Readings

• Szeliski, R. Ch. 7
• Bergen et al. ECCV 92, pp. 237-252.
• Shi, J. and Tomasi, C. CVPR 94, pp.593-600.

• Slide Credits: Szeliski, Shah and B. Freeman
Recap: Estimating Optical Flow

- Assume the image intensity $I$ is constant

\[ I(x, y, t) = I(x + dx, y + dy, t + dt) \]
First Assumption: Brightness Constraint

\[ I(x, y, t) \approx I(x + dx, y + dy, t + dt) \]

\[ I(x(t) + u\Delta t, y(t) + v\Delta t) - I(x(t), y(t), t) \approx 0 \]

Assuming \( I \) is differentiable function, and expand the first term using Taylor’s series:

\[ \frac{\partial I}{\partial x} \frac{dx}{dt} + \frac{\partial I}{\partial y} \frac{dy}{dt} + \frac{\partial I}{\partial t} = 0 \]

Compact representation

\[ I_x u + I_y v + I_t = 0 \]

Brightness constancy constraint
**Second Assumption: Gradient Constraint**

Velocity vector is constant within a small neighborhood *(LUCAS AND KANADE)*

\[
E(u, v) = \int_{x, y} (I_x u + I_y v + I_t)^2 \, dx \, dy
\]

\[
\frac{\partial E(u, v)}{\partial u} = \frac{\partial E(u, v)}{\partial v} = 0
\]

\[
2(I_x u + I_y v + I_t)I_x = 0
\]

\[
2(I_x u + I_y v + I_t)I_y = 0
\]
Recap: Lucas-Kanade

\[
\begin{bmatrix}
\sum I_x^2 & \sum I_x I_y \\
\sum I_x I_y & \sum I_y^2
\end{bmatrix}
\begin{bmatrix}
u \\ v
\end{bmatrix}
= - \begin{bmatrix}
\sum I_x I_t \\
\sum I_y I_t
\end{bmatrix}
\]

Structural Tensor representation

\[
\begin{bmatrix}
T_{xx} & T_{xy} \\
T_{xy} & T_{yy}
\end{bmatrix}
\begin{bmatrix}
u \\ v
\end{bmatrix}
= - \begin{bmatrix}
T_{xt} \\
T_{yt}
\end{bmatrix}
\]

\[u = \frac{T_{yt} T_{xy} - T_{xt} T_{yy}}{T_{xx} T_{yy} - T_{xy}^2}\]
and
\[v = \frac{T_{xt} T_{xy} - T_{yt} T_{xx}}{T_{xx} T_{yy} - T_{xy}^2}\]
Pitfalls & Alternatives

- Brightness constancy is not satisfied
  - Correlation based method could be used
- A point may not move like its neighbors
  - Regularization based methods
- The motion may not be small (Taylor does not hold!)
  - Multi-scale estimation could be used
Multi-Scale Flow Estimation

Gaussian pyramid of image $I_t$

- $u=10$ pixels
- $u=5$ pixels
- $u=2.5$ pixels
- $u=1.25$ pixels

Gaussian pyramid of image $I_{t+1}$

- $u=10$ pixels

Lecture 10: Motion Models, Feature Tracking, and Alignment
Recap: Horn & Schunck

- Global method with smoothness constraint to solve aperture problem
- Minimize a global energy function

\[ E(u, v) = \int_{x,y} \left[ (I_x u + I_y v + I_t)^2 + \alpha^2 (|\nabla u|^2 + |\nabla v|^2) \right] dxdy \]

- Take partial derivatives w.r.t. \( u \) and \( v \):

\[
(I_x u + I_y v + I_t)I_x - \alpha^2 \nabla u = 0 \\
(I_x u + I_y v + I_t)I_y - \alpha^2 \nabla v = 0
\]
Global Motion Models (Parametric)

All pixels are considered to summarize global motion!

