コンピュータビジョン
物体追跡

動画像中の物体の追跡

- 小領域の類似度が最大になる変位を推定：テンプレートマッチング

$$E_{\mathrm{SSD}}(\boldsymbol{u}) = \sum_i [I_1(\boldsymbol{x}_i + \boldsymbol{u}) - I_0(\boldsymbol{x}_i)]^2 = \sum_i e_i^2,$$

sum of squared difference

$$E_{\mathrm{SAD}}(\boldsymbol{u}) = \sum_i |I_1(\boldsymbol{x}_i + \boldsymbol{u}) - I_0(\boldsymbol{x}_i)| = \sum_i |e_i|.$$

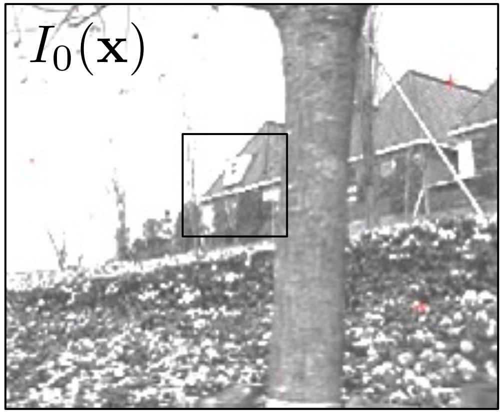
sum of absolute difference

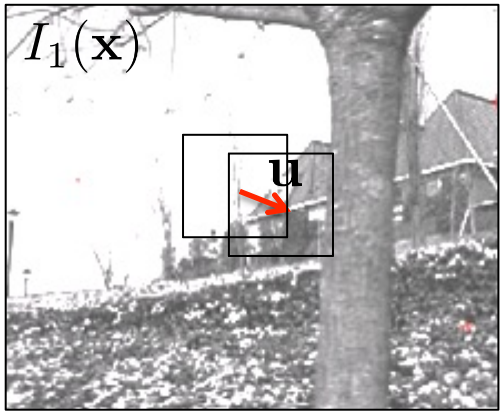
$$E_{\mathrm{CC}}(\boldsymbol{u}) = \sum_i I_0(\boldsymbol{x}_i) I_1(\boldsymbol{x}_i + \boldsymbol{u})$$

cross correlation

$$E_{\mathrm{NCC}}(\boldsymbol{u}) = \frac{\sum_i [I_0(\boldsymbol{x}_i) - \overline{I_0}]\,[I_1(\boldsymbol{x}_i + \boldsymbol{u}) - \overline{I_1}]}{\sqrt{\sum_i [I_0(\boldsymbol{x}_i) - \overline{I_0}]^2}\sqrt{\sum_i [I_1(\boldsymbol{x}_i + \boldsymbol{u}) - \overline{I_1}]^2}},$$

