CHAPTER 38

Structure and Motion Offline: SFM Pipelines

The description of SFM in Chapter 36.3 is highly idealized. It should be clear that SFM *could* work, but there are some big steps to take between that description and a useful implementation. Modern structure from modern pipelines are built around Procedure 37.2, but with important refinements. The best modern pipelines can operate very efficiently at enormous scales (entire cities; millions of images). Two very important assumptions in Procedure 37.2 present major practical obstacles. First, the procedure assumed all correspondences were known. But they are not; and a great deal of care is needed to construct accurate correspondences efficiently at a large scale. Second, the procedure assumed that each step of odometry, calibration and triangulation could be executed exactly. But they can't; and the obvious recipe – use these steps to get estimates, then polish the estimates with an optimization procedure – lands you with an optimization problem that is very large and quite nasty.

This chapter will focus on the case where every camera is calibrated. There is quite enough to do in that case. When some or many cameras are uncalibrated, the main thread of the pipeline remains the same, but there are some extra details that need handling. There are a wide range of possible efficiencies avaliable, too. I sketch a variety of harder issues at the end of the chapter.

38.1 A REAL WORLD SFM PIPELINE

This account is a simplified (really!) account of the main structure of an important SFM pipeline, COLMAP. I have omitted considerable detail to make the overall structure of the pipeline clearer. Read the papers [**?**, **?**], and try the software (Section **??**).

38.1.1 Core Difficulties with the Model Recipe

The procedure of Section **??** will not work in the real world. There are a number of serious problems.

- **Correspondence:** The procedure assumes you know correspondences between points, but you don't. You have to find pairs of corresponding points. If you have a very large collection of images, you cannot test every pair of images for corresponding points. Because there are many points, you expect that even quite careful tests of correspondence will produce pairs that do not, in fact, correspond.

- **Robustness:** You should expect that some correspondences are wrong, and may contribute very large error terms. One important mechanism that produces correspondence failures is marks on images, like watermarks, superimposed text, and date-and-time stamps. These could generate correspondences between image pairs that have nothing to do with camera or scene geometry. Other mechanisms can produce large errors, too (below).
- **Correspondence ambiguity:** You might not be able to compute a fundamental matrix for two images with many corresponding points. For example, these two images might be two different images of a plane pattern, and so related by a homography **exercises**.
- **Odometry failures:** In a large pool of images of a big world, some pairs of images may just see parts of the same plane (say, the ground plane). For these pairs, you can't construct the essential matrix, so you can't estimate the relations between the cameras.
- **Triangulation failures:** In a very large set of images, you might have camera pairs which lead to poor or nonsensical triangulations. One source of these problems is a small baseline. Another is a camera that has simply rotated about its focal point – equivalently, a zero baseline.
- **Error propagation:** The procedure assumes you reconstructed the pool correctly, then adds cameras and points. But you didn't reconstruct the pool exactly, meaning the cameras and points you add are likely wrong. The next round might be even more wrong (experience shows it usually is), so errors propagate. This propagation needs to be controlled.

These effects have a strong tendency to interact. For example, correspondence problems can lead to poor camera reconstructions leading to poor triangulations resulting in bundle adjustment converging to a poor solution.

38.1.2 Building Scene Graphs

Procedure 37.2 requires you choose some image which isn't in the working pool and has enough points in common with the points in the working pool. But you cannot tell whether an image has points in common with points in the working pool **exercises**. The best you will be able to do is choose an image which shares points with some or many of the *images* in the working pool.

Definition: 38.1 *Scene graph*

In SFM, a *scene graph* is a graph where each node represents an image. There is an edge between each pair of images where you have strong evidence that enough points in the two images match. The number of points that match should be big enough to yield an acceptable estimate of the transformation between cameras and, if necessary, calibration. *Be careful:* the term "scene graph" has a different meaning in other contexts.

Building a scene graph for a very large pool of images takes care. You can't simply try to count the matches between every pair of images, because there are too many pairs. Instead, you need to find pairs of images that are likely to share many points, then compute the camera transformations for these pairs.

Procedure: 38.1 *Building a scene graph, I*

Find pairs of images that, at a coarse scale, have moderately similar appearance. You could do this by, for example, forming a short code for images that: (a) represents image appearance reasonably accurately and (b) has the property that similar images produce codes that are close and different images produce codes that are far apart. Now use an approximate nearest neighbors procedure to recover the closest k images to each image, as long as they are closer than some threshold. There isn't room here for the details; look up the constructions of [].

This is a form of clustering procedure that yields a set of image pairs where you have a reasonable prospect of obtaining a camera transformation between the images. You can discard any images that do not appear in any pair.

Procedure: 38.2 *Building a scene graph, II*

For all images that appear in a pair, obtain the interest points and their descriptors (SIFT features are canonical, but many others can be used). Create a node for each image. For each image, obtain the others that have moderately similar appearance, yielding a set of possible edges containing that image.
Now for each proposed pair, estimate relations between the cameras. Typically, you use RANSAC or a variant to estimate a fundamental matrix. You also use RANSAC or a variant to estimate a homography. If the homography estimate has a large number of inliers, then either the part of the scene the cameras share is planar or the camera just rotated. In either case, you can't reconstruct using this pair of cameras, so there is no edge between the images in the graph. If the fundamental matrix estimate has few inliers, there is no edge between the pair of images in the graph. Otherwise, construct an edge between the pair in the scene graph. Because I am assuming both cameras are calibrated, you have the fundamental matrix, the essential matrix and so the rotation and unit vector in the direction of translation. You can save some trouble by attaching some information to the edge for later use.

38.1.3 Working With a Scene Graph

At this point, you can mostly follow Procedure 37.2. To do so, you need to:



FIGURE 38.1: **Top** *shows a colmap reconstruction of my study from the 14 photographs on the* **bottom**. *There are two reconstructions. One produces a sparse point cloud, the other a dense point cloud; I show the dense point cloud here. The* **red** *arrow points to one of the reconstructed camera configurations (small red polygons in this view). The* **light blue** *arrows point to features: (a) is the top of a small aquarium; (b) points to a banister visible through the doorway; (c) points to another doorway visible through the doorway. The* **yellow** *arrows on the images point to some large areas not reconstructed, likely because they lack features. The* **orange** *arrows point to some large regions not reconstructed likely because they do not appear in more than one image.*

- choose a pair of images to start the reconstruction process;
- choose which new image you add to the reconstruction;
- triangulate and calibrate effectively;
- control error propagation in the reconstruction.

.

