Exercise session using MATLAB: Quasiconvex Optimixation

Overview

In this laboratory session you are going to use matlab to study structure and motion estimation using the L_∞-norm as well as how to create panoramic images by stitching. For further reading, see the PAMI paper by Kahl/Hartley.

First, download and unpack `OCV2008.zip` from the course homepage.

`http://www2.imm.dtu.dk/projects/OCVschool/Materials.html`

To start the correct version of MatLab on DTU's unix terminals you should type `matlab72` in the terminal window. Remember to run `startup.m` to get the correct paths. The files you are going to use can be found in the subdirectory `QuasiConvexity`. Some of the instructions here are matlab commands that you are supposed to try. To relieve you of some of the tedious typing, we have collected some of those commands in a script `qc_cheats.m`.

Structure and Motion under the L_∞-Norm

Brief Introduction

This section is just to recapitulate some basic facts about optimization with the L_∞-norm for geometric problems. If this is already familiar to you, then proceed directly to the next section.

The traditional way of minimizing the sum of squares error can sometimes fail. The problem is that many reconstruction problems are non-convex and the optimization may get stuck in local non-optimal solutions. The globally optimal solution can be hard to compute. However, instead of minimizing the sum of squares of reprojection errors, it turns out to be a simpler problem to minimize the *maximum* error for certain structure and motion problems. This corresponds to changing the two-norm (denoted L_2) to the max-norm (denoted L_∞) of the residual vector in our optimization problem.

More specifically, let d denote the (Euclidean) distance between the measured image point $\tilde{\mathbf{x}} = (\tilde{x},\tilde{y})^T$ and the reprojected image point $\mathbf{x} = (x,y)^T$, that is, $d = ||(\tilde{x}-x,\tilde{y}-y)||$. From

the camera equation, we know that the reprojected point is obtained through

$$\lambda \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} = P\mathbf{X}$$

where P is the camera matrix and $\mathbf{X}$ the homogeneous coordinates of the 3D point. Let $\mathbf{p}_i$ denote the ith row of camera matrix P. The distance d can then be expressed by

$$d = ||(\tilde{x} - \frac{\mathbf{p}_1^T\mathbf{X}}{\mathbf{p}_3^T\mathbf{X}}, \tilde{y} - \frac{\mathbf{p}_2^T\mathbf{X}}{\mathbf{p}_3^T\mathbf{X}})|| = \frac{||(\tilde{x}\mathbf{p}_3^T\mathbf{X} - \mathbf{p}_1^T\mathbf{X}, \tilde{y}\mathbf{p}_3^T\mathbf{X} - \mathbf{p}_2^T\mathbf{X})||}{\mathbf{p}_3^T\mathbf{X}}.$$

Now suppose we are given a number of measured points $\tilde{\mathbf{x}}_i$. Instead of minimizing the sum of squared distances $\sum_i d_i^2$, we will minimize $\max_i d_i$. This can be hard to do directly, but if we introduce an extra unknown scalar α (serving as an upper bound), the problem can equivalently be written

$$\begin{array}{ll} \min & \alpha \\ \text{such that} & d_i \leq \alpha, \end{array}$$

and things will become simpler. The reason is that for a given, hypothesized α, the problem of checking if there is a solution with $d_i \leq \alpha$ for all i is usually *easy* and can be solved by intersecting convex cones. Each constraint $d_i \leq \alpha$ can be geometrically interpreted as a second-order cone (just as the solution set to the inequality $\mathbf{a}^T\mathbf{x} \leq b$ can be interpreted as a half space). If such a solution exists, then the optimal α_{opt} must be lower, otherwise the optimal α_{opt} must be higher than α. Hence, we have turned the optimization problem into a one-dimensional search for α_{opt}.

The Triangulation Problem

We will first look at the triangulation problem. A synthetic three-view problem has already been generated. Type

```
load sam_example
```

to load the data for this example. The image coordinates can be found in `u` and the camera matrices in `P`. Type

```
drawscene
```

to plot the configuration of the three image planes with one measured point in each image. The blue lines correspond to the viewing rays of the image points. Recall that a viewing ray for an image point is the set of points in front of the camera that projects exactly to the image point. You can rotate the figure by holding down the left mouse button and moving the mouse pointer at the same time.