normalized cc





Template matching: コード

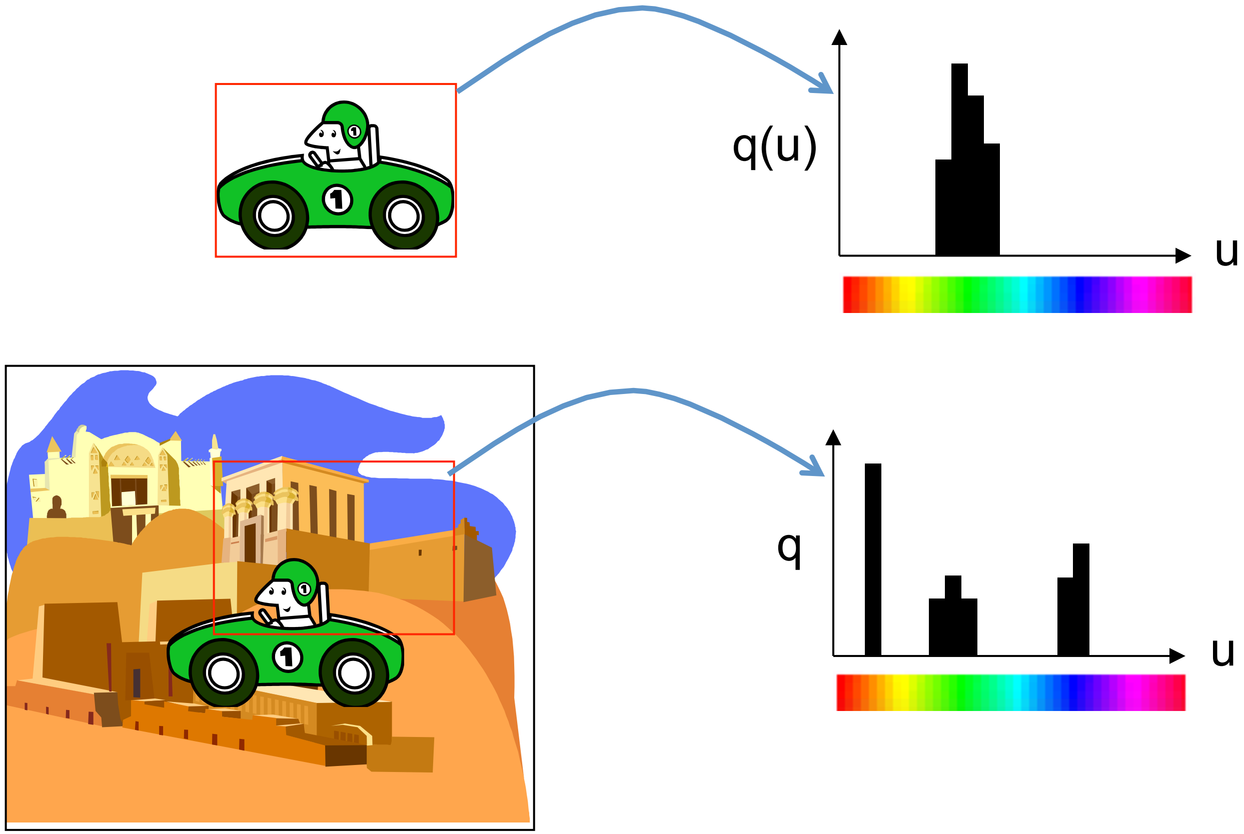
temp_track.py

```
while 1:
    ret, frame = cap.read()
    frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    if templ != None:
        resp = cv2.matchTemplate(frame_gray, templ,\
                                 method=cv2.cv.CV_TM_SQDIFF)
        min, max, minpos, maxpos = cv2.minMaxLoc(resp)
        x, y = minpos[0], minpos[1]

    cv2.rectangle(frame, (x, y), (x+w, y+h), 255, 2)
    cv2.imshow('Template tracker', frame)

    k = cv2.waitKey(5)
    if k == 0x20: # Space bar
        # Capture the target
        templ = frame_gray[y:y+h, x:x+w].copy()
    elif k == 0x1b: # ESC
        break
```

Mean shift tracker

[Comaniciu-Ramesh-Meer03]