A good choice of starting pair is important. You would like to have a pair of images that: share many points (because then you should have a better estimate of the camera transformation); have a large baseline (so your reconstruction of the points is good); and share many points with many other images (so you can calibrate many other cameras effectively).

Procedure: 38.3 *Obtaining an initial reconstruction*

Choose a pair of images in a dense portion of the scene graph, that is, images that have edges to many other images. Mark the nodes of the scene graph that you use to start. Place the world coordinate system such that the first image is a canonical camera. You know the transformation from the first camera to the second from the scene graph. Choose a scale for the translation between the two cameras **exercises**. Now triangulate the points that match between these two images. You have an initial reconstruction.

It can be useful to have multiple initial reconstructions, grow each separately, then join them later. You could do this by segmenting the scene graph, reconstructing each connected component in its own coordinate system, then using the edges between components to estimate the relations between coordinate systems. More information in the reading list.

Definition: 38.2 *Resectioning*

Choosing the next image and inserting it into the reconstruction is known as *resectioning*.

A good choice of the next image is important. The ideal next image shows many points that have already been reconstructed (so you can estimate the camera well). The reconstructed points should be spread out in the image to get a good camera reconstruction, too.

Procedure: 38.4 *Resectioning, I*

Compute a score for each image that is connected to the marked nodes in the scene graph. One scoring procedure breaks the image into a grid of boxes, then counts the number of boxes that have at least one reconstructed point in them []. This score is larger when there are more common points and the common points are spread out in the image. It is easy to maintain as points are reconstructed **exercises**. Choose the image with the largest score, add that to the reconstruction and mark its node.

Now you must add this new image to the reconstruction.

Procedure: 38.5 *Resectioning, II*

Insert the camera into the working pool using the estimate of the camera transformation and the known calibration

Finally, you must recover positions for any new points. The main obstacle will be correspondence errors. Experience shows that correspondence errors can result in large numbers of outliers. A version of RANSAC, sketched below, can deal with this problem.

Definition: 38.3 *Track*

A *track* is a set of instances of the same point in many different images.

Procedure: 38.6 *Resectioning, III*

Choose some point in the reconstruction you want to triangulate. You have a track corresponding to this point, containing T cameras in the reconstruction. Write $\mathcal{P}_i = [\mathcal{R}_i, -\mathcal{R}_i\mathbf{t}_i]$ for the parameters of the i'th camera *in the track*. Write $\mathbf{x}_i$ for the image coordinates of this point in the i'th image *in the track*. For any pair of images in the track, say the a'th image and the b'th image, you can triangulate to obtain a 3D reconstruction $\mathbf{X}_{ab}$. Write

$$\mathbf{X}_{ab} = t(\mathbf{x}_a, \mathbf{x}_b, \mathcal{P}_a, \mathcal{P}_b)$$

where t is the triangulation procedure you choose. Now use RANSAC to search for a reconstruction of the point that has a large number of inliers. Search by randomly choosing pairs of cameras on the track and triangulating. An inlier occurs when the reprojection error for some other camera is small, and you can trust the triangulation. A triangulation is obtained by intersecting two rays (Figure **??**). If these two rays are nearly parallel, the triangulation is untrustworthy. Test this property by testing the dot-product between the two unit vectors in the direction of the rays. If this is small enough, the triangulation can be trusted. The two vectors are $\mathbf{v}_1 = \mathbf{t}_a - \mathbf{X}_{ab}$ and $\mathbf{v}_1 = \mathbf{t}_a - \mathbf{X}_{ab}$, and the test follows **exercises**. A further test, of *chirality*, checks that a reconstructed point is in front of all the cameras that observe it.

Procedure: 38.7 *Error control*

Each time you add an image to the reconstruction, it is a good idea to perform a partial bundle adjustment. Choose a set of images that is well connected to one another and the new image, and perform bundle adjustment on those images and the associated points in the reconstruction.



FIGURE 38.2: **Top** *shows a colmap reconstruction of my study from the 14 photographs on the* **bottom**. *This is the same reconstruction as shown in Figure 38.1, but now viewed from the top. There are two reconstructions. One produces a sparse point cloud, the other a dense point cloud; I show the dense point cloud here. The* **red** *arrow points to one of the reconstructed camera configurations (small red polygons in this view; easier to see than in the other figure). The* **light blue** *arrows point to features: (a) is a rock in a large aquarium, reconstructed rather well; (b) points to a banister visible through the doorway. The* **yellow** *arrows on the images point to some large areas not reconstructed, likely because they lack features. The* **orange** *arrows point to some large regions not reconstructed likely because they do not appear in more than one image.*

It is a good idea to bundle adjust all cameras and all points currently in the reconstruction once you have grown the reconstruction by a fixed percentage. This is expensive, but helps control error propagation.

38.1.4 Problems Caused by Scene Geometry

Some scene geometries create serious problems for modern SFM pipelines. Multiple instances of the same object in a scene can create significant difficulties. Features on one instance must match features on another instance, and it is hard to tell that these matches are "bad". Figure 38.3 shows an example. A careful choice of the resectioning order and the features used for resectioning can control this problem [**?**].

The structure of the scene graph can cause problems, too. As an extreme example, imagine you have two sets of images. In the first set, you have M_1 images

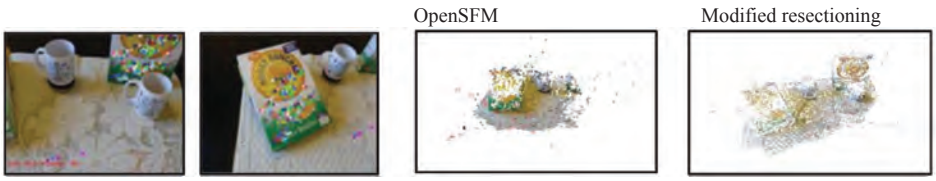


FIGURE 38.3: *There are two instances of the cereal packet in each image (check the cups to confirm this). The packet that is big in one image matches the packet that is almost completely obscured in the other. However, the two large packets are a rich source of local matches that appear rather good (the patterns on the two packets are the same) but are completely misleading. These images are two in a sequence of images. OpenSFM, a strong reference SFM pipeline described below, is misled, and produces a reconstruction that represents the cereal packed rather well, but is not helpful for the rest of the geometry (***center left***). Careful choice of the resectioning order can produce the reconstruction on the* **far left**. Image credit: *This figure shows parts of Figure 5 of "Improving Structure from Motion with Reliable Resectioning", by R. Kataria, J. DeGol and D. Hoiem, 3DV 2020 [?].*

of N_1 points, and in the second, you have M_2 images of N_2 completely different points. Start with two images in the first set. You will not be able to add any image in the second set to the working pool of images, because you can't use points in the first set to calibrate cameras in the second set. You will be able to reconstruct each set of points separately, *but you can't register them to one another.* It follows that a scene graph that has more than one connected component will cause problems. You should be able to reconstruct each connected component separately, but you will not be able to compute the transformation from one reconstruction to the other.

