Lecture 9:
Marker-Based Positioning and Pattern Registration

Adding Data Association to Vision Sensor Block Diagram

Today we discuss Positioning without Segmentation:
Today’s Summary

1. Geometric Image Transformations

2. Navigation and Positioning Using Multiple Image Points
   - Pattern Registration (Correlation)
   - Point-Based Positioning

Geometric Image Transformations

- In Lecture 7, we discussed linear and nonlinear transformations for image smoothing (and sharpening)
- Another class of operation that can be applied to an image is the geometric transformation. Examples include:
  - Rotation (Linear)
  - Scaling (Linear)
  - Translation (Nonlinear - Affine)
  - Perspective (Nonlinear)
  - Radial Distortion (Nonlinear)
- These geometric distortions are the basis for the two navigation strategies we’ll discuss today
Image Rotation

- Image rotation implies camera motion about its optical axis
- Matlab command: `imrotate`

\[
\text{rot}\_\text{im} = \text{imrotate} (\text{base}\_\text{im}, 30, \text{‘bilinear’});
\]

Image Scaling

- Image scaling implies camera motion along its optical axis
- Matlab command: `imresize`

\[
\text{scale}\_\text{im} = \text{imresize} (\text{base}\_\text{im}, 0.4, \text{‘bicubic’});
\]
**Image Translation**

- Image scaling implies camera motion normal to optical axis
- Matlab command: `imtransform`
  
  ```matlab
  %translation is an affine transform:  \[x', y'] = [x, y] \cdot A + B
  A = [1 0; 0 1];  % linear transform: scaling, rotation, shearing
  B = [350.3 0];  % B is translation vector - lateral pixel shift
  T = maketform('affine', [A; B]);
  translated_im = imtransform(base_im, T, 'bilinear',...
     'XData', [1 size(qq,2)], 'YData', [1 size(qq,1)]);  %see below
  ```

**Image Perspective**

- Image scaling implies camera motion normal to optical axis
- Matlab command: `imtransform`
  
  ```matlab
  %Projective transform defined as operation on image corners
  ht = size(qq,1); wd = size(qq,2);  %size of base image
  b_corner = [0 0; wd 0; 0 ht; wd ht];  %base image corners
  t_corner = [0 0; wd/4 ht; wd*3/4 ht];  %transformed corners
  T = maketform('projective', b_corner, t_corner);
  perspective_im = imtransform(base_im, T, 'bilinear');
  ```
Geometric Transformation: Interpolation

Geometric transformations are operations on image pixel coordinates

- If the Image Processing Toolbox functions are unavailable, or if no function exists for a particular transformation, you can still use interpolation
- Example: Rotation

```matlab
%rotate monochrome image by 30° using bilinear interpolation
maxX = size(base_im,2); x = 1:1:maxX;
maxY = size(base_im,1); y = 1:1:maxY;
[Xold,Yold] = meshgrid(x,y); %Coordinates for old image
Xnew = Xold; Ynew = Yold; %Coordinates for new image
ang = - 30 / 180 * pi; %radians %Negative of Rotation Angle
Xintp = cos(ang)*(Xnew - maxX/2) + sin(ang)*(Ynew - maxY/2) + maxX/2;
Yintp = -sin(ang)*(Xnew - maxX/2) + cos(ang)*(Ynew - maxY/2) + maxY/2;
Xbase(nodata) = 1; Ybase(nodata) = 1; %Fix out-of-range coordinates
rot_im = interp2(Xold,Yold,base_im,Xintp,Yintp,'linear');
rot_im(nodata) = 0;
```

Radial Distortion

- Radial Distortion correction can remove lens aberration
- Matlab command: `interp2`

```matlab
%Distortion transform based on radial stretching
Rintp = sqrt((Xnew-maxX/2).^2 + (Ynew-maxY/2).^2);
Xnew = (Xnew-maxX/2).*((1 + .0005*Rintp.^2 + 2e-8*Rintp.^2) - maxX/2);
Ynew = (Ynew-maxY/2).*((1 + .0005*Rintp.^2 + 2e-8*Rintp.^2) - maxY/2);
Xbase(nodata) = 1; Ybase(nodata) = 1; %Fix out-of-range coordinates
for rgb = 1:3
    temp_im(:,:,rgb) = interp2(Xold,Yold,base_im(:,:,rgb),Xintp,Yintp);
    dist_im(:,:,rgb) = 0;
end
```
Registration

- **Registration** is the process of finding a geometric transformation that aligns two images
- Registration can be used for navigation
  - The geometric transformation describes camera motion between snapping each of two images
  - Restriction: registration-based navigation is most useful when observing a planar (or nearly planar) surface, such as the ground
  - Advantage: registration can be performed using image texture, only, and thus no markers are required

Registration-Based Navigation

- The basic concept behind registration-based navigation is iterative application of geometric transformations until alignment is achieved
- In order to evaluate “good” alignment, it is necessary to define a metric for the similarity of images
- Two common image similarity metrics are
  
  **Correlation:**
  \[
  C = \sum_{i=1}^{N_{\text{rows}}} \sum_{j=1}^{N_{\text{cols}}} (A_{i,j} - \bar{A})(B_{i,j} - \bar{B})
  \]

  **Sum-of-Squared Differences (SSD):**
  \[
  SSD = \sum_{i=1}^{N_{\text{rows}}} \sum_{j=1}^{N_{\text{cols}}} (A_{i,j} - B_{i,j})^2
  \]

  Bar indicates the mean image value over all pixels in correlation window
Correlation Example: Translation

- **Window**: A subregion of one image (window) is typically used in computing the metric. This keeps the total number of pixels in the summation constant for both correlation and SSD.