- 平均値シフト法の物体追跡への応用
  - 追跡対象を色分布（ヒストグラム）で表現
  - 現在の位置（前フレーム結果）からどちら方向にあるか？

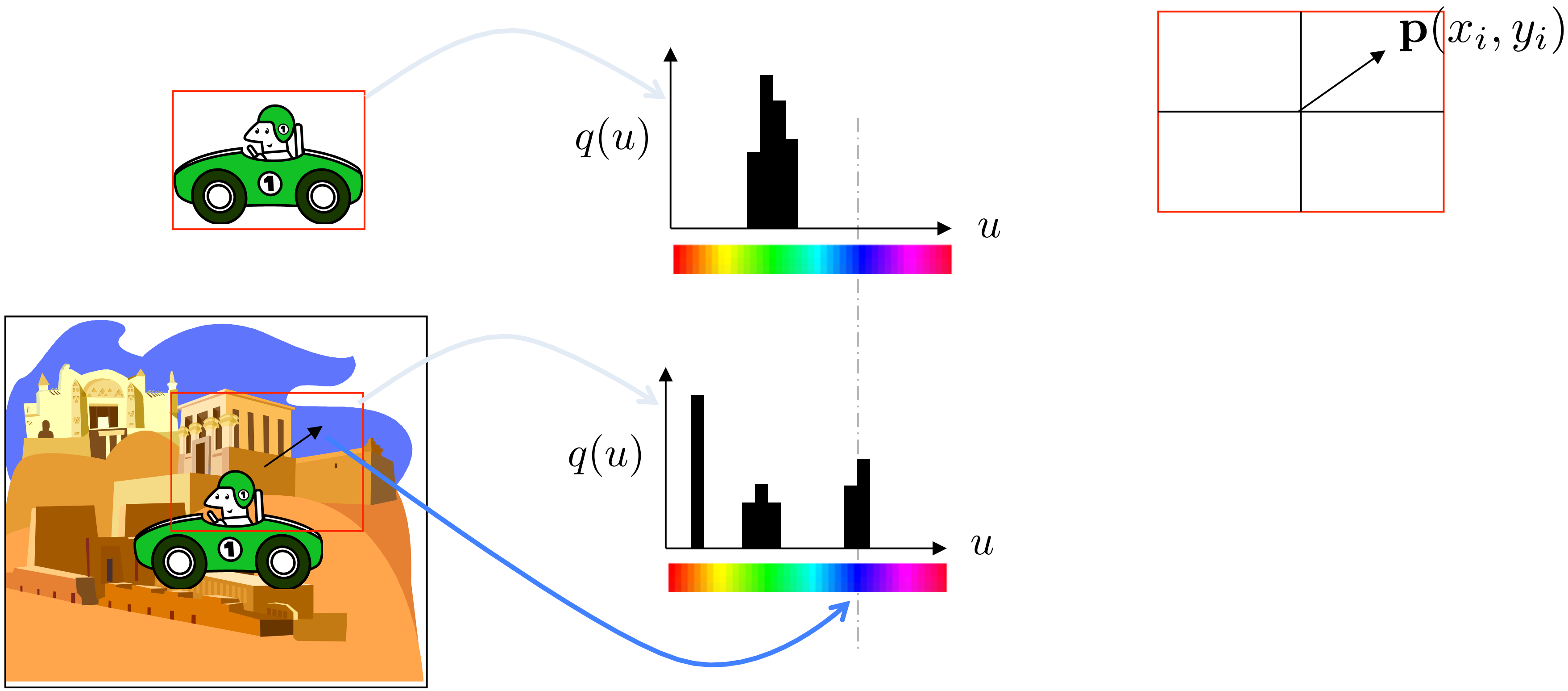


Mean shift tracker

- Mean shift vector: $\mathbf{v} = \sum_D \underline{w(x_i, y_i)}\,\underline{\mathbf{p}(x_i, y_i)}$