• **2D Models**
  – Affine
  – Quadratic
  – Planar projective (homography)

• **3D Models**
  – Inst. Camera motion models
  – Homography+epipole
  – Plane+parallax
Motion Models

Translation

Affine

Perspective

3D rotation

2 unknowns

6 unknowns

8 unknowns

3 unknowns
Example: Affine Motion

\[ u(x, y) = a_1 + a_2 x + a_3 y \]
\[ v(x, y) = a_4 + a_5 x + a_6 y \]

Each pixel provides 1 linear constraint in 6 global unknowns (a1,..,a6)
Locally Affine Motion Model

- Assume that $I(x,y,t)$ and $I(x,y,t-1)$ are two frames where affine motion occurred:
  - A simple way to estimate affine matrix parameters is to minimize quadratic error
  - Function $E(A)$ which can be defined as

\[
E(A) \approx \sum_{x,y \in D} [I(x,y,t) - I(x,y,t-1)]^2
\]

Where $A$ is a matrix consisting of affine parameters.

\[
A = \begin{bmatrix}
    a_1 & a_2 & a_3 \\
    a_4 & a_5 & a_6 \\
    0 & 0 & 1
\end{bmatrix}
\]
Locally Affine Motion Model

- $E(A)$ should be computed within a small neighborhood (i.e. locally affine term, let say $D$)
Locally Affine Motion Model

• $E(A)$ should be computed within a small neighborhood (i.e. locally affine term, let say $D$)

• From Taylor’s expansion
Locally Affine Motion Model

- E(A) should be computed within a small neighborhood (i.e. locally affine term, let say D)
- From Taylor’s expansion

\[ f(x) = P_n(x) + R_n(x), \]

\[ P_n(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \ldots \]
Locally Affine Motion Model

- $E(A)$ should be computed within a small neighborhood (i.e. locally affine term, let say D)
- From Taylor’s expansion

$$E(A) \approx \sum_{x,y \in D} \left[ I(x, y, t) - \left[ (I(x, y, t) + (a_1 + a_2 x + a_3 y - x)I_x(x, y, t) + (a_4 + a_5 x + a_6 y - y)I_y(x, y, t) - I_t(x, y, t)) \right]^2 \right]$$
Locally Affine Motion Model

- $E(A)$ should be computed within a small neighborhood (i.e. locally affine term, let say $D$)
- From Taylor’s expansion

$$E(A) \approx \sum_{x,y \in D} [I(x, y, t) - [(I(x, y, t) + (a_1 + a_2x + a_3y - x)I_x(x, y, t) +
(a_4 + a_5x + a_6y - y)I_y(x, y, t) - I_t(x, y, t))]^2$$

Further reduces to:

$$E(A) \approx \sum_{x,y \in D} [I_t(x, y, t) - (a_1 + a_2x + a_3y - x)I_x - (a_4 + a_5x + a_6y - y)I_y]^2$$
Locally Affine Motion Model

\[ E(A) \approx \sum_{x,y \in D} [I_t(x, y, t) - (a_1 + a_2 x + a_3 y - x)I_x - (a_4 + a_5 x + a_6 y - y)I_y]^2 \]
Locally Affine Motion Model

\[
E(A) \approx \sum_{x,y \in D} [I_t(x, y, t) - (a_1 + a_2x + a_3y - x)I_x - (a_4 + a_5x + a_6y - y)I_y]^2
\]

\[
k = I_t + xI_x + yI_y
\]

\[
c^T = (xI_x \ yI_y \ xI_y \ yI_x \ I_x \ I_y)
\]

\[
a = (a_1 \ a_2 \ a_3 \ a_4 \ a_5 \ a_6)
\]
Locally Affine Motion Model

\[ E(A) \approx \sum_{x,y \in D} [I_t(x, y, t) - (a_1 + a_2 x + a_3 y - x)I_x - (a_4 + a_5 x + a_6 y - y)I_y]^2 \]

\[ k = I_t + xI_x + yI_y \]

\[ c^T = (xI_x \ yI_y \ xI_y \ yI_x \ I_x \ I_y) \]

\[ a = (a_1 \ a_2 \ a_3 \ a_4 \ a_5 \ a_6) \]

\[ E(A) = \sum_{x,y \in D} [k - c^T a]^2 \]
Locally Affine Motion Model

\[ E(A) = \sum_{x,y \in D} [k - c^T a]^2 \]
Locally Affine Motion Model

\[ E(A) = \sum_{x,y \in D} [k - c^T a]^2 \]

\[ \frac{dE(A)}{da} = \sum_{x,y \in D} -2c[k - c^T a] \]
Locally Affine Motion Model

\[ E(A) = \sum_{x,y \in D} \left[ k - c^T a \right]^2 \]

\[ \frac{dE(A)}{da} = \sum_{x,y \in D} -2c \left[ k - c^T a \right] \]

\[ a = \left[ \sum_{x,y \in D} cc^T \right]^{-1} \left[ \sum_{x,y \in D} ck \right] \]
Smoothness Constraint on Locally Affine Motion Model

