Bundle Adjustment

COSC 470: Special Topic
Computer Vision | 3D Reconstruction
Steven Mills

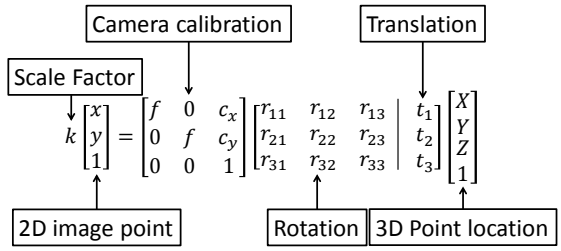
Bundle Adjustment

- At several points we have said that the proper solution is a non-linear optimisation problem
  - Homography estimation
  - Fundamental matrix estimation
  - 3D point reconstruction by triangulation
  - Pose estimation
- For the full reconstruction task this is called bundle adjustment

Bundle Adjustment

- We have estimates (from linear methods) of
  - Camera calibration parameters
  - Camera orientation and position
  - 3D point locations
- We also have some measurements
  - 2D image locations
- These are related by non-linear equations

The Projection Equations



Nonlinearities

- Scale factor means we divide the first and second elements on the left by the third
- Rotation matrix has 9 elements but only 3 degrees of freedom
  - Using Euler angles this gives combinations of trigonometric parameters
  - Axis-angle representation also non-linear but has more uniform properties

Measurements and Unknowns

- Measurements
  - Two values – *x* and *y*
- Unknowns
  - (at least) 3 camera calibration parameters
  - 6 camera pose parameters
  - 3 point location parameters
- Multiple views of the same point give us more measurements than unknowns

The Projection Equations

- Suppose we have *n* cameras and *m* points
- Projection of the i^{th} point in the j^{th} camera is:

$$k\begin{bmatrix} x_i^j \\ y_i^j \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix} \left[\begin{array}{ccc|c} r_{11}^j & r_{12}^j & r_{13}^j & t_1^j \\ r_{21}^j & r_{22}^j & r_{23}^j & t_2^j \\ r_{31}^j & r_{32}^j & r_{33}^j & t_3^j \end{array}\right] \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix}$$

- Here we assume one moving camera so just one calibration matrix

Measurements and Unknowns

- If each point ($1 < i < m$) is visible in every camera ($1 < j < n$) we have:
  - $2nm$ measurements
  - $4 + 6n + 3m$ unknowns
- Example: 9 points in 3 cameras gives
  - $2 \times 9 \times 3 = 54$ measurements
  - $3 + 6 \times 3 + 3 \times 9 = 3 + 18 + 27 = 48$ unknowns
- As long as more measurements than unknowns, we can solve – but it is not easy

Non-Linear Least Squares

- We have a nonlinear system of equations
- We want to minimise some error term
  - This is the difference between our measured image point locations, $\left[\hat{x}_i^j, \hat{y}_i^j\right]$, and that predicted by the projection equations
  - This gives us a sum of squares error term:

$$\epsilon = \sum_{i=1}^{m}\sum_{j=1}^{n}\left(\hat{x}_i^j - x_i^j\right)^2 + \left(\hat{y}_i^j - y_i^j\right)^2$$

Non-Linear Least Squares

- We want to find a set of parameters that minimises this error term
- Direct solution is not possible, so we use an iterative solution
  - Start with an initial guess (from the linear solutions we have already seen)
  - Refine this by making a local approximation to the function that is easier to work with

Non-Linear Optimisation

- We'll start with a one-dimensional case
  - We have some (non-negative) error function we want to minimise

$$y = f(x)$$

  - We want to find x to minimise $f(x)$ (ideally = 0)
  - Our solutions are based on the Taylor series

$$f(x_0 + \delta) \approx f(x_0) + \delta f'^{(x_0)} + \frac{\delta^2}{2} f''^{(x_0)} + \cdots + \frac{\delta^n}{n!} f^{(n)}(x_0) + \cdots$$

Newton's Method

- First order approximation

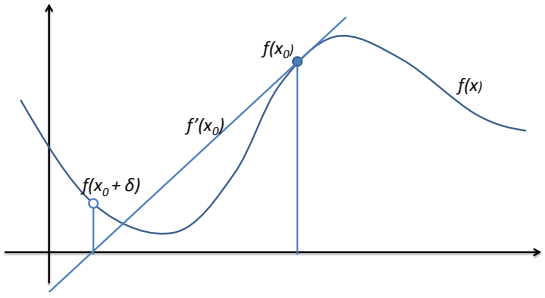
$$f(x) \approx f(x_o) + \delta f'(x_0)$$