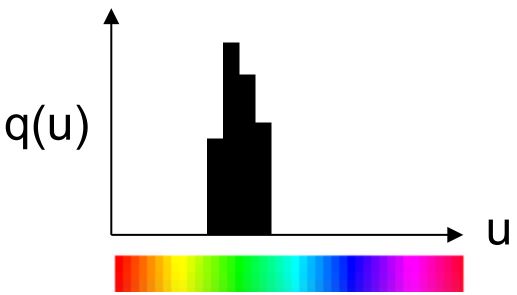
その画素の色が対象に
含まれているか
（多くあるほど大）

画素の中央からの
「シフトベクトル」

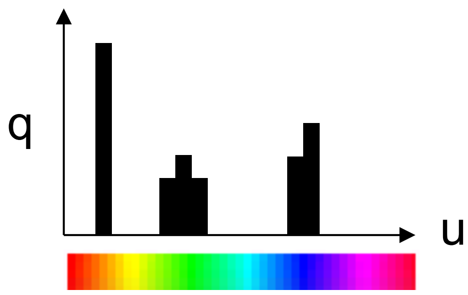


Mean shift tracker

- 今の位置でmean shiftベクトルを計算し，その向きに領域を少し動かすことを反復

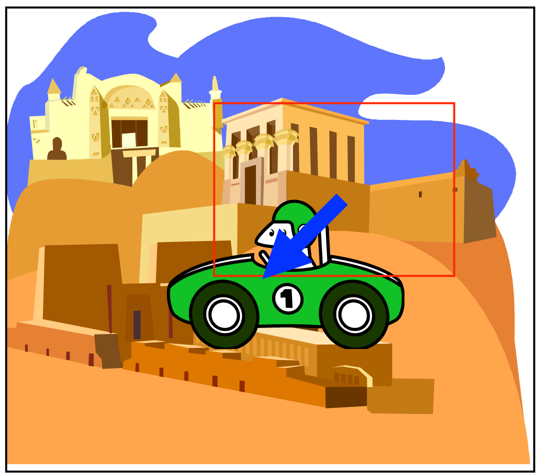


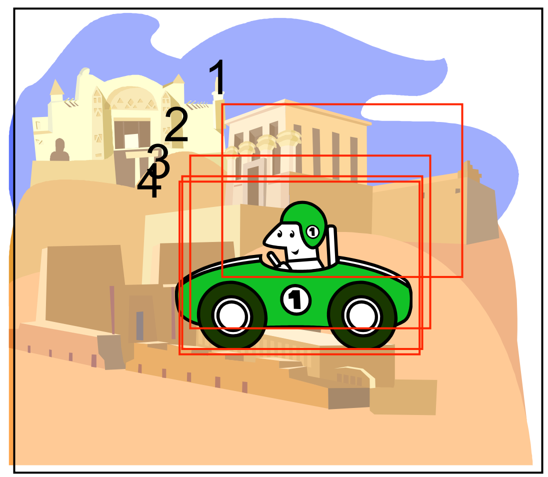
ターゲットの
ヒストグラム



現在位置での
ヒストグラム

現在位置での
mean shift
ベクトル





Mean shift tracker: コード

mshift_tracker.py

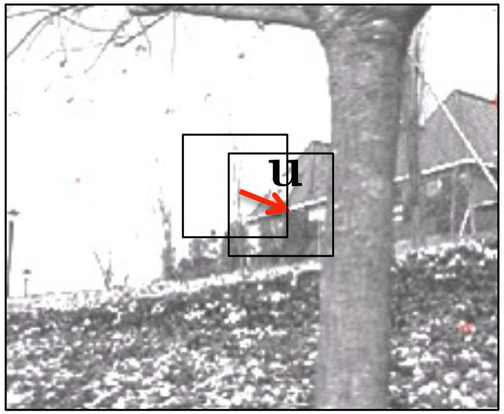
```
# Initialization
x0, y0, w, h = 320-50, 240-50, 100, 100
track_window = (x0, y0, w, h)
roi_hist = None

# Termination criteria = 10 iteration or 1 pix motion
term_crit = ( cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 1 )

while 1:
    ret, frame = cap.read()

    if roi_hist != None:
        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
        dst = cv2.calcBackProject([hsv], [0], roi_hist, [0,180], 1)
        ret, track_window = cv2.meanShift(dst, track_window, term_crit)

    x, y, w, h = track_window
    cv2.rectangle(frame, (x, y), (x+w, y+h), 255, 2)
    cv2.imshow('Meanshift tracker', frame)

    k = cv2.waitKey(5)
    if k == 0x20: # Space bar
        # Create histogram for the target
        hsv_roi =  cv2.cvtColor(frame[y:y+h,x:x+w], cv2.COLOR_BGR2HSV)
        roi_hist = cv2.calcHist([hsv_roi], [0], None, [32], [0,180])
        cv2.normalize(roi_hist, roi_hist, 0, 255, cv2.NORM_MINMAX)
```

特徴点の追跡

- 特徴点周りの小領域の濃淡の差を最小化 [Lucas-Kanade81]

$$
\begin{aligned}
E_{\mathrm{LK-SSD}}(\boldsymbol{u}+\Delta\boldsymbol{u}) &= \sum_i [I_1(\boldsymbol{x}_i+\boldsymbol{u}+\Delta\boldsymbol{u}) - I_0(\boldsymbol{x}_i)]^2 \\
&\approx \sum_i [I_1(\boldsymbol{x}_i+\boldsymbol{u}) + \boldsymbol{J}_1(\boldsymbol{x}_i+\boldsymbol{u})\Delta\boldsymbol{u} - I_0(\boldsymbol{x}_i)]^2
\end{aligned}
$$

$$
\boldsymbol{J}_1(\boldsymbol{x}_i+\boldsymbol{u}) = \nabla I_1(\boldsymbol{x}_i+\boldsymbol{u}) = (\frac{\partial I_1}{\partial x}, \frac{\partial I_1}{\partial y})(\boldsymbol{x}_i+\boldsymbol{u})
$$



