Colorado School of Mines

Computer Vision

Professor William Hoff
Dept of Electrical Engineering & Computer Science

http://inside.mines.edu/~whoff/
Bundle Adjustment
Bundle Adjustment

• Recall
  – We can compute the essential matrix E from two views from a moving camera
  – From E, we can estimate the point locations (“structure”) and the poses of the camera (“motion”), up to a scale factor

• However, the results can be noisy!

• The most accurate method (the “gold standard”) is bundle adjustment.
  – It gives the Maximum Likelihood solution to the problem, that is a solution that is optimal for the inaccuracies of image measurements.

• Problem:
  – It is an iterative process, which needs a good starting guess
  – The essential matrix solution can provide this guess
Bundle Adjustment

• We estimate structure and motion that directly minimizes the squared reprojection errors for the 2D points

• We can do this for N camera poses (N≥2)

• The unknowns are:
  – The (X,Y,Z) positions of the K 3D points
  – The relative poses of the camera in N-1 poses
    • The first pose is arbitrary and we can just set it to the origin
  – There are a lot of unknown parameters to find!

• We can use non-linear least squares
  – It will be slow ... probably not suitable for real-time
  – We will need a good starting guess
Re Projection Error

Objective function:

$$g(C, X) = \sum_{i=1}^{n} \sum_{j=1}^{m} w_{ij} \|q_{ij} - P(C_i, X_j)\|^2$$

Indicator variable:

1 if point $j$ is visible in camera $i$
0 otherwise
Example Application

- A vehicle needs to map its environment that it is moving through, and estimate its self motion
- The main sensors are cameras mounted on the vehicle
Structure from Motion

• Parameters to be estimated
  – \( N-1 \) vehicle poses wr.t. 1\(^{st} \) vehicle pose; each has 6 degrees of freedom
  – \( K \) 3D point locations wr.t. 1\(^{st} \) vehicle pose

• Measurements
  – Image point observations
    \[ \mathbf{p}_{ij} = (u \ v)^T, \quad i = 1..N, \quad j = 1..K \]
  – Assumptions
    • Calibrated cameras
    • Point correspondences are known
Bundle Adjustment

• We need a function that
  – Predicts the image locations of the K 3D points, in each of the N images
  – The inputs to the function are the current guess for the locations of the 3D points, and the N-1 poses

• As we did before, we can take the Jacobian of the function and iteratively add corrections to our guess

• Note - there are somewhat more efficient algorithms than the Newton’s method that we used
  – One is called “Levenberg-Marquardt”
    • It adaptively switches between Newton’s method and steepest descent
  – It is implemented in Matlab’s “lsqnonlin” function
    • This is part of the optimization toolbox
Scale Factor

• Results have an unknown scale factor
  – All points, and all translations are scaled by the same unknown amount

• Possible ways to determine scale factor:
  – Take an picture of a known object in one of the images
  – Use stereo cameras
  – Use some other data source, such as odometry

• Odometry
  – Many vehicles can estimate self-motion from wheel encoders and heading sensors
  – It is fairly accurate over short distances, but can drift over longer distances
Structure from Motion

• Algorithm approach
  – Search for parameters to minimize residual errors between predictions and measurements

\[
\text{minimize } E = \sum_{i} \left\| v_{i+1} \beta_{odom} - f\left( v_{i-1} \beta, v_{i} \beta \right) \right\|^{2}_{\Omega_{i}} + \sum_{i,j} \left\| p_{ij} - g\left( P_{j}, v_{i} \beta \right) \right\|^{2}_{\Xi_{j}}
\]

  – where

\[
\left\| y \right\|^{2}_{\Sigma} = y^{T} \Sigma^{-1} y
\]

• Nonlinear least squares problem
  – Levenberg Marquardt algorithm
  – “bundle adjust”
Minimization Algorithm

- Bundle adjust is computationally expensive
- Limit # poses, # points
- Use a sliding window of last N poses
  - Solutions are with respect to the first pose of this set
  - Effectively, mapping is done w.r.t. a moving local coordinate system

- Also, a good initial guess is needed
  - Pose guesses come from odometry
  - Point locations are initialized using triangulation
Mapping Program Control Flow

do

Get next set of measurements (images and odometry)
Predict pose of vehicle from odometry
Track existing points
Eliminate inactive points
If predicted distance travelled is at least $D$ meters
    Define this pose as a “keyframe”
    Try to acquire new points to track
    Try to initialize 3D locations of 2D points using triangulation
    Do bundle adjust algorithm over last 7 keyframes

while measurements are available

Note: $D = 1.5..3.0$ m, depending on vehicle speed
Tracking Videos

Scene: roxborough1

- Squares: tracked points
- Red: 2D information only
- Orange: Triangulation avail
- Yellow: 3D position estimate
- Blue bar: uncertainty est
- Red text: can’t compute pose

