Introduction to Computer Vision Homework 3

Jeeeun Kim (Jeeeun.kim@colorado.edu)

Due in one week in class. All renderings should be done in Matlab, saved as some form of scalar vector graphic (such as ps, eps, or pdf) and included in your final .pdf report (typeset in LATEX).

1. Download this data file: http://arpg-web.colorado.edu/~arpg/cv_class.tar.gz which contains images of a Calibu camera calibration target. Build vicalib (see notes online)
(a) Unzipcv_class.tar.gzinthevicalibbuild-applicationsdirectory(./vi- calib/build/applications/vicalib).

2. Produce a conics.csv file
(a) Download, build and run the program "vicalib" with the dot extractor using the command line:

```
$ ./vicalib_exec -output_conics -grid_preset medium -cam file://./calib_dataset/pgm/*.pgm
```

(b) Each line of conics.csv is <frame number, dot id, u, v, x, y, z>

3. Matlab programing:
(a) Load and display the points from conics.csv.

```
global cmod;

pts = load('conics.csv');
pts(pts(:,2)<0,:) = [];  % remove negative id

[i,j]= unique(pts(:,2)); % remove duplicates of points id at same frame
                         % i = frame#, j = point# at that frame

xwp = pts(j,5:7)'        % xwp = 3d porints, 3 by N matrix
ids = pts(j,2)';         % points id

ctx.nImg = numel(unique(pts(:,1)));
ctx.target_pts = xwp;

for ii = 0:ctx.nImg-1 %img frame: 0 ~ 105 (total 106 frames)
    ctx.cur_ids = pts(find(pts(:,1) == ii), 2);   % match points and frame
    ctx.z       = pts(find(pts(:,1) == ii), 2:4)';

    hold on
    axis ij; xlim([0 800]); ylim([0 600]);
    project();
end

function p = project()
    hold on;
    plot(ctx.z(1,:), ctx.z(2,:), '.');

    pause(0.02);
    clf();
end
```

Figure 1. Load uv points into uv(ctx.z) and plot