特徴点の追跡

- 解法：線形方程式を解いて u を更新することを反復
  - 更新：$\boldsymbol{u} + \Delta\boldsymbol{u} \rightarrow \boldsymbol{u}$
  - $\Delta\boldsymbol{u}$ の決定：

$$
\begin{aligned}
E_{\mathrm{LK-SSD}}(\boldsymbol{u}+\Delta\boldsymbol{u}) &\approx \sum_i [I_1(\boldsymbol{x}_i+\boldsymbol{u}) + \boldsymbol{J}_1(\boldsymbol{x}_i+\boldsymbol{u})\Delta\boldsymbol{u} - I_0(\boldsymbol{x}_i)]^2 \\
&= \sum_i [\boldsymbol{J}_1(\boldsymbol{x}_i+\boldsymbol{u})\Delta\boldsymbol{u} + e_i]^2,
\end{aligned}
$$

（線形最小二乗）

$$e_i = I_1(\boldsymbol{x}_i+\boldsymbol{u}) - I_0(\boldsymbol{x}_i)$$

$$\Downarrow$$

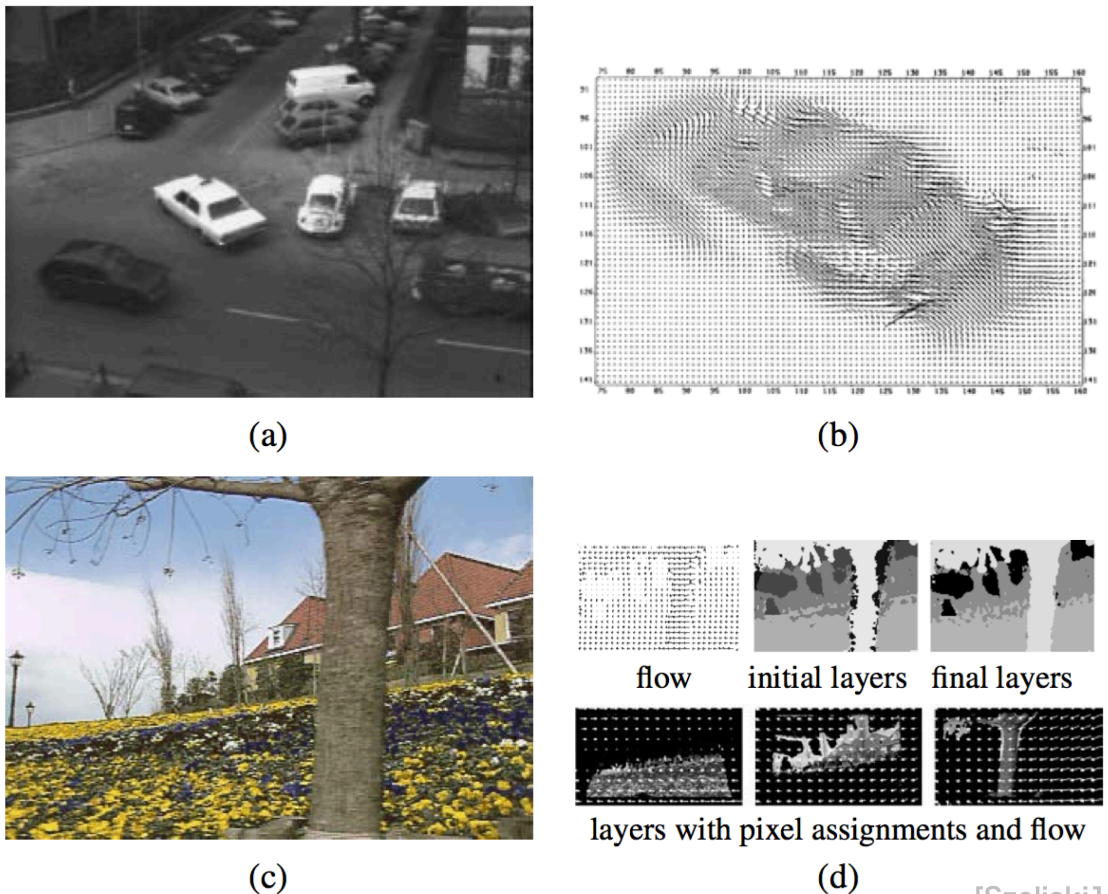
線形方程式：$\boldsymbol{A}\Delta\boldsymbol{u} = \boldsymbol{b}$

$$\boldsymbol{A} = \sum \boldsymbol{J}_1^T(\boldsymbol{x}_i+\boldsymbol{u})\boldsymbol{J}_1(\boldsymbol{x}_i+\boldsymbol{u})$$

$$\boldsymbol{b} = -\sum_i e_i \boldsymbol{J}_1^T(\boldsymbol{x}_i+\boldsymbol{u})$$