Scene graphs that have multiple connecte components are uncommon. The more common case occurs is a scene graph which you can separate into several component by cutting a very small number of edges. You expect that you should be able to reconstruct the geometry and cameras for each component. But you might have real trouble getting a good estimate of the transformations from component to component. This problem is important in practice. For example, this situation might occur when a robot travels down a corridor then turns right to travel down another corridor. Images taken pointing down each corridor will have lots of shared points. But at the turn, the robot might produce pictures with very few shared points. Worse, there will be repetitions in the corridors, meaning that there are many false matches. The result will be a scene graph which is easily misinterpreted (Figure 38.4). This problem occurs in practice (Figure 38.5).

For very large geometries, error propagation can be a serious problem. For example, think about reconstructing the facade of a very long building. You reconstruct the far left end of the building, then get some more points on the left side from more pictures, yet more points in the center from yet more pictures, and so on. The reconstruction at one end of the building is registered to the reconstruction at the other end by a long chain of estimates of camera transformations. Each estimate likely has an error, and these errors might (and do!) accumulate. Figures **??**

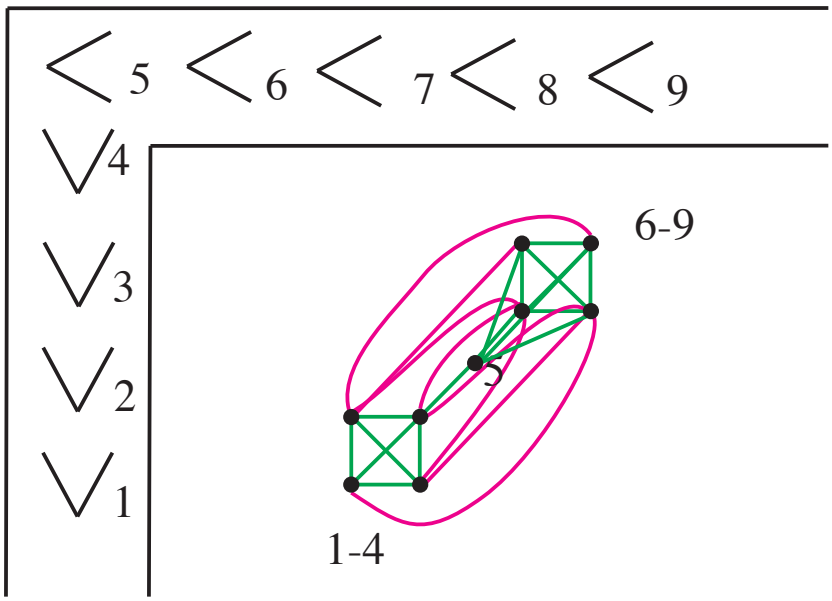


FIGURE 38.4: *A robot travels down a corridor, turns right, and continues, taking pictures all the while. The V's indicate camera position and direction. The scene graph, shown in the* **inset**, *may be very bad. Edges in* **green** *indicate reasonable matches, obtained because there are features (doors, whiteboards, and so on) along the corridor. For example, there are many matches between 1 and 2. At viewpoint 5, the image sees very few features visible in the vertical corridor; these link 4 to 5. However, many features are shared with 6-10. Because the doors and features down the corridor largely look like one another, there will be many feature matches from frames in the vertical corridor to those in the horizontal corridor* that are wrong *(and are indicated in* **red***). As a result, it is relatively straightforward to reconstruct these images as a single corridor.*

and 38.7 show examples of this effect.

This error can be controlled by noticing that the facade of buildings tends to be flat on a long scale. The difficulty is determining just what underlying plane faces of the building are. One way to do this is to fit planes to local reconstructions, then use these to estimate a larger plane. If many local planar patches are pieces of quite similar planes, you can estimate one plane that explains all, then force the reconstruction to respect that plane (details in [**?**]).

Alternatively, you can estimate the planes by identifying parts of the reconstruction that should have mirror symmetries. These will tend to (a) produce matching errors and (b) match in the reconstruction. Enough symmetric pairs in turn suggest a building aligned coordinate system, which consists of directions along the facade; across the facade; and up/down. Finally, you adjust the reconstruction

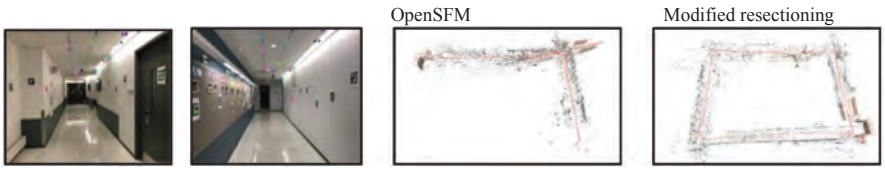


FIGURE 38.5: *The building plan has corners where there are few matches. Further, the corridors don't have distinctive features (so the corridors marked with the* **blue** *arrows look mostly like those marked with the* **red** *arrows). The OpenSFM matches "fold" the plan because of this symmetry. Careful choice of the resectioning order can produce the reconstruction on the* **far left**. Image credit: *This figure shows parts of Figure 5 of "Improving Structure from Motion with Reliable Resectioning", by R. Kataria, J. DeGol and D. Hoiem, 3DV 2020 [?].*

to respect this coordinate system (details in [**?**]).

38.1.5 Improving Efficiency

Some images may not actually be very helpful. For example, if the baseline between two images is small compared to the distance to all the points, triangulating using those images will work badly. Pairs of images with a small baseline are common when you try to reconstruct famous monuments from large numbers of internet images. Typically, tourists like to get iconic views of famous monuments, so their viewpoints tend to cluster quite strongly. Two pictures from the a cluster will tend to have a small baseline. Having many pictures from a cluster will increase the cost of computation without significantly improving estimates of point locations (you will likely reject a lot of triangulations as unreliable). You may find reconstructions are actually worse, because these images can make robustness problems harder to manage. The cure for this problem is to check the camera estimate when you insert an image into the working pool. If it is close to an existing camera, it might be a good idea to reject the image.

It is quite usual for scene graphs to cluster. There will be groups of vertices that have very many edges internally, but relatively few edges from group to group. Experience shows that it is a good idea to cluster the scene graph (for example, using methods like those of Section **??**). Reconstruct inside each cluster first, then use the cross-cluster edges to improve the estimate.