• We define a better error function by
Smoothness Constraint on Locally Affine Motion Model

• We define a better error function by

\[ E(A) = E_b(A) + E_s(A) \]
Smoothness Constraint on Locally Affine Motion Model

• We define a better error function by

\[ E(A) = E_b(A) + E_s(A) \]

\[ E_b(A) = \sum_{x,y \in D} [k - c^T a]^2 \]
Smoothness Constraint on Locally Affine Motion Model

- We define a better error function by

\[ E(A) = E_b(A) + E_s(A) \]

\[ E_b(A) = \sum_{x,y \in D} [(k - c^T a)^2] \]

\[ E_s(A) = \sum_{i=1}^{6} \lambda_i \left[ \left( \frac{\partial a_i}{\partial x} \right)^2 + \left( \frac{\partial a_i}{\partial y} \right)^2 \right] \]
Minimizing Error Function

\[
\frac{dE(A)}{da} = \frac{dE_b(A)}{da} + \frac{dE_s(A)}{da}
\]
Minimizing Error Function

\[
\frac{dE(A)}{da} = \frac{dE_b(A)}{da} + \frac{dE_s(A)}{da}
\]

\[
\frac{dE_b(A)}{da} = \sum_{x,y \in D} -2c[k - c^T a]
\]
Minimizing Error Function

\[
\frac{dE(A)}{da} = \frac{dE_b(A)}{da} + \frac{dE_s(A)}{da}
\]

\[
\frac{dE_b(A)}{da} = \sum_{x,y \in D} -2c[k - c^T a]
\]

\[
\frac{dE_s(A)}{da} = \sum_{x,y \in D} -2L(\bar{a} - a)
\]
Minimizing Error Function

\[
\frac{dE(A)}{da} = \frac{dE_b(A)}{da} + \frac{dE_s(A)}{da}
\]

\[
\frac{dE_b(A)}{da} = \sum_{x,y \in D} -2c[k - c^Ta]
\]

\[
\frac{dE_s(A)}{da} = \sum_{x,y \in D} -2L(\bar{a} - a)
\]

Where \(L\) is a 6 x 6 diagonal matrix with lambdas located at diagonals, and \(\bar{a}\) indicates average.
Minimizing Error Function-Solution Vector

\[ a = (cc^T + L)^{-1}(ck + L\bar{a}) \]
Global Motion

Estimate motion using all pixels in the image
Global Motion

Estimate motion using **all pixels** in the image

Global Motion can be used to
- Remove camera motion
- Object-based segmentation
- generate mosaics
Global Motion

Estimate motion using all pixels in the image

Global Motion can be used to
• Remove camera motion
• Object-based segmentation
• Generate mosaics
Recap: Object Tracking

- Track an object over a sequence of images
Challenges in Object Tracking

- Which features to track?
- Efficient tracking
- Appearance constraint violation
- ...

Challenges in Object Tracking

• Which features to track?
• Efficient tracking
• Appearance constraint violation
• ...

Shi-Tomasi Feature Tracker

• Good Features to Track
Shi-Tomasi Feature Tracker

• Good Features to Track
  – Find good features using eigenvalues of Hessian matrix (threshold on the smallest eigenvalue when computing Harris corner detection)
Shi-Tomasi Feature Tracker

• Good Features to Track
  – Find good features using eigenvalues of Hessian matrix (threshold on the smallest eigenvalue when computing Harris corner detection)
  – Track from frame to frame with LK
Shi-Tomasi Feature Tracker

• Good Features to Track
  – Find good features using eigenvalues of Hessian matrix (threshold on the smallest eigenvalue when computing Harris corner detection)
  – Track from frame to frame with LK
  – Check consistency of tracks by “affine registration” to the first observed instance of the feature
Shi-Tomasi Feature Tracker

Figure 1: Three frame details from Woody Allen’s *Manhattan*. The details are from the 1st, 11th, and 21st frames of a subsequence from the movie.

Figure 2: The traffic sign windows from frames 1, 6, 11, 16, 21 as tracked (top), and warped by the computed deformation matrices (bottom).

