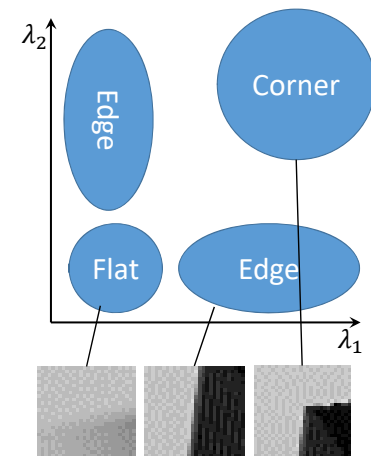
1. 各画素 $(x, y)$ におけるStructure Tensor  $\mathbf{A}$  と固有値 $\lambda_1, \lambda_2$ を計算
2. 各画素 $(x, y)$ において $R = \lambda_1 \lambda_2 - k(\lambda_1 + \lambda_2)^2$ を計算
3.  $R$ が極大かつ閾値以上の点をコーナーとして出力する

※ただし、 $k$ はユーザが指定するパラメタ (0.04~0.06)

※ $R = \lambda_1 \lambda_2 - k(\lambda_1 + \lambda_2)^2$ は、コーナーらしさを現す関数：  
 $\lambda_1$ と $\lambda_2$ が大きいかつ近いときに大きな値を返す

評価式Rの3Dプロット →

[http://www.wolframalpha.com/input/?i=z%3Dx\\*y+-+0.02\\*\(x%2By\)%5E2](http://www.wolframalpha.com/input/?i=z%3Dx*y+-+0.02*(x%2By)%5E2)



## Harrisのコーナー検出アルゴリズム

グレースケール画像からコーナーを検出

1. 各画素 $(x, y)$ におけるStructure Tensor  $\mathbf{A}$  と固有値 $\lambda_1, \lambda_2$ を計算
2. 各画素 $(x, y)$ において $R = \lambda_1 \lambda_2 - k(\lambda_1 + \lambda_2)^2$ を計算
3.  $R$ が極大かつ閾値以上の点をコーナーとして出力する



グレースケール画像からコーナーを検出 **new**

1. 各画素 $(x, y)$ におけるStructure Tensor  $\mathbf{A}$  を計算
2. 各画素 $(x, y)$ において $R = \det \mathbf{A} - k(\text{tr } \mathbf{A})^2$ を計算
3.  $R$ が極大かつ閾値以上の点をコーナーとして出力する

**固有値の計算時間が無駄**

$$\det \mathbf{A} = \lambda_1 \times \lambda_2$$

$$\text{tr } \mathbf{A} = \lambda_1 + \lambda_2$$

**という関係を利用すると  
計算を効率化できる**

※練習) 上記の関係を証明せよ

## まとめ：コーナー・輪郭検出

### コーナー検出：画像中の『角』形状を検出

- **Harris Corner detection**  
→ Structure Tensorの固有値により角らしさを定義
- 様々な手法が知られる(FAST/SUSAN/ヘッセ行列)

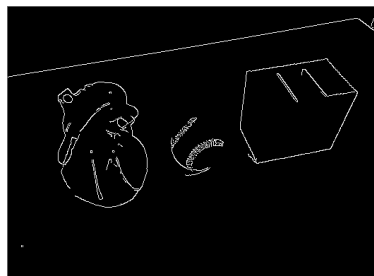
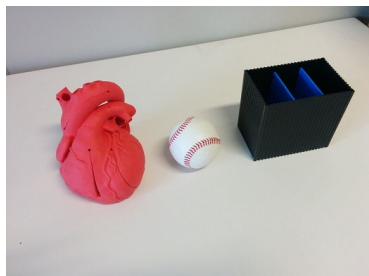
### 輪郭検出：画像中の物体と物体の境界を検出

- **Canny Edge Detection**
  - 微分フィルタによる勾配画像取得
  - 勾配方向を考慮した細線化
  - 二つの閾値処理
- 様々な手法が知られる(Sobel/Hough変換…)

## Cannyの輪郭線検出アルゴリズム

※井尻はキャニーと呼んでますが、教科書はケニーですね。。。

画像からエッジ（輝度値変化の大きな輪郭線）を抽出する



## Cannyの輪郭線検出アルゴリズム(1/2)

※井尻はキャニーと呼んでますが、教科書はケニーですね。。。

### 1. ガウシアンフィルタをかける： $I \rightarrow G * I$

例)  $5 \times 5$ ,  $\sigma = 1.4$  のガウシアンなどが利用される



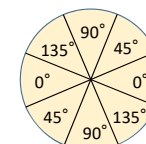
### 2. 勾配強度・勾配方向計算

Sobel filterにより縦横方向の微分を計算： $I \rightarrow I_x, I_y$

$$\text{勾配強度： } g(x, y) = \sqrt{I_x(x, y)^2 + I_y(x, y)^2}$$

$$\text{勾配方向： } d(x, y) = \tan^{-1} \frac{I_y(x, y)}{I_x(x, y)}$$

( $0^\circ/45^\circ/90^\circ/135^\circ$ の4通りに量子化)



参考: OpenCV [http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/canny\\_detector/canny\\_detector.html](http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html)

原著論文: Canny, J., A Computational Approach To Edge Detection, IEEE PAMI, 1986.