38.1.6 Application: Visualizing Cities using SFM

38.1.7 Application: Construction Monitoring using SFM

38.2 BUNDLE ADJUSTMENT

Bundle adjustment requires care. If you start with millions of images, you could expect hundreds of millions of points and millions of cameras. The resulting optimization problem is very large, and needs careful handling. Even at small scales, the optimization problem is quite hard. There has been considerable work on constructing well-behaved variants (Section 36.3). It is usual to phrase bundle ad-

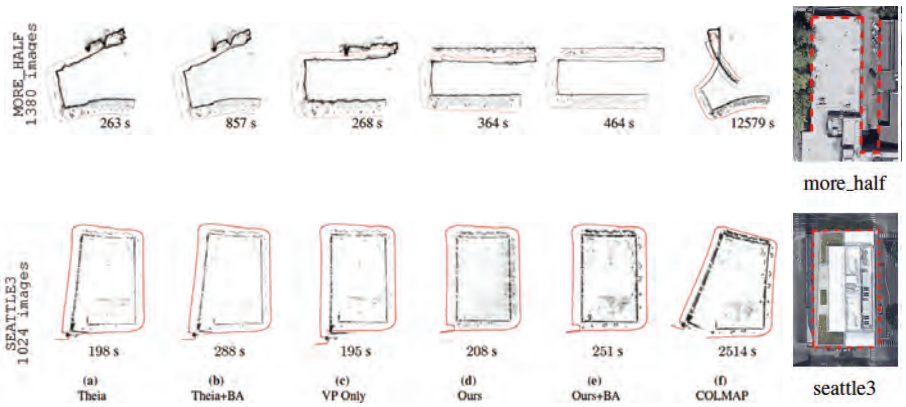


FIGURE 38.6: *Error accumulation in reconstructions can have serious effects. Each row shows sets of reconstructions of two datasets, shown from above, varying by the method used, together with a satellite image of building shown in the dataset (***far right***). The satellite image is marked with an estimate of the building facade in red. For the top row, you have to rotate the picture in your head to match to the reconstructions. The key point here is that pure incremental structure from motion pipelines tend to produce significant cumulative errors. For example, if you used the reconstructions on the bottom row, you would not know the building is rectangular. Cumulative error can be controlled by identifying shared global properties in the images; details in text.* Image credit: *This figure shows parts of Figure 5 and parts of Figure 8 of "Reducing Drift in Structure From Motion Using Extended Features," by A. Holynski, D. Geraghty, J-M. Frahm, C. Sweeney and R. Szeliski, 3DV, 2020 [**?**]*

justment as a non-linear least-squares problem, then use Newton's method with the Gauss-Newton approximation. With appropriately chosen images, an efficient updating procedure computes a camera update using fixed points, then a point update using the refined cameras.

Many readers will not need all the detail below. There are some key points to be aware of:

- Very often, it is unwise to use all images. Careful winnowing of images can lead to smaller, better behaved bundle adjustment.
- Experience shows that bundle adjustment can quite easily converge to bad solutions from quite good start points.
- In the ideal case, each point appears in a small number of images where the baseline between the cameras is relatively large, but each camera sees many points.
-

These points follow from fairly detailed properties of the optimization, sketched below.

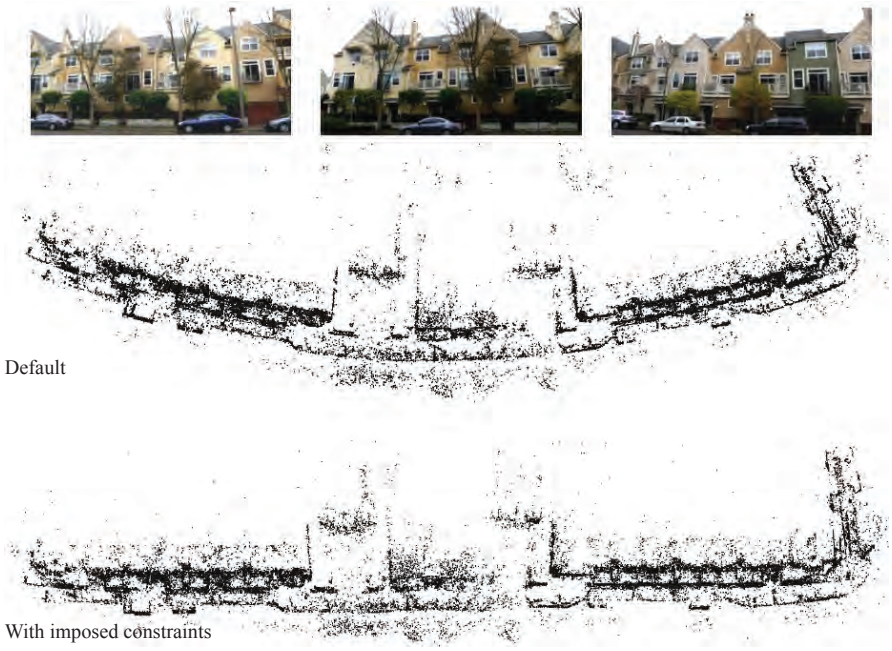


FIGURE 38.7: *Error accumulation in reconstructions can be controlled by reasoning about the global structure of buildings. The* **top** *row shows some frames of the long, straight facade of a building. The* **center** *row shows an overhead view of the reconstructed points. Note the curve, caused by accumulated error. This error can be controlled by noting the facade of a building tends to be flat on a long scale. The* **bottom** *row shows a reconstruction that employs this constraint. Note significant improvements in the geometry.* Image credit: *This figure shows parts of Figure 1 of "Discovering and exploiting 3D symmetries in structure from motion", by A. Cohen, C. Zach, S. N. Sinha and M. Pollefeys, CVPR 2012 [?].*

38.2.1 Newton's Method

You wish to minimize $R(\theta)$. Assume you have an estimate $\theta^{(n)}$ of the variables. Then Newton's method computes an update δ by minimizing a Taylor series around that estimate as a function of δ, then computes $\theta^{(n+1)} = \theta^{(n)} + \delta$.