オプティカルフロー

- Optical flows = 画像上の動きベクトルが作る場
  - Brightness constancy equation: $I_x u + I_y v + I_t = 0$,



[Szeliski10]

Lucas-Kanade tracker: コード

lk_track.py

```
p0 = None
while 1:
    ret, frame = cap.read()
    frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    if p0 != None:
        p1, st, err = cv2.calcOpticalFlowPyrLK(old_gray, frame_gray,\
                                               p0, None, **lk_params)
        # Draw the flows
        for i in range(len(p0)):
            if st[i] == 1:
                a, b = p1[i,0,0], p1[i,0,1] #p1[i][0][0], p1[i][0][1]
                c, d = p0[i][0][0], p0[i][0][1]
                cv2.line(frame, (a,b), (c,d), (255,0,255), 2)
                cv2.circle(frame, (a,b), 5, (255,0,255), -1)

    cv2.imshow('frame', frame)
    if cv2.waitKey(300) & 0xff == 0x1b: # ESC
        break

    # Find good feature points to track
    old_gray = frame_gray.copy()
    p0 = cv2.goodFeaturesToTrack(old_gray, mask = None, **feature_params)
```

Tracking parametric motion

射影変換： $x' = \dfrac{(1+h_{00})x + h_{01}y + h_{02}}{h_{20}x + h_{21}y + 1}$ and $y' = \dfrac{h_{10}x + (1+h_{11})y + h_{12}}{h_{20}x + h_{21}y + 1}$.

$$\mathbf{H} = \begin{bmatrix} 1+h_{00} & h_{01} & h_{02} \\ h_{10} & 1+h_{11} & h_{12} \\ h_{20} & h_{21} & 1 \end{bmatrix}$$

$$\boldsymbol{p} = [h_{00}, h_{01}, h_{02}, h_{10}, h_{11}, h_{12}, h_{20}, h_{21}]^\top$$

$$\boldsymbol{J}_{\tilde{\boldsymbol{x}}} = \left.\frac{\partial \tilde{\boldsymbol{x}}}{\partial \boldsymbol{p}}\right|_{\boldsymbol{p}=0} = \begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -x^2 & -xy \\ 0 & 0 & 0 & x & y & 1 & -xy & -y^2 \end{bmatrix}$$

Forward compositional:

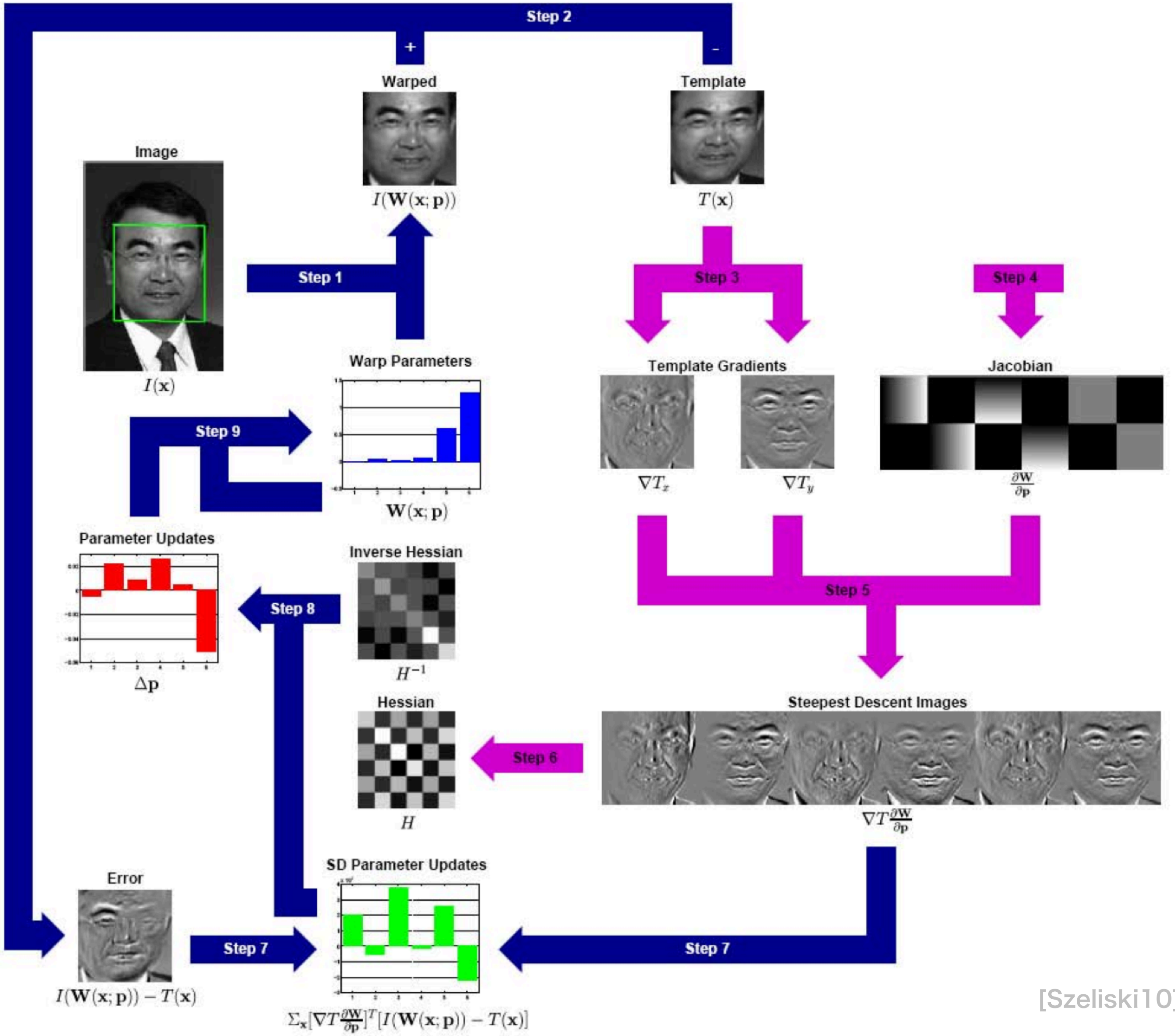
$$\begin{aligned} E_{\mathrm{LK-SS}}(\Delta\boldsymbol{p}) &= \sum_i [\tilde{I}_1(\tilde{\boldsymbol{x}}(\boldsymbol{x}_i; \Delta\boldsymbol{p})) - I_0(\boldsymbol{x}_i)]^2 \\ &\approx \sum_i [\tilde{J}_1(\boldsymbol{x}_i)\Delta\boldsymbol{p} + e_i]^2 \\ &= \sum_i [\nabla\tilde{I}_1(\boldsymbol{x}_i)\boldsymbol{J}_{\tilde{\boldsymbol{x}}}(\boldsymbol{x}_i)\Delta\boldsymbol{p} + e_i]^2 \end{aligned}$$

Inverse compositional:

$$\begin{aligned} E_{\mathrm{LK-BM}}(\Delta\boldsymbol{p}) &= \sum_i [\tilde{I}_1(\boldsymbol{x}_i) - I_0(\tilde{\boldsymbol{x}}(\boldsymbol{x}_i; \Delta\boldsymbol{p}))]^2 \\ &\approx \sum_i [\nabla I_0(\boldsymbol{x}_i)\boldsymbol{J}_{\tilde{\boldsymbol{x}}}(\boldsymbol{x}_i)\Delta\boldsymbol{p} - e_i]^2 \end{aligned}$$

更新式： $\boldsymbol{p} \leftarrow \boldsymbol{p} + \Delta\boldsymbol{p}$ $\qquad \tilde{I}_1(\boldsymbol{x}) = I_1(\boldsymbol{x}'(\boldsymbol{x}; \boldsymbol{p}))$.

Tracking parametric motion



[Szeliski10]