SSD Example: Translation

- **Window**: A subregion of one image (window) is typically used in computing the metric. This keeps the total number of pixels in the summation constant for both correlation and SSD.
Removing Shadows

- The previous example was dominated by shadow, rather than local “texture”
  - For navigation purposes, locking onto shadows may be dangerous
  - Shadows often move with the camera/robotic platform
- A good way to reduce shadow and lighting effects is with a high pass filter
  - Second derivative (Laplacian) filter kernel removes local brightness and local gradients in brightness
  - Derivatives increase image noise, so we should also add in smoothing filter (such as Gaussian)
  - Since both filters are linear, they can be applied together in superposition (Laplacian of Gaussian, LoG)

LoG Filter

- Convolve Laplacian and Gaussian kernels to get LoG kernel
  - i.e. LoG = conv2(L, G);
  
- There is no unique form for Gaussian filter; a particular 3x3 implementation is chosen here
Correlation Example (LoG Filter)

- Apply LoG filter to both image1 and image2 using the “conv2" routine in Matlab, as we learned in Lesson 7

Correlation Comparison (Unfiltered)

- Filtered correlation has several advantages over unfiltered
  - Shadow removal
  - More precise (sharper correlation peak)
  - Better ambiguity resolution (weaker secondary peaks)
Convolution and Correlation

- The last example used convolution (filtering) and correlation (registration) operations
  - Convolution and Correlation are mathematically related
  - **Convolution** is a “flip and drag” operation (Recall Lecture 7)
  - **Correlation** is simply a “drag” operation

- Similar Matlab commands exist for both operations
  - Convolution
    \[ y(m,n) = \sum_{j=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} h(m-j, n-k) x(j,k) \]
  - Cross-Correlation
    \[ y(m,n) = \sum_{j=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} h(j-m, k-n) x(j,k) \]

Other Geometric Transforms

- Translation is the most efficient geometric transformation to implement
- Rotation, scaling, and other operations are also possible
  - These operations are significantly more computationally expensive
  - The basic concept is identical to translation:
    - Discretize the space of possible geometric transforms
    - Generate the transformed correlation window
    - Check the value of the correlation (or SSD) function for each possible geometric transform
    - The best-case metric (minimum SSD or maximum correlation) indicates the most likely image transformation

- If other sensors are available (compass, inclinometers, sonar ranging devices), a much smaller space of geometric transforms can be searched, improving computational speed
Point-Based Positioning

- An alternate technique for computing camera position changes is **Point-Based Positioning**
  - This technique has high computational efficiency
  - This technique relies on the identification of point features (markers) that exhibit rigid motion between subsequent images.

Procedure for Point-Based Positioning

- Fundamentally, Point-Based Positioning consists of three steps
  1. Marker Identification (Week 4): Find point markers in image
  2. Data Association (Week 8): Correspond point markers between two images
  3. Geometry: Infer camera translation and orientation changes given changes in marker position – similar to **Calibration** (Week 3)
Point-Based Positioning vs. Pattern Registration

- **Point-Based Positioning**
  - Method uses only a few marker points, so it’s more computationally efficient than pattern registration
  - In principle, this method computes geometry changes given measurements of any 3D, rigidly attached point cloud (in contrast with pattern registration, which requires a 2D or nearly 2D surface)

- **Pattern Registration**
  - Method uses entire image, so it’s more resistant to noise and to partial occlusion
  - Method can use texture-based filtering to remove shadows

Special Case: Planar Positioning

- In general, a 3D state can be determined using point-based positioning
  - 3D case solves for rotation matrix and translation vector that describe camera’s relative position from image 1 to image 2
  - All 6 DoF can be computed, but an arbitrary scaling factor remains (all point position measurements could be multiplied by a constant without changing the appearance of the images)
  - The book discusses this general procedure as a Calibration method

- For the purposes of this class, we will focus on state determination for the special case of motion aligned with a 2D plane
Planar Rigid Motion

- If a camera is constrained to point orthogonal to a surface, only 4 degrees of freedom remain (three translational and one rotation)
- Point-based positioning can be used to solve this constrained geometric transformation, up to an arbitrary scaling factor

Analytic Geometry

$$\vec{P} = \begin{bmatrix} x \\ y \end{bmatrix}$$

Each measured point can be described with two coordinates in the image

$$R = \begin{bmatrix} \cos(\psi) & \sin(\psi) \\ -\sin(\psi) & \cos(\psi) \end{bmatrix}$$