You must minimize

$$R(\theta^{(n)}) + \mathbf{g}^T \mathcal{J}_{\mathbf{g},\theta}\delta + (1/2)\delta^T \mathcal{H}_R \delta$$

which yields

$$\mathcal{H}_R\delta = -\mathcal{J}^T_{\mathbf{g},\theta}\mathbf{g}$$

and $\theta^{(n+1)} = \theta^{(n)} + \delta$. Experience shows that this iteration very often does not converge. It is guaranteed to converge if $\mathcal{H}_R$ is positive definite **exercises**. There is usually no particular reason to believe that $\mathcal{H}_R$ will be positive definite. However,

you could minimize

$$R(\theta^{(n)}) + \mathbf{g}^T \mathcal{J}_{\mathbf{g},\theta}\delta + (1/2)\delta^T(\mathcal{H}_R + \lambda\mathcal{I})\delta$$

for some λ large enough to make $(\mathcal{H}_R + \lambda\mathcal{I})$ positive definite. This yields

$$(\mathcal{H}_R + \lambda\mathcal{I})\delta = -\mathcal{J}^T_{\mathbf{g},\theta}\mathbf{g}$$

and you update θ as before. This approach is known as the *Levenberg Marquardt* method. It is a quasi Newton method. The tricky bit here is choosing λ; too small, and the method may diverge; too big, and the method will be gradient descent. Generally, methods use an adaptive procedure. If a step has been successful – which you can test by looking at the objective – reduce λ and try the next step. If it has not, increase λ and retry. More efficient procedures for managing a Hessian that is not positive definite are available [**?**].

38.2.2 The Gauss-Newton Approximation

Bundle adjustment as an optimization problem has a number of special features. Recall the notation of Section 37.2; you must minimize

$$R(\mathcal{P}_1, \ldots, \mathcal{P}_M, \mathbf{X}_1, \ldots, \mathbf{X}_N) = \sum_{i,j} \delta_{i,j} d(\mathbf{x}_{i,j} - L(\mathcal{P}_j, \mathbf{X}_i)).$$

Assume that $d(\mathbf{r}) = \mathbf{r}^T\mathbf{r}$ for the moment. Then bundle adjustment is a *nonlinear least-squares problem*. These problems have the form

$$R(\theta) = (1/2)\mathbf{g}(\theta)^T\mathbf{g}(\theta)$$

where θ are the parameters you need to solve for and $\mathbf{g}$ is a vector of functions **exercises**. Write $\mathcal{J}_{\mathbf{g},\theta}$ for the matrix of derivatives (or Jacobian) whose i, j'th element is

$$\frac{\partial g_i}{\partial \theta_j}.$$

Now

$$\nabla R = \mathbf{g}^T \mathcal{J}_{\mathbf{g},\theta}$$

exercises. In turn, the Hessian (matrix of second partial derivatives) is given by

$$\mathcal{H}_R = \mathcal{J}^T_{\mathbf{g},\theta}\mathcal{J}_{\mathbf{g},\theta} + \text{term in second derivatives}$$

exercises and the Gauss-Newton approximation ignores the last term and uses the estimate

$$\hat{\mathcal{H}}_R = \mathcal{J}^T_{\mathbf{g},\theta}\mathcal{J}_{\mathbf{g},\theta}.$$

38.2.3 Sparsity and Efficiency

You would not actually evaluate $R(\mathcal{P}_1, \ldots, \mathcal{P}_M, \mathbf{X}_1, \ldots, \mathbf{X}_N)$ by summing as in the expression

$$\sum_{i,j} \delta_{i,j} d(\mathbf{x}_{i,j} - L(\mathcal{P}_j, \mathbf{X}_i))$$

because most $\delta_{i,j}$ are zero. Instead, you would keep a list of the non-zero $\delta_{i,j}$ terms, and evaluate and sum the $d(\mathbf{x}_{i,j} - L(\mathcal{P}_j, \mathbf{X}_i))$ terms corresponding to them.

Similar efficiencies are available for Newton's method, because the Hessian of R is very sparse. Write h_{jk} for the j, k'th element of $\hat{\mathcal{H}}_R$. Then

$$h_{kl} = \sum_{i,j} \frac{\partial d(\mathbf{x}_{i,j} - L(\mathcal{P}_j, \mathbf{X}_i))}{\partial \theta_k} \frac{\partial}{\partial \theta_l}$$

exercises. This general point has important consequences for bundle adjustment. You can use it to show that many of the entries in the Hessian are zero. The cases are:

- If j indexes one component of a point and k indexes another component of the same point, h_{jk} is not, in general, zero **exercises**.
- If j indexes one component of a point and k indexes a component of a different point, h_{jk} is zero **exercises**.
- if j indexes a camera variable and k indexes a point that appears in that camera's image, h_{jk} is not, in general, zero **exercises**.
- if j indexes a camera variable and k indexes a point that does not appear in that camera's image, h_{jk} is zero **exercises**.
- if j indexes a camera variable and k indexes a camera variable from the same camera, h_{jk} is not, in general, zero **exercises**.
- if j indexes a camera variable and k indexes a camera variable from a different camera, h_{jk} is zero **exercises**.

Generally, you want to have a Hessian that is very sparse, because that makes computing the update easier. You might think that you want to have many pictures of the same point so as to estimate the point's coordinates better. But if you do, the Hessian will be less sparse, and the Gauss-Newton method might be unpleasantly slow. The ideal is to have each point appear in a small number of images where the baseline between the cameras is relatively large.

38.2.4 Schur Factorization

Now organize the variables so that $\theta = [\mathcal{P}_1, \ldots, \mathcal{P}_M, \mathbf{X}_1, \ldots, \mathbf{X}_N]^T = [\mathbf{c}, \mathbf{p}]^T$ (so separating camera variables and point variables). In this ordering, you can write

$$\mathcal{J}_{\mathbf{g},\theta} = [\mathcal{J}_{\mathbf{g},\mathbf{c}}, \mathcal{J}_{\mathbf{g},\mathbf{p}}]$$

and so

$$\hat{\mathcal{H}}_R = \begin{bmatrix} \mathcal{J}_{\mathbf{g},\mathbf{c}}^T \mathcal{J}_{\mathbf{g},\mathbf{c}} & \mathcal{J}_{\mathbf{g},\mathbf{c}}^T \mathcal{J}_{\mathbf{g},\mathbf{p}} \\ \mathcal{J}_{\mathbf{g},\mathbf{p}}^T \mathcal{J}_{\mathbf{g},\mathbf{c}} & \mathcal{J}_{\mathbf{g},\mathbf{p}}^T \mathcal{J}_{\mathbf{g},\mathbf{p}} \end{bmatrix} = \begin{bmatrix} \mathcal{C} & \mathcal{X} \\ \mathcal{X}^T & \mathcal{P} \end{bmatrix}.$$

Here $\mathcal{C}$ contains only camera terms and has a block diagonal structure **exercises**; $\mathcal{P}$ has only point terms and also has a block diagonal structure **exercises**; and $\mathcal{X}$ contains cross-terms (one index refers to a point variable and the other to a

camera variable) and might be quite dense depending on what points appear in what cameras **exercises**.

