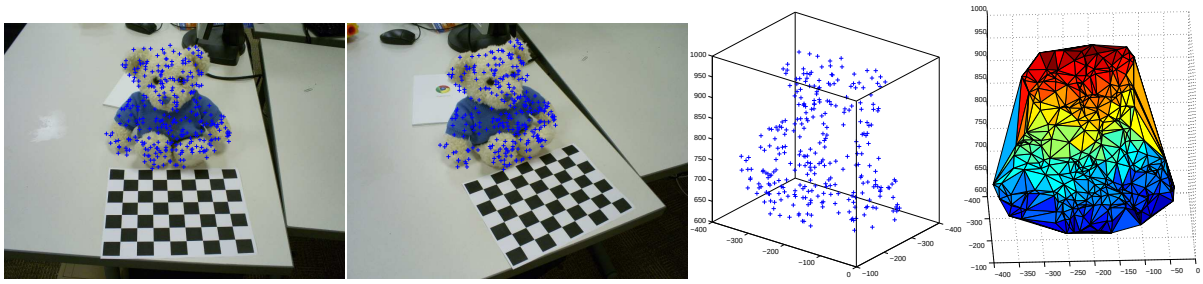
3D Reconstruction from Multiple Views



I. 3D Reconstruction Using Multiple View Images [60 points]

1) Take pictures of a small object near a checkerboard. You should take at least 6 pictures from different view points. If you have a zoom camera, do not change the zoom when you take pictures. If you do not have a camera, come to vision center, I will set up a place for image capturing.
2) Calibrate your camera using the camera calibration toolbox at http://www.vision.caltech.edu/bouguetj/calib_doc/. You need to download the toolbox. Before using the functions in the toolbox, be sure to set the path in Matlab by issuing the command:

   ```
   path(path, 'your_toolbox_path');
   ```

   where `your_toolbox_path` is the directory of the camera toolbox. After calibrating your camera and saving the result, a file named as `Calib_Results.mat` will be stored in your current directory. Check the projection error. It should be less than 1 pixel.
3) Compute the camera matrix. You can compute the first camera matrix using

   ```
   load('Calib_Results.mat');
   kk = [fc(1) 0 cc(1); 0 fc(2) cc(2); 0 0 1];
   P1 = kk*[Rc_1,Tc_1];
   ```

   The second one is `P2 = kk*[Rc_2,Tc_2]` and so on. Write your camera parameters and matrices in the report.
4) Compute your camera projective centers for each view and write them in your report. The first camera projective center is

   ```
   C1 = inv(P1(1:3, 1:3)) * (-P1(:,4));
   ```

5) Choose two views. Write a program that accepts user clicks on the corresponding point pairs on the two images and plot the reconstructed 3D point clouds. Put the result on your report. You can use `ginput` to get user input. For instance, you can show one image in `figure(1)` and the other is in `figure(2)`. Use the following script to get user mouse click location:

   ```
   figure(1);
   [x1, y1] = ginput(1);
   figure(2);
   [x2, y2] = ginput(2);
   ```

   Call the following Matlab function to reconstruct the 3D points:

   ```
   function M = triangulate(P1, m1, P2, m2)

   % TRIANGULATE computes the 3D point location using 2D camera views
   % P1: camera matrix of the first camera.
   % m1: pixel location (x1, y1) on the first view. Row vector.
   % P2: camera matrix of the second camera
   % m2: pixel location (x2, y2) on the second view. Row vector.
   % M: the (x, y, z) coordinate of the reconstructed 3D point. Row vector.

   % Camera one
   C1 = inv(P1(1:3, 1:3)) * (-P1(:,4));
   x0 = C1(1);
   ```

```
y0 = C1(2);
z0 = C1(3);
m1 = [m1’; 1];
M1 = pinv(P1) * m1;
x = M1(1)/M1(4);
y = M1(2)/M1(4);
z = M1(3)/M1(4);
a = x-x0;
b = y-y0;
c = z-z0;
% Camera Two
C2 = inv(P2(1:3, 1:3)) * (-P2(:,4));
x1 = C2(1);
y1 = C2(2);
z1 = C2(3);
m2 = [m2’; 1];
M2 = pinv(P2) * m2;
x = M2(1)/M2(4);
y = M2(2)/M2(4);
z = M2(3)/M2(4);
d = x-x1;
e = y-y1;
f = z-z1;
% Solve u and v
A = [a^2 + b^2 + c^2,  -(a*d + e*b + f*c);...
     -(a*d + e*b + f*c), d^2 + e^2 + f^2];
v = [ (x1-x0)*a + (y1-y0)*b + (z1-z0)*c;...
      (x0-x1)*d + (y0-y1)*e + (z0-z1)*f];

r = inv(A) * v;
M = [x0+a*r(1) y0+b*r(1) z0+c*r(1)];
```

Show the 3D points in your report. It helps to use the same unit in 3 different axises:

```
plot3(points(:,1), points(:,2), points(:,3), ’+’);
set(gca, ’DataAspectRatio’, [1 1 1]);
```

Here `points` contain the 3D points you reconstructed. Rotate the point cloud and see whether the reconstruction makes sense. You can also generate an interpolated surface using the following procedure. `x` and `y` are image points in either of the two views.

```
tri = delaunay(x(:,1), y(:,2));
trisurf(tri, points(:,1), points(:,2), points(:,3));
set(gca, ’DataAspectRatio’, [1 1 1]);
```

II. Bonus question (50 points)

Automatically find the corresponding points and use multiple pictures to reconstruct a real 3D cloud of the object.