The triangulation problem consists of finding a 3D point $\mathbf{X}$ such that its projection is as close to the measured image point as possible. Notice that the blue lines do not intersect, so there does not exist a 3D point which projects exactly to the image points. Suppose we

start by looking for a 3D point which projects within 2 pixels in one of the images. Then the 3D points must lie with a viewing cone whose radius is 2 when crossing the image plane. Type

```
nbrcones=1;
radius=2;
drawscene
```

to plot such a viewing cone. Change `nbrcones` to first 2 and then 3 to study the other viewing cones.

Question: Is there a 3D point in the intersection of the three cones and hence a 3D point which projects to less than 2 pixels in all three images?

Try other values of the radius for the viewing cones, for example:

```
nbrcones=3;
radius=0.5;
drawscene
```

Question: Is there a 3D point which projects to less than 0.5 pixels in all three images?

Try to figure out approximately what the smallest radius is such that there is a common intersection of all three viewing cones by visually examining the figure for different values of `radius`.

Checking whether the intersection of three (second-order) cones is empty or not can also be done by solving a convex program. This has already been implemented for you. Try the following commands:

```
radius=2;
U = intersect_cones(u,P,radius);
```

If the intersection is non-empty, a 3D point `U` is returned which lies in the intersection.

Question: What are the **actual** image reprojection errors for the point `U` which is returned with `radius=2`? (It must be less than 2 pixels, right?)

Try to find out approximately what the smallest radius is such that there is a common intersection of all three viewing cones by testing different values of `radius` for the function `intersect_cones`.

The one-dimensional search for the smallest radius can also be automated by using *bisection*. This has been implemented in a function called `linf_triangulation`. It can be called by the following command.

```
U = linf_triangulation(u(1:2,:),P);
```

Question: What is the min-max reprojection error for the three-view triangulation problem? Is the maximum reprojection error attained in all three images?

Homographies and Panoramas

Two (or more) images that are taken at the same location (that is, the corresponding camera centres are the same) are homographically related. This means that there exists a homography which can be represented by a 3×3 matrix H such that if a point in image 2 has coordinates $\mathbf{x}_2$, then the corresponding point in image 1 can be found at position $H\mathbf{x}_2$. This observation is frequently used for creating panoramic images by stitching together several images to one big mosaic.

Given corresponding points in two images, it is possible to estimate the homography relating the images. A homography has 8 degrees of freedom, 9 parameters defined up to scale. (What is the minimum number of corresponding points needed to be able to compute it?) Moreover, it is possible to estimate the homography using min-max optimization. The difference compared to the triangulation problem is that now the convex cones are parametrized by the 8 degrees of freedom in the homography. The computations are done in a similar manner: we need to check a series of cone intersections and determine if they are empty or not in order to determine the maximum residual error.

Load and plot two images of a mountain including 6 corresponding image points, stored in the variables `u1` and `u2`. The images were taken at the same location.

```
load snowpoints;
im1=imread('snow1.jpg');
im2=imread('snow2.jpg');
figure(1);clf;subplot(1,2,1);
image(im1);
hold on;rita(u1,'g*');
subplot(1,2,2);
image(im2);
hold on;rita(u2,'g*');
```

The homograpy relating the two images can be estimated with the following command.

```
H=linf_homography(u1(1:2,:),u2);
```

Verify the quality of the computed homography H by transferring the points in image 2 to image 1.

```
u2t=pflat(H*u2);
subplot(1,2,1);
rita(u2t,'ro');
```

Zoom in on the original points (green stars) and the transferred points (red circles).

Question: What is the maximum residual error for the estimated homography H? How many of the correspondences attain the maximum residual error?

Click on some more point locations in image 2 and transfer them to image 1 to see that this is not only true for the 6 original correspondences. The following commands can be used.

```
while 1,
    disp('click on a point in image 2');
    x=ginput(1)';
    subplot(1,2,2);
    rita(x,'b+');
    subplot(1,2,1);
    rita(pflat(H*[x;1]),'b+');
end
```

Alright, now the images can be stitched together to create a large mosaic of the two pictures! Look at the script `stitch.m` and try it out. The stitching for this example is rather primitive as you can see from the result.

Problem: Suggest a method to make the transition between the two images smoother in the resulting panoramic image.