You can also break up the update, so $\delta = [\delta_c, \delta_p]$. Further, break up the term $-\mathcal{J}^T_{\mathbf{g},\theta}\mathbf{g}$ (Newton's method, Section 36.3) to get

$$-\mathcal{J}^T_{\mathbf{g},\theta}\mathbf{g} = \begin{bmatrix} \mathbf{u} \\ \mathbf{v} \end{bmatrix}.$$

With some reorganization, etc. you can show that

$$\left[\mathcal{C} - \mathcal{X}\mathcal{P}^{-1}\mathcal{X}^T\right]\delta_c = \left[\mathbf{u} - \mathcal{X}\mathcal{P}^{-1}\mathbf{v}\right] \text{ and } \mathcal{P}\delta_p = \left[\mathbf{v} - \mathcal{X}^T\delta_c\right].$$

The procedure is known as *Schur factorization*. It is often advantageous in bundle adjustment. You expect many more points than cameras, so $\mathcal{C}$ is a lot smaller than $\mathcal{P}$. In the ideal case, each camera has many points *but* each point appears in relatively few cameras. In this case, $\mathcal{X}$ will have a lot of zeros. Further, many points contribute to a camera estimate. In this case, Schur factorization boils down to using all the points to update the cameras; then using the camera update to estimate the point update. Schur factorization can be efficient because $\mathcal{P}^{-1}$ is relatively easy to form. $\mathcal{P}$ consists of N blocks, each 3×3, on the diagonal. You invert each block separately, and you have $\mathcal{P}^{-1}$. Notice that you can update δ_p point by point because $\mathcal{P}^{-1}$ is block diagonal.

38.2.5 GloMap

GloMap is an SFM pipeline that uses a variant of bundle adjustment. Adjusting camera parameters and point positions simultaneously becomes demanding when there are many points. GloMap estimates rotations from the camera coordinate system to the world coordinate system, then uses these rotations to estimate camera center locations and point positions. This section sketches the main strategies, but I have omitted considerable detail. Read the paper [**?**]**exercises**.

Write $\mathcal{R}_i$ for the rotation that takes points in camera i's coordinate system to points in the world coordinate system. You do not know these rotations. Write $\hat{\mathcal{R}}_{i\to j}$ for the rotation that takes points in camera i's coordinate system to points in camera j's coordinate system. The hat is there to remind you that you have good estimates of these rotations from the odometry procedure. In the procedure above, you obtain a world coordinate system by selecting one camera and making that camera a canonical camera. Call this camera 1, so $\mathcal{R}_1$ is the identity.

Now you could compute $\mathcal{R}_i$ it by following a path in the scene graph connecting image i to image 1. If there is no such path, you're in trouble because the scene graph has more than one connected component (Section 38.1.4). Write $(i, j, k, \ldots, s, 1)$ for the cameras along this path. Then you could estimate $\mathcal{R}_i$ as $\hat{\mathcal{R}}_{i\to j}\hat{\mathcal{R}}_{j\to k}\ldots\hat{\mathcal{R}}_{s\to 1}$. You expect that there are many paths. Nothing in the construction of the scene graph so far guarantees that different paths will produce the same rotation estimate.

Coming up with a single, consistent rotation estimate from a collection of slightly different estimates is known as *rotation averaging*. Rotation averaging is tricky, because rotation matrices are constrained to be orthonormal. If you simply average a set of rotation matrices, the result will not usually be a rotation matrix.

Write $\mathcal{E}$ for the edges of the scene graph, which correspond to pairs of cameras where you have an estimate of rotation and translation between the two. Write $d(\mathcal{R}, \mathcal{S})$ for a distance function on pairs of matrices $\mathcal{R}$ and $\mathcal{S}$. GloMap solves for the rotations that minimize

$$\sum_{(i,j)\in\mathcal{E}} d(\mathcal{R}_i^T \hat{\mathcal{R}}_{ij} \mathcal{R}_j, \mathcal{I}).$$

This is a constrained optimization problem (the matrices you obtain must be rotation matrices). Because $\mathcal{R}_i$ represents the rotation that takes points in camera i's coordinate system to points in the world coordinate system, $\mathcal{R}_j^T \mathcal{R}_i$ will take points in i's coordinate system to points in j's coordinate system. Check that if you replace each $\hat{\mathcal{R}}_{ij}$ with $\mathcal{R}_j^T \mathcal{R}_i$, using the rotations from the rotation averaging solution, you will have a graph where every path from i to j yields the same rotation **exercises**. Using the Frobenius norm for d may lead to robustness problems **exercises**.

The rotation averaging strategy above does not modify point location estimates or camera translation estimates. GloMap now uses the rotation estimates to improve camera translation estimates and point estimates. Write $\mathbf{v}_{ik}$ for the ray from the current estimate of the center of camera i to the current estimate of the location of point k, *rotated into the world coordinate system*. You can compute this vector because you know the rotation from camera i's coordinate system to the world coordinate system. Write $\mathbf{o}_i$ for the (currently unknown) location of the i'th camera center, $\mathbf{X}_k$ for the (currently unknown) location of the k'th point in the world coordinate system, and d_{ik} for an unknown positive value that represents depth to the point. Write $\mathcal{D}$ for the set of all pairs (u, v), where u is the index of a camera and v is the index of a point that can be seen in that camera. Finally, write ρ for a robust cost function (GloMap uses the Huber loss). Then you can estimate the camera centers and the point locations by minimizing

$$\sum_{(i,k)\in\mathcal{D}} \rho(\| \mathbf{v}_{ik} - d_{ik}(\mathbf{X}_k - \mathbf{o}_i) \|_2)$$

as a function of the point locations, camera centers and depths. You should use the constraint $d_{ik} > 0$.

38.2.6 FastMap

The bundle adjustment problem I described involved optimizing all camera parameters and all point positions simultaneously, using a start point derived from the scene graph. This presents really serious scale problems. FastMap is a recent procedure that formulates bundle adjustment in terms only of camera parameter estimates. This leads to a speedup, because the number of points could be extremely large. FastMap uses a first order procedure rather than Newton's method, which offers a further speedup. This section sketches the main strategies, but I have omitted considerable detail. Read the paper [**?**]**exercises**.