Shi and Tomasi CVPR 1994
Good Features To Track.
KLT Tracking

• KLT: Kanade-Lucas-Tomasi
KLT Tracking

- KLT: Kanade-Lucas-Tomasi
- Tracking deals with estimating the trajectory of an object in the image plane as it moves around a scene
KLT Tracking

- KLT: Kanade-Lucas-Tomasi
- **Tracking** deals with estimating the trajectory of an object in the image plane as it moves around a scene
- Object tracking (car, airplane, person)
KLT Tracking

• KLT: Kanade-Lucas-Tomasi

• Tracking deals with estimating the trajectory of an object in the image plane as it moves around a scene

• Object tracking (car, airplane, person)

• Feature tracking (Harris corners)
KLT Tracking

- KLT: Kanade-Lucas-Tomasi
- Tracking deals with estimating the trajectory of an object in the image plane as it moves around a scene
- Object tracking (car, airplane, person)
- Feature tracking (Harris corners)
- Multiple object tracking
KLT Tracking

• KLT: Kanade-Lucas-Tomasi

• Tracking deals with estimating the trajectory of an object in the image plane as it moves around a scene

• Object tracking (car, airplane, person)

• Feature tracking (Harris corners)

• Multiple object tracking

• Tracking in single/multiple camera(s)
KLT Tracking

• KLT: Kanade-Lucas-Tomasi
• Tracking deals with estimating the trajectory of an object in the image plane as it moves around a scene
• Object tracking (car, airplane, person)
• Feature tracking (Harris corners)
• Multiple object tracking
• Tracking in single/multiple camera(s)
• Tracking in fixed/moving camera
KLT Tracking Algorithm

• Find **GoodFeaturesToTrack**
  – Harris Corners (thresholded on smallest eigenvalues)

• Use LK algorithm to find optical flows

• Use Coarse-to-Fine strategy to deal with large movements

• When creating long tracks, check appearance of registered patch against appearance of initial patch to find points that have drifted
Recent Developments at Optical Flow

• Start with LK or similar methods
  + Gradient consistency
  + Energy minimization with smoothing term
  + Region matching
  + Keypoint matching

Large displacement optical flow, Brox et al., CVPR 2009
Very Recent Developments at Optical Flow

• Use of Machine Learning
  – Deep Learning (ICCV 2015, Fischer et al., FlowNet)
DeepFlow (Large Displacement Optical Flow)

• Basically it is a matching algorithm with variational approach [Weinzaepfel et al., ICCV 2013].

• Dense correspondence (matching)
• Self-smooth matching
• Large displacement optical flow
  – https://www.youtube.com/watch?v=k_wkDLJ8lJE
SIFT Tracking

Frame 0  →  Frame 100
Practice: Horn-Schunck Algorithm

1: for \( y = 1 \) to \( N_{rows} \) do
2:   for \( x = 1 \) to \( N_{cols} \) do
3:     Compute \( I_x(x, y) \), \( I_y(x, y) \), and \( I_t(x, y) \);
4:     Initialize \( u(x, y) \) and \( v(x, y) \) (in even arrays);
5:   end for
6: end for
7: Select weight factor \( \lambda \); select \( T > 1 \); set \( n = 1 \);
8: while \( n \leq T \) do
9:   for \( y = 1 \) to \( N_{rows} \) do
10:      for \( x = 1 \) to \( N_{cols} \) {in alternation for even or odd arrays} do
11:         Compute \( \alpha(x, y, n) \);
12:         Compute \( u(x, y) = \bar{u} - \alpha(x, y, n) \cdot I_x(x, y, t) \);
13:         Compute \( v(x, y) = \bar{v} - \alpha(x, y, n) \cdot I_y(x, y, t) \);
14:      end for
15:   end for
16:   \( n := n + 1 \);
17: end while
Practice: Horn-Schunck Algorithm

1: for $y = 1$ to $N_{\text{rows}}$ do
2:   for $x = 1$ to $N_{\text{cols}}$ do
3:     Compute $I_x(x, y)$, $I_y(x, y)$, and $I_t(x, y)$;
4:     Initialize $u(x, y)$ and $v(x, y)$ (in even arrays);
5:   end for
6: end for
7: Select weight factor $\lambda$; select $T > 1$; set $n = 1$;
8: while $n \leq T$ do
9:   for $y = 1$ to $N_{\text{rows}}$ do
0:     for $x = 1$ to $N_{\text{cols}}$ do [in alternation for even or odd arrays] do
1:       Compute $\alpha(x, y, n)$;
2:       Compute $u(x, y) = \bar{u} - \alpha(x, y, n) \cdot I_x(x, y, t)$;
3:       Compute $v(x, y) = \bar{v} - \alpha(x, y, n) \cdot I_y(x, y, t)$;
4:     end for
5:   end for
6: $n := n + 1$;
7: end while