Known Rotations

Another problem that can be solved with min-max optimization is the structure and motion problem with *known* rotations. If we are given sufficient number image correspondences $\tilde{\mathbf{x}}_{ij}$ and the first 3×3 block of the camera matrix R_i, then the problem of estimating 3D points $\mathbf{X}_j$ and camera centres $\mathbf{t}_i$ (corresponding to the fourth column of the camera matrix) can be solved globally with the same technique. In detail, we want to find the min-max reprojection error between the measured point $\tilde{\mathbf{x}}_{ij}$ and the reprojected point $\mathbf{x}_{ij}$, given by

$$\lambda_{ij}\mathbf{x}_{ij} = [R_i\,\mathbf{t}_i]\mathbf{X}_j \text{ for all } i, j.$$

All we need to do is solve a series of cone intersection problems. The difference is that the corresponding cones are not in $\mathbb{R}^3$ anymore (as for the triangulation problem), but in a higher dimension. The dimension is equal to the number of unknowns in the problem instance.

Let us look at an example. We have already tracked 328 points in 36 images in a turntable sequence of a toy dinosaur. The rotations have been computed by some other means. Load and plot the first image. The image points are stored in the variable `u` and the 36 rotation matrices in `R`.

```
load dino_example;
im=imread('dino.jpg');
figure(1);clf;image(im);hold on;
rita(u{1},'g*')
```

Note that only a few points are visible in the first image, and not all of the 328 points. This is common in real image sequences - image points will appear and then later disappear and something we have to cope with.

To intersect the viewing cones, try the following commands:

```
radius=5;
[U,P]=intersect_cones_rot(u,R,radius)
```

If there is a non-empty intersection, then one point in the intersection set is returned. Such an intersection point corresponds to 328 3D-points `U` and 36 camera matrices `P`. Try different values of `radius`.

Question: What is approximately the min-max image reprojection error for the dinosaur sequence?

One way to check the quality of the computed solution is to reproject it to the image. Try:

```
figure(1);
ureproj=pflat(P{1}*U);
rita(ureproj,'ro');
```

Zoom in to see if there is a reprojected point (red circle) corresponding to each measured image point (green star).

Another way to verify the result is to plot the solution in 3D, both the camera path and the 3D points.

```
figure(2);clf;
rita3(U,'r*');
hold on;
for ii=1:36;
c=pflat(null(P{ii}));
rita3(c,'b*');
end
rotate3d on;
```

Can you see the shape of the dinosaur? Is the camera trajectory approximately circular as it should be due to the turn-table setup? If the answer is yes: congratulations! You have just reconstructed 36 cameras and 328 points and not only that, you have found the globally optimal solution with respect to the L_∞-norm.

Implement on your own: Triangulation with Linear Programming

Three camera matrices (of size 3×4) P_1, P_2 and P_3 are pre-computed for a scene and can be found in the matlab file `cameras.mat`. Further, assume there is a measured image point with (x,y)-coordinates $(0,0)$ in each image (that is, the same coordinates in all three images).

Image points $\mathbf{x}$ lying on a line $\mathbf{l}$ satisfy $\mathbf{l}^T\mathbf{x} = 0$ where $\mathbf{l}$ and $\mathbf{x}$ are homogeneous 3-vectors. Similarly, 3D points $\mathbf{X}$ on a plane π satisfy $\pi^T\mathbf{X} = 0$ where π and $\mathbf{X}$ are homogeneous 4-vectors. Given a camera matrix P, then all points on $\pi = P^T\mathbf{l}$ project to the image line $\mathbf{l}$. (Why?) Around each image point, it is possible to form a square with side length s. The four lines forming this square are given by $x = \pm s/2$ and $y = \pm s/2$.

Questions: What are the corresponding 3D planes describing the viewing cones of 3D points that project to this square? Express the answer using a camera matrix P and the side length s. Determine if the three viewing cones intersect for side lengths $s = 4$, $s = 2$ and $s = 1$ and if so give a 3D point in the intersection. *Hint:* Each plane $aX + bY + cZ + d = 0$ divides space into two halfspaces $aX + bY + cZ + d \leq 0$ and $aX + bY + cZ + d \geq 0$. Use this to express the cones on the form $A[X,Y,Z]^T \leq b$ and then utilize Matlab: `X=linprog([],A,b);` to compute the intersection. What is approximately the smallest s in order for the cones to have an intersection?

Optional: Back to Stitching

Implement your proposal of removing the border effects in the stitching problem in order to create a visually appealing panorama.