- Set $f(x) = 0$ and solve for δ

$$\delta = -f(x_0)/f'(x_0)$$

- Repeat this until we get to 0
  - We might never get to 0: stop when gradient ≈ 0

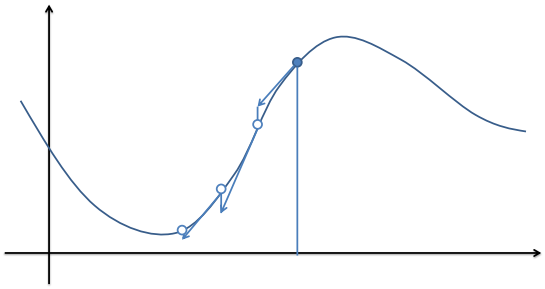
Newton's Method



Gradient Descent

- Gradient descent is a similar approach, but can choose different step sizes along the negative gradient direction
  - Line search – binary division between δ from Newton's method and δ=0 to find minimum value
  - Or just take small steps until you stop going down
- For small enough steps, guaranteed to end up in a (local) minimum

Gradient Descent



Second Order Methods

- At a minimum the gradient of the function is 0

$$f'(x_0) = 0$$

$$\frac{d}{dx}[f(x_0) + \delta f'(x_0)] = 0$$

$$f'(x_0) + \delta f''(x_0) = 0$$

$$f''(x_0)\delta = -f'(x_0)$$

Multivariate Functions

- Our function has many parameters and many output values

  $[y_1, y_2, \dots, y_m]^T = f([x_1, x_2, \dots, x_n]^T)$
- The same arguments apply but derivatives get more complicated
- The first derivative matrix is the *Jacobian*
- The terms in the matrix are partial derivatives – how does y_i change if you change x_j?

The Jacobian

$$J = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_2}{\partial x_n} \\ \vdots & & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \frac{\partial y_m}{\partial x_2} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix}$$

Gauss-Newton Method

- The matrix of second derivatives is called the *Hessian*
- This is difficult to compute, so is often approximated by $J^T J$
- This leads to the Gauss-Newton method where the adjustment is found from

$$J^T J = -J^T \delta$$

- These are known as the *normal equations*

Levenberg-Marquardt

- The Gauss-Newton step usually works well
- Sometimes it does not, so we want to fall back to gradient descent
- This leads to the Levenberg-Marquardt step:

$$(J^T J + \lambda I) = -\delta J$$

- If λ is small this is Gauss Newton
- If λ is big then this is like gradient descent

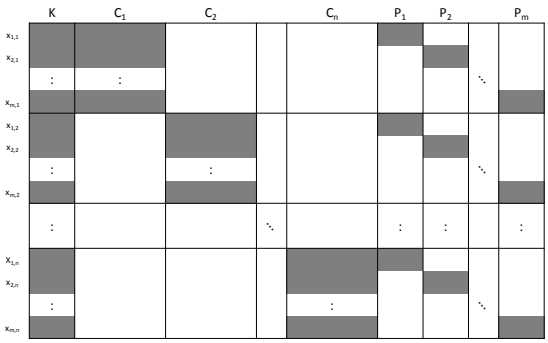
Levenberg-Marquardt

- The Levenberg-Marquardt algorithm gives a method for setting the value of λ over time
- Try the current value of λ to make a step
  - If the result is an improvement, we decrease λ – the function is behaving, so use Gauss-Newton
  - If the result is worse than the current solution, ignore that solution and increase λ – the function is difficult, head towards gradient descent

Sparsity

- We end up with a lot of equations in a lot of unknowns – this makes the Jacobian very big
- Fortunately most of the derivatives are 0
  - Each measurement depends only on one camera's parameters, and one 3D point's parameters
  - All the other entries are zero
- Sparse matrices lead to much more efficient solutions to the normal equations

Sparse Structure



Computing Derivatives

- The remaining task is to compute the derivatives for the Jacobian matrix
- Can derive them analytically
  - Can be difficult for complex cost functions
- Can derive them numerically from

$$f'(x) = \lim_{\delta \to 0} \frac{f(x+\delta) - f(x)}{\delta}$$

  - This depends on the choice of δ – can go wrong

Automatic Differentiation

- Our functions can get complex, but when implemented in code we have only a few basic functions that are combined
- We can use the *chain rule* to reason about the composition of basic functions
- If $y = f(g(x))$ then

$$\frac{dy}{dx} = \frac{df}{dg}\frac{dg}{dx}$$

Automatic Differentiation

- Applying this behind the scenes means that you can compute the derivative automatically
- This can be done using operator and function overloading
- Whenever you apply a function or operator the value and derivative are both computed
- Better than numeric derivatives but can lead to accumulation of error (many operations)