$(I_xu + I_yv + I_t)I_x - \alpha^2 \nabla u = 0$
$(I_xu + I_yv + I_t)I_y - \alpha^2 \nabla v = 0$
### Practice: Horn-Schunck Algorithm

```plaintext
1: for y = 1 to \( N_{rows} \) do
2:     for x = 1 to \( N_{cols} \) do
3:         Compute \( I_x(x, y) \), \( I_y(x, y) \), and \( I_t(x, y) \);
4:         Initialize \( u(x, y) \) and \( v(x, y) \) (in even arrays);
5:     end for
6: end for
7: Select weight factor \( \lambda \); select \( T > 1 \); set \( n = 1 \);
8: while \( n \leq T \) do
9:     for y = 1 to \( N_{rows} \) do
0:         for x = 1 to \( N_{cols} \) {in alternation for even or odd arrays} do
1:             Compute \( \alpha(x, y, n) \);
2:             Compute \( u(x, y) = \bar{u} - \alpha(x, y, n) \cdot I_x(x, y, t) \);
3:             Compute \( v(x, y) = \bar{v} - \alpha(x, y, n) \cdot I_y(x, y, t) \);
4:         end for
5:     end for
6:     \( n := n + 1 \);
7: end while

\[
\begin{align*}
(I_x u + I_y v + I_t) I_x - \alpha^2 \nabla u &= 0 \\
(I_x u + I_y v + I_t) I_y - \alpha^2 \nabla v &= 0 \\

u^{k+1} &= \bar{u}^{k} - \frac{I_x (I_x \bar{u}^k + I_y \bar{v}^k + I_t)}{\alpha^2 + I_x^2 + I_y^2} \\
v^{k+1} &= \bar{v}^{k} - \frac{I_y (I_x \bar{u}^k + I_y \bar{v}^k + I_t)}{\alpha^2 + I_x^2 + I_y^2}
\end{align*}
\]

\[
\alpha(x, y, n) = \frac{I_x(x, y) \bar{u}^{n}_{xy} + I_y(x, y) \bar{v}^{n}_{xy} + I_t(x, y)}{\lambda^2 + I_x^2(x, y) + I_y^2(x, y)}
\]
```
Optical Flow - Quantitative Evaluation

\[ E_{ep2} = \sqrt{(u - u^*)^2 + (v - v^*)^2} \]

\[ E_{ep1} = |u - u^*| + |v - v^*| \]

- Where \( u=(u,v) \) is computed, \( u=(u^*,v^*) \) ground truth velocity vectors.

\[ E_{ang} = \arccos \left( \frac{u^T u^*}{|u||u^*|} \right) \]
Interpretation of Optical Flow Fields
Interpretation of Optical Flow Fields

Object features all have Zero velocity.
Interpretation of Optical Flow Fields
Interpretation of Optical Flow Fields

Object is moving to the Right.
Interpretation of Optical Flow Fields
Interpretation of Optical Flow Fields

Camera is moving into the scene, and an object moving passed the camera
Interpretation of Optical Flow Fields
Object is moving directly toward the camera that is stationary.
Interpretation of Optical Flow Fields
Interpretation of Optical Flow Fields

Object is rotating about the line of sight to the camera
Interpretation of Optical Flow Fields
Interpretation of Optical Flow Fields

Object is rotating about an axis perpendicular to the line of sight.
Application in Image Alignment

- Motion can be used for image alignment

Pixel locations at time $t+1$

\[
\begin{bmatrix}
    x' \\
    y'
\end{bmatrix} =
\begin{bmatrix}
    a & b \\
    c & d
\end{bmatrix} \begin{bmatrix}
    x \\
    y
\end{bmatrix}
\]

Pixel locations at time $t$
**Practice: Homogenous Coordinates**

Q: How can we represent translation as a 3x3 matrix?

\[
x' = x + t_x
\]
\[
y' = y + t_y
\]
Q: How can we represent translation as a 3x3 matrix?

\[ x' = x + t_x \]
\[ y' = y + t_y \]

Translation = \[
\begin{bmatrix}
1 & 0 & t_x \\
0 & 1 & t_y \\
0 & 0 & 1
\end{bmatrix}
\]
Practice: Homogenous Coordinates

Q: How can we represent translation as a 3x3 matrix?

\[
x' = x + t_x \\
y' = y + t_y
\]

\[
\begin{bmatrix}
1 & 0 & t_x \\
0 & 1 & t_y \\
0 & 0 & 1
\end{bmatrix}
\]