Rotation matrix and translation vector (pixel units) that describe the camera rotational and translational motion between time 0 and time 1

$$\alpha = \frac{z - t_z}{z}$$

Scaling describes camera motion normal to plane

$$\vec{P}_1 = \alpha R (\vec{P}_0 - \vec{t})$$
Solution to 2D Motion Problem

\[ x_i = \alpha \cos(\psi)(x_0 - t_x) + \alpha \sin(\psi)(y_0 - t_y) \]
\[ y_i = -\alpha \sin(\psi)(x_0 - t_x) + \alpha \cos(\psi)(y_0 - t_y) \]

- We must solve for 4 unknowns, using 2 equations \((x, y)\) per each marker
  - In theory, we need only 2 markers
  - Additional markers reduce sensitivity to measurement and data-association errors
- Use Least-Squares solution

Least-Squares (LSQ) Solution

Single-Point Equations

\[ x_i = \alpha \cos(\psi)x_0 + \alpha \sin(\psi)y_0 + \alpha (-t_x \cos(\psi) - t_y \sin(\psi)) \]
\[ y_i = -\alpha \sin(\psi)x_0 + \alpha \cos(\psi)y_0 + \alpha (-t_x \sin(\psi) - t_y \cos(\psi)) \]

Multi-Point LSQ Equations

\[
\begin{bmatrix}
  x_1 & y_1 & 1 & 0 \\
  x_2 & y_2 & 1 & 0 \\
  \vdots & \vdots & \vdots & \vdots \\
  x_n & y_n & 1 & 0 \\
\end{bmatrix}
= 
\begin{bmatrix}
  x_{10} & y_{10} & 1 & 0 \\
  x_{20} & y_{20} & 1 & 0 \\
  \vdots & \vdots & \vdots & \vdots \\
  x_{N0} & y_{N0} & 1 & 0 \\
\end{bmatrix}
\]

Leading subscript for each \((x, y)\) measurement indicates the index of the marker (assuming a total of \(N\) markers)
Deriving Motion Parameters

\[ \alpha = \sqrt{\beta_1^2 + \beta_2^2} \]

\[ \psi = \text{atan} \left( \frac{-\beta_2}{\beta_1} \right) \]

\[ t_x = \frac{1}{2} \left( -\cos(\psi) \beta_2 + \sin(\psi) \beta_1 \right) \]

\[ t_y = -\frac{1}{2} \left( \sin(\psi) \beta_2 + \cos(\psi) \beta_1 \right) \]

Units of Pixels

If we know the camera field of view angle, FOV, and the initial distance, \( \rho \) (in meters), between the camera and the planar surface for the first image, we can convert \( t_x \), \( t_y \) and \( t_z \) from pixel units to dimensional units (i.e. meters)

\[ \Delta \hat{T} = \begin{bmatrix} t_x \tan \left( \frac{1}{2} \text{FOV} \right) \rho / P_x \\ t_y \tan \left( \frac{1}{2} \text{FOV} \right) \rho / P_y \\ (\alpha - 1) \rho \end{bmatrix} \]

Dimensional, Same Units as \( R \)

\( P_x \) and \( P_y \) are the pixel half-widths of the image

Complete Algorithm

- The complete Point-Based Positioning Algorithm combines marker extraction, data association and a camera geometry solution
  - Marker Extraction: Use eigenvalue method
  - Data Association: Use nearest neighbor in pixel (x, y) space
  - Geometry Solution: Use rigid planar constraint and LSQ method
Example

1. Filter with uniform 5x5 kernel
2. Find markers using eigenvalues over 5x5 window
3. Iteratively associate points starting with best match (x,y nearest neighbor); in subsequent association steps, exclude markers that have been previously associated

Result

- LSQ Gives:

\[
\begin{align*}
\alpha &= 1.02 \\
\psi &= 2.6^\circ \\
t_x &= -3.1 \text{ pixel} \\
t_y &= -2.8 \text{ pixel}
\end{align*}
\]

Abs(Image2 – Image1)  
Abs(Image2 – Register(Image1))

Bright denotes large difference
Roll-Pitch Observability

- Pitch and roll deviations are not observable
- For rigidly attached points on a planar surface, deviations of camera axis away from orthogonality (pitch or roll angles nonzero) look like small translations
- Full 6 DoF geometry solution is not possible for this case (4 DoF is the best we can do)
- Sensor fusion may be necessary (i.e. inclinometers to measure pitch and roll)

Related Problems in Robotics

- Structure from Motion
- Mosaic Mapping
- Recognition

Photo Mosaic, Courtesy of Monterey Bay Aquarium Research Institute and Stanford Aerospace Robotics Lab
Review

1. Geometric Image Transformations
   • Solve for new image given geometry transformation

2. Registration for Navigation
   • Solve for geometry transformation given two images
   • Correlation, Sum-of-Squared Differences
   • Pattern Registration (Correlation)

3. Point-Based Positioning
   • For faster computation, solve for geometry transformation between sets of points (markers) in two images