Here is one strategy. The i'th node of the scene graph is connected to a set of other nodes $\mathcal{N}(i)$ (the neighbors – there are $|\mathcal{N}(i)|$ of these).

Write $d(\mathcal{R}, \mathcal{S})$ for a distance function on pairs of rotation matrices $\mathcal{R}$ and $\mathcal{S}$. It is quite usual to use

$$d(\mathcal{R}, \mathcal{S}) = \cos^{-1}\left[\frac{\text{Trace}(\mathcal{R}^T\mathcal{S}) - 1}{2}\right].$$

exercises

Now find the rotations $\mathcal{R}$ that minimize

$$\sum_{i \in \text{vertices}} \left[\frac{1}{|\mathcal{N}(\rangle)|} \sum_{j \in \mathcal{N}(i)} d(\mathcal{R}_i, \mathcal{R}_{i\to j}\mathcal{R}_j)\right].$$

This is a large optimization problem, because you must minimize over all rotations together. It has a reputation for producing local minima. One strategy for dealing with these difficulties is given in the FastMap paper [**?**]. A pointer to a codebase appears in Section 38.3.

The rotation averaging strategy above does not modify point location estimates or camera translation estimates. FastMap uses the rotation estimates to obtain camera translation estimates independent of point estimates.

Write $\mathbf{o}_i$ for the (currently unknown) location of the i'th camera center, and $\mathbf{o}_{i\to j}$ for the unit vector pointing from camera center i to camera center j. This is known, because odometry gives you a unit vector in the coordinate system of a camera, and you know the rotation from that camera coordinate system to the world coordinate system. FastMap then solves

$$\sum_{(i,j)\in\mathcal{E}} \| \frac{\mathbf{o}_i - \mathbf{o}_j}{\|\mathbf{o}_i - \mathbf{o}_j\|_2} - \mathbf{o}_{i\to j} \|_1.$$

Experience shows this problem to be well behaved. FastMap applies further polishing steps to the camera parameter estimates. In some applications (Chapter 36.3), you may not actually need to reconstruct the points if you know the camera parameters. If you do need point positions, recovering points from very large numbers of accurate cameras is straightforward (Section 36.3).

38.3 RESOURCES

There is a substantial volume of open source software related to structure from motion. There are a number of pipelines, and a variety of toolkits.

Resource pointers: 38.1 *Compilations*

Patricio Gonzalez Vivo offers a list of SFM tools at `https://gist.github.com/patriciogonzalezvivo/0cc2d0fb6e9af9040eff`. Wikipedia has a list of toolboxes at `https://en.wikipedia.org/wiki/Bundle_adjustment`. Structure from motion falls into the general aegis of *photogrammetry* (obtaining measurements using photographs), and Wikipedia has a detailed comparison of photgrammetry software at `https://en.wikipedia.org/wiki/Comparison_of_photogrammetry_software`.

There is a large number of open source packages available on the internet. The list below gives a reasonable sample. I just don't know enough to recommend any one package, but COLMAP has a reputation as a reference package. Many people start with COLMAP.

Resource pointers: 38.2 *COLMAP and GLOMAP*

COLMAP is often used as a reference package. You can find it at `https://colmap.github.io`. It was originally described in [**?**, **?**]. COLMAP regularly incorporates revisions. For example, GLOMAP – which was built around a novel procedure for global bundle adjustment, sketched above and originally described in [**?**] – is available at `https://github.com/colmap/glomap`, but is deprecated because it has been incorporated into COLMAP. Authors, in alphabetical order, are: Daniel Barath and Jan-Michael Frahm and Linfei Pan and Marc Pollefeys and Johannes Lutz Schönberger and Enliang Zheng.

Resource pointers: 38.3 *Fastmap*

Fastmap, at `https://github.com/pals-ttic/fastmap`, uses first order optimization procedures for bundle adjustment. These can be run on a GPU, and offer significant speed and accuracy improvements. Authors are: Jiahao Li and Haochen Wang and Muhammad Zubair Irshad and Igor Vasiljevic and Matthew R. Walter and Vitor Campagnolo Guizilini and Greg Shakhnarovich; you can find more details in [**?**]

Resource pointers: 38.4 *BerkeleySFM*

Berkeley SFM is described by its authors, Erik Nelson and David Fridovich-Keil, as " a structure from motion library created to explore new solutions to challenging 3D reconstruction problems (large scale, indoors, incremental, homogeneously textured, etc.)." You can find it at `https://erik-nelson.github.io/berkeley_sfm/`.

Resource pointers: 38.5 *Bundler*

Bundler is an SFM package specifically aimed at scattered images such as collections of tourist pictures. Authors are: N. Snavely, S.M. Seitz and R. Szeliski; you can find more details in [**?**, **?**]. Find Bundler at `https://www.cs.cornell.edu/~snavely/bundler/`. There are a number of links to associated packages there.

Resource pointers: 38.6 *GTSFM*

GTSFM or Georgia Tech Structure-from-Motion is a structure from motion pipeline based around a specific optimization system aimed at solving smoothing and optimization problems on manifolds. It supports parallel computation. You can find it at `https://github.com/borglab/gtsfm`. Authors are: Ayush Baid, John Lambert, Travis Driver, Akshay Krishnan, Hayk Stepanyan and Frank Dellaert; more details in [**?**].

Resource pointers: 38.7 *OpenSFM*

OpenSfM is an open source SFM pipeline created and maintained by Mapillary. Mapillary is a platform for automating mapping, see `https://www.mapillary.com`. I cannot identify the authors of OpenSFM, but Mapillary was founded by: Jan Erik Solem, Johan Gyllenspetz, Peter Neubauer and Yubin Kuang. You can find OpenSFM at `https://opensfm.org`.

Resource pointers: 38.8 *OpenMVG*

OpenMVG is a software resource, consisting of libraries (for tasks like feature matching), binaries (for tasks like reconstruction) and pipelines. Pipelines are created by chaining binaries to, for example, triangulate or reconstruct. There is too long a list of authors to put here; you can find the authors at `https://raw.githubusercontent.com/openMVG/openMVG/master/AUTHORS`. You can find OpenMVG at `https://github.com/openmvg/openmvg`.