Translation

\[
\begin{bmatrix}
x' \\
y' \\
1
\end{bmatrix} = \begin{bmatrix}
1 & 0 & t_x \\
0 & 1 & t_y \\
0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
x \\
y \\
1
\end{bmatrix} = \begin{bmatrix}
x + t_x \\
y + t_y \\
1
\end{bmatrix}
\]
**Practice:** Basic 2D Transformations

Translation:

\[
\begin{bmatrix}
  x' \\
  y' \\
  1
\end{bmatrix} =
\begin{bmatrix}
  1 & 0 & t_x \\
  0 & 1 & t_y \\
  0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
  x \\
  y \\
  1
\end{bmatrix}
\]

Rotation:

\[
\begin{bmatrix}
  x' \\
  y' \\
  1
\end{bmatrix} =
\begin{bmatrix}
  \cos \Theta & -\sin \Theta & 0 \\
  \sin \Theta & \cos \Theta & 0 \\
  0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
  x \\
  y \\
  1
\end{bmatrix}
\]

Scaling:

\[
\begin{bmatrix}
  x' \\
  y' \\
  1
\end{bmatrix} =
\begin{bmatrix}
  s_x & 0 & 0 \\
  0 & s_y & 0 \\
  0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
  x \\
  y \\
  1
\end{bmatrix}
\]

Shearing:

\[
\begin{bmatrix}
  x' \\
  y' \\
  1
\end{bmatrix} =
\begin{bmatrix}
  1 & sh_x & 0 \\
  sh_y & 1 & 0 \\
  0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
  x \\
  y \\
  1
\end{bmatrix}
\]
Affine Transformation

- Affine transformations are combinations of ...
  - Linear transformations, and
  - Translations
- Properties of affine transformations:
  - Origin does not necessarily map to origin
  - Lines map to lines
  - Parallel lines remain parallel
  - Ratios are preserved
  - Closed under composition
  - Models change of basis

\[
\begin{bmatrix}
x' \\
y' \\
w'
\end{bmatrix} = \begin{bmatrix}
a & b & c \\
d & e & f \\
0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
x \\
y \\
w
\end{bmatrix}
\]

\[
\begin{bmatrix}
x' \\
y' \\
w'
\end{bmatrix} = \begin{bmatrix}
a & b & c \\
d & e & f \\
g & h & i
\end{bmatrix} \begin{bmatrix}
x \\
y \\
w
\end{bmatrix}
\]

projective
Affine Transformation

\[
\begin{bmatrix}
x' \\
y' \\
w'
\end{bmatrix} = \begin{bmatrix}
a & b & c \\
d & e & f \\
0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
x \\
y \\
w
\end{bmatrix}
\]

Affine matrix decomposition

**Translation + rotation + scaling**

\[
\begin{bmatrix}
x' \\
y' \\
w'
\end{bmatrix} = \begin{bmatrix}
1 & 0 & tx \\
0 & 1 & ty \\
0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
\cos \Theta & -\sin \Theta & 0 \\
\sin \Theta & \cos \Theta & 0 \\
0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
sx & 0 & 0 \\
0 & sy & 0 \\
0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
x \\
y \\
w
\end{bmatrix}
\]

\[p' = T(t_x, t_y) \quad R(\Theta) \quad S(s_x, s_y) \quad p\]
Reminders

• Choose your mini-project until **15th October**.
  – At most, 3 people can select the same project.
  – You can come up with your own project ideas too.

• **CVPR 2015/2016 – Search Papers, Find Data Set, and implement it!**
  • KLT Object Tracking with Python
  • Interactive Segmentation of Images (Thresholding) as Smart Phone Application (with Processing language)
  • Canny Edge Detection as Smart Phone Application (with Processing language)
  • Locally Affine Motion Model for Image Registration Application in 3D (C/C++ or Python Implementation only).
  • 3D Graph-Cut Segmentation
  • 2D Conditional Random Field based image segmentation
  • Action Recognition (sports) with Fisher Vectors
  • Image Stitching for Smart Phone Applications
  • Real Time Face Detection for Android Applications
  • 3D Edge Detection for Surface Reconstruction
  • Anisotropic Diffusion Filtering (Perona/Malik)
  • SIFT-Flow for image registration (See: [http://people.csail.mit.edu/celiu/pdfs/SIFTflow.pdf](http://people.csail.mit.edu/celiu/pdfs/SIFTflow.pdf))