Colorado School of Mines

Computer Vision
Scene Animations

Scene: roxborough1

- Yellow: tracked point
- Blue: no longer tracked (archived)
- Ellipsoids = 1 sigma point uncertainty

Orbital View

Top-down View
Mead Way

- 1.1 km
- 660 keyframes
- 3833 point landmarks

*Colorado School of Mines*  
*Computer Vision*
Roxborough Road

- 4.2 km
- 2403 keyframes
- 5790 point landmarks

Figure 29 Roxborough Road site, showing mapped points in the zoomed in portion. Each pixel is 13.7 meters.
Rampart Range Road

• 4.5 km
• 5268 keyframes
• 24508 point landmarks

Figure 32 Rampart Range Road site. Each pixel is 7.1 meters.
Matlab Example (two views)

• Take the pavilion example we did earlier

![Pavilion Examples](image1.jpg) ![Pavilion Examples](image2.jpg)

• Add this to the bottom of the code:

```matlab
% Save results.
save('pavillionResults', ...
    'pts1', 'pts2', ...
    'points3D', ...
    'R_c1_c2', ...
    't2org_c1', ...
    'K');

% File name
% Image points
% Mx3 point locations in 3D
% Rotation of c1 wrt c2
% translation of c2 wrt c1
% Camera intrinsic matrix
```
Useful Matlab function

- **bundleAdjustment()**

  ```matlab
  [refinedPoints,refinedPoses] = bundleAdjustment( ...  
  points3D, ... % Input 3D points (N x 3)  
  pointTracks, ... % An array of "pointTrack" objects  
  cameraPoses, ... % A "table" of camera poses  
  cameraParams, ... % The intrinsic camera parameters  
  'Verbose', true);
  ```

- A “pointTrack” object has
  - A list of view id numbers where the point was seen
  - The 2D image locations of the points

- “cameraPoses” is a table with these entries for each view
  - Orientation (a 3x3 rotation matrix)
  - Location (a 1x3 translation vector)
clear all
close all

% Load previous results:
%    'pts1', 'pts2', ...             % Image points
%    'points3D', ...                 % Mx3 point locations in 3D
%    'R_c1_c2', ...                  % Rotation of c1 wrt c2
%    't2org_c1', ...                 % translation of c2 wrt c1
%    'K'                             % Camera intrinsic matrix
load('pavillionResults.mat');

% Initialize the "cameraParameters" object.
cameraParams = cameraParameters('IntrinsicMatrix', K');

% Make a "table" of camera poses.
ViewId = uint32([1;2]);
Orientation = { eye(3,3); R_c1_c2 };% Make a "table" of camera poses.
Location = { zeros(1,3); t2org_c1 };% Make an array of "pointTrack" objects.
cameraPoses = table(ViewId, Orientation, Location);

Npts = size(points3D,1);pointTracks = [];for i=1:Npts
    pointTracks = [pointTracks pointTrack([1,2], [pts1(i).Location; pts2(i).Location])];
end

% Do bundle adjustment.
[refinedPoints,refinedPoses] = bundleAdjustment( ...% Display results.
points3D, ...            % Input 3D points (N x 3)
pointTracks, ...         % An array of "pointTrack" objects
cameraPoses, ...         % A "table" of camera poses
cameraParams, ...        % The intrinsic camera parameters
'Verbose', true);% Output orientation:'), disp(refinedPoses.Orientation{2});
disp('Input location:'), disp(refinedPoses.Location{2});
disp('Output location:'), disp(refinedPoses.Location(2));

Bundle adjust code (1 of 2)
% Visualize the scene in 3D.
ptCloud = pointCloud(refinedPoints); % Create the point cloud

% Visualize the camera locations and orientations
cameraSize = 0.3;
figure
plotCamera('Size', cameraSize, 'Color', 'r', 'Label', '1', 'Opacity', 0);
hold on
grid on
R = refinedPoses.Orientation{2};
t = refinedPoses.Location{2};
plotCamera('Location', t, 'Orientation', R, 'Size', cameraSize, ...
    'Color', 'b', 'Label', '2', 'Opacity', 0);

% Visualize the point cloud
pcshow(ptCloud, 'VerticalAxis', 'y', 'VerticalAxisDir', 'down', ...
    'MarkerSize', 45);

% Rotate and zoom the plot
camorbit(0, -30);
camzoom(1.5);

% Label the axes
xlabel('x-axis');
ylabel('y-axis');
zlabel('z-axis');

Bundle adjust
code (2 of 2)