## Cannyの輪郭線検出アルゴリズム(2/2)

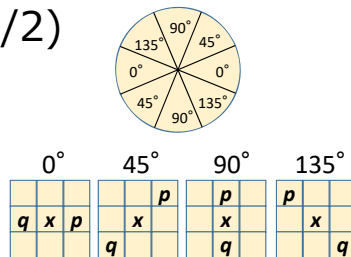
### 3. non-maximum suppression

細い輪郭線抽出のため、勾配強度が極大となる画素のみを残す

勾配強度画像の各画素 $x$ に対して…

勾配方向に隣接する2画素 $p, q$ と $x$ の勾配強度を比較

画素 $x$ の勾配強度が $p, q$ と比べて最大でないなら $x$ の勾配強度を0に



### 4. 閾値処理

二つの閾値 $T_{max}$ と $T_{min}$ を用意

勾配強度画像の画素 $x$ の勾配強度が…

- $T_{max}$ より大きい → Strong edge: 画素 $x$ は輪郭線である
- $T_{min}$ より小さい → not edge: 画素 $x$ は輪郭線でない
- それ以外 → weak edge: もしstrong edgeに隣接していれば輪郭線とする

※紹介したものは実装の一例です。

## まとめ：コーナー・輪郭検出

### コーナー検出：画像中の『角』形状を検出

#### • Harris Corner detection

→ Structure Tensorの固有値により角らしさを定義

• 様々な手法が知られる(FAST/SUSAN/ヘッセ行列)

### 輪郭検出：画像中の物体と物体の境界を検出

#### • Canny Edge Detection

- 微分フィルタによる勾配画像取得
- 勾配方向を考慮した細線化
- 二つの閾値処理
- 様々な手法が知られる(Sobel/Hough変換…)

## まとめ：コーナー・輪郭検出

### コーナー検出：画像中の『角』形状を検出

#### • Harris Corner detection

→ Structure Tensorの固有値により角らしさを定義

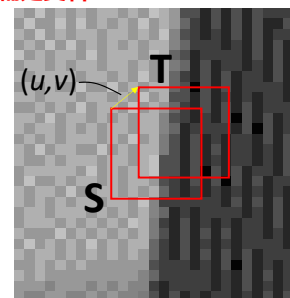
• 様々な手法が知られる(FAST/SUSAN/ヘッセ行列)

### 輪郭検出：画像中の物体と物体の境界を検出

#### • Canny Edge Detection

- 微分フィルタによる勾配画像取得
- 勾配方向を考慮した細線化
- 二つの閾値処理
- 様々な手法が知られる(Sobel/Hough変換…)

#### 補足資料



### Structure Tensor Matrix (導出)

[A Combined Corner and Edge Detector in 1988]

窓領域 $S$ と $S$ を微量 $(u, v)$ だけ移動した領域 $T$ を考える。

この2領域の重み付き二乗誤差は以下の通り。

$$D(u, v) = \sum_{(x, y) \in S} G(x, y) (I(x + u, y + v) - I(x, y))^2 \quad \dots (1)$$

これは $S$ を $(u, v)$ だけずらした際の画像の変化量を示す

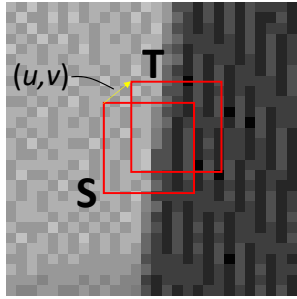
※ 重み関数 $G(x, y)$ には、ガウシアンがよく用いられる。

テーラー展開し2次以降の項を無視すると、以下の変形が得られる

$$I(x + u, y + v) \approx I(x, y) + uI_x(x, y) + vI_y(x, y)$$

これを(1)に代入すると、以下の通りStructure Tensor Matrix  $A$  が現れる

$$D(u, v) = (u, v) \mathbf{A} \begin{pmatrix} u \\ v \end{pmatrix}, \quad \mathbf{A} = \sum_{(x, y) \in S} G(x, y) \begin{pmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{pmatrix}$$



## Structure Tensor Matrix（導出）

窓領域 $S$ と $S$ を $(u, v)$ だけ移動した領域 $T$ の二乗誤差は以下の通り

$$D(u, v) = (u, v) \mathbf{A} \begin{pmatrix} u \\ v \end{pmatrix}, \quad \mathbf{A} = \sum_{(x, y) \in S} G(x, y) \begin{pmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{pmatrix}$$

今知りたいのは、どの方向 $(u, v)$ に動かすと差分が最大になるか？つまり、画像の変化が大きいか？である。そのため以下の最大化問題を考える。

$$\operatorname{argmax}_{(u, v)} \frac{(u, v) \mathbf{A} \begin{pmatrix} u \\ v \end{pmatrix}}{\begin{pmatrix} u \\ v \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix}}$$

この目的関数はレイリー商と呼ばれ、 $(u, v)$ が行列 $\mathbf{A}$ の固有ベクトルに一致するとき、最大値（最小値）をとり、最大値・最小値は固有値と一致することが知られている（証明省略）。

つまり、Structure Tensor matrixの固有値固有ベクトルを $\lambda_1, \lambda_2$  ( $\lambda_1 > \lambda_2$ )

$\mathbf{v}_1, \mathbf{v}_2$ とすると、 $(u, v)$ が $\mathbf{v}_1$ に一致するとき画像は最も大きく変化する。

また $(u, v)$ が $\mathbf{v}_2$ に一致するとき画像の変化は最小になる。