Resource pointers: 38.9 *VisualSFM*

VisualSFM is a GUI oriented pipeline for SFM reconstruction (though a command line interfaceis available). The author is Changchang Wu. You can find VisualSFM at `http://ccwu.me/vsfm/`

Resource pointers: 38.10 *Theia*

Theia is a library for SFM, authored by Chris Sweeney. The library contains many algorithms for pose, feature detection and description, matching and reconstruction. Find Theia at `http://www.theia-sfm.org`

Resource: Bundle adjustment There are a number of bundle adjustment libraries, including:

- SBA, for sparse bundle adjustment, at `https://github.com/mlourakis/sba`. The authors are: M. I. A. Lourakis and A. A. Argyros
- SSBA, for sparse bundle adjustment, at `https://github.com/chzach/SSBA/tree/master/SSBA-4.0`. The author is Christopher Zach.
- MCBA is a multicore bundle adjustment package, at `https://grail.cs.washington.edu/projects/mcba/`. The author is Changchang Wu.

You can also use large-scale general optimization packages for bundle adjustment, including:

- Libdogleg, at `https://github.com/dkogan/libdogleg`. The author is Dima Kamen.
- Ceres, at `http://ceres-solver.org`. The authors are Sameer Agarwal and Keir Mierle and The Ceres Solver Team.

38.4 YOU SHOULD

38.4.1 remember these facts:

38.4.2 remember these procedures:

Building a scene graph, I 693
Building a scene graph, II 693
Obtaining an initial reconstruction 695
Resectioning, I 695
Resectioning, II 696
Resectioning, III 696
Error control 697

38.4.3 exploit these resources:

Compilations 708
COLMAP and GLOMAP 708
Fastmap 708
BerkeleySFM 708
Bundler 709
GTSFM 709
OpenSFM 709
OpenMVG 709
VisualSFM 710
Theia 710
General purpose optimizers for bundle adjustment 710

38.4.4 be able to:

EXERCISES

QUICK CHECKS

38.1. Section 38.2.2 claims that bundle adjustment is a non-linear least squares problem. Verify this claim.

38.2. Section 38.2.2 claims that, if $R(\theta) = (1/2)\mathbf{g}(\theta)^T\mathbf{g}(\theta)$ is a nonlinear least squares problem, and $\mathcal{J}_{\mathbf{g},\theta}$ is the matrix of derivatives (or Jacobian) of $\mathbf{g}$, then $\nabla R = \mathbf{g}^T\mathcal{J}_{\mathbf{g},\theta}$. Verify this claim.

38.3. Section 38.2.2 claims that, if $R(\theta) = (1/2)\mathbf{g}(\theta)^T\mathbf{g}(\theta)$ is a nonlinear least squares problem, and $\mathcal{J}_{\mathbf{g},\theta}$ is the matrix of derivatives (or Jacobian) of $\mathbf{g}$, then $\mathcal{H}_R = \mathcal{J}_{\mathbf{g},\theta}^T\mathcal{J}_{\mathbf{g},\theta} +$ term in second derivatives. Verify this claim.

38.4. Section 38.2.3 claims that if h_{jk} is the j, k'th element of $\hat{\mathcal{H}}_R$, $h_{jk} = \sum_i \frac{\partial g_i}{\partial \theta_j}\frac{\partial g_i}{\partial \theta_k}$. Verify this claim (remember, $\hat{\mathcal{H}}_R$ had the second derivative terms dropped).

38.5. Section 38.2.3 claims that if j indexes one component of a point and k indexes another component of the same point, h_{jk} is not, in general, zero. Verify this claim.

38.6. Section 38.2.3 claims that if j indexes one component of a point and k indexes a component of a different point, h_{jk} is zero. Verify this claim.

38.7. Section 38.2.3 claims that if j indexes a camera variable and k indexes a point that appears in that camera's image, h_{jk} is not, in general, zero. Verify this claim.

38.8. Section 38.2.3 claims that if j indexes a camera variable and k indexes a point that does not appear in that camera's image, h_{jk} is zero. Verify this claim.

38.9. Section 38.2.3 claims that if j indexes a camera variable and k indexes a camera variable from the same camera, h_{jk} is not, in general, zero. Verify this claim.

38.10. Section 38.2.3 claims that if j indexes a camera variable and k indexes a camera variable from a different camera, h_{jk} is zero. Verify this claim.

38.11. Section 38.2.4 claims that $\mathcal{C}$ contains only camera terms and has a block diagonal structure. Verify this claim.

38.12. Section 38.2.4 claims that $\mathcal{P}$ has only point terms and also has a block diagonal structure. Verify both claims.

38.13. Section 38.2.4 claims that $\mathcal{X}$ contains cross-terms (one index refers to a point variable and the other to a camera variable) and might be quite dense depending on what points appear in what cameras. Verify this claim. Under what circumstances is $\mathcal{X}$ very dense? under what circumstances might it be quite sparse?

38.14. Section 38.1.2 says: "For real world images, you do not know which points in which image correspond. This means you can't use Procedure 37.2 as described. You can't choose some image which isn't in the working pool and has enough points in common with the points in the working pool. This is because you cannot tell whether an image has points in common with points in the working pool." Explain.

38.15. Section 38.1.3 says: "Choose a scale for the translation between the two cameras." Why do you need to do this? Why do you not need to choose a scale for the translation to the third camera?

38.16. Section 38.1.3 says: "This score is larger when there are more points and they are spread out in the image, and is easy to maintain as points are reconstructed." Explain.

38.17. Section 38.1.3 says you can test whether a triangulation is trustworthy by checking two vectors, $\mathbf{v}_1 = \mathbf{t}_a - \mathbf{X}_{ab}$ and $\mathbf{v}_1 = \mathbf{t}_a - \mathbf{X}_{ab}$. What would you

do with these vectors? Why is the triangulation untrustworthy when they are nearly parallel?

LONGER PROBLEMS

38.18. You wish to minimize $f(\mathbf{x})$, which you can evaluate at every $\mathbf{x}$, and has the property that $\mathcal{H}_f$ is defined and is positive definite at every $\mathbf{x}$. Do so by adopting Newton's method. Update an estimate of the minimum $\mathbf{x}^{(n)}$ by choosing δ to minimize $f(\mathbf{x}^{(n)} + \delta^T \nabla f + (1/2)\delta^T \mathcal{H}_f \delta$, then forming $\mathbf{x}^{(n+1)} = \mathbf{x}^{(n)} + \delta$.

(a) Show that $f(\mathbf{x}^{(n+1)}) \leq f(\mathbf{x}^{(n)}$.

(b) Under what circumstances are $f(\mathbf{x}^{(n+1)})$ and $f(\mathbf{x}^{(n)}$ equal?

(c) Is f necessarily convex? (not for everyone)

(d) You now have a function g where you don't know whether $\mathcal{H}_g$ is positive definite or not. Show that there always exists some λ such that $\mathcal{H}_g + \lambda\mathcal{I}$ is positive definite.

PROGRAMMING EXERCISES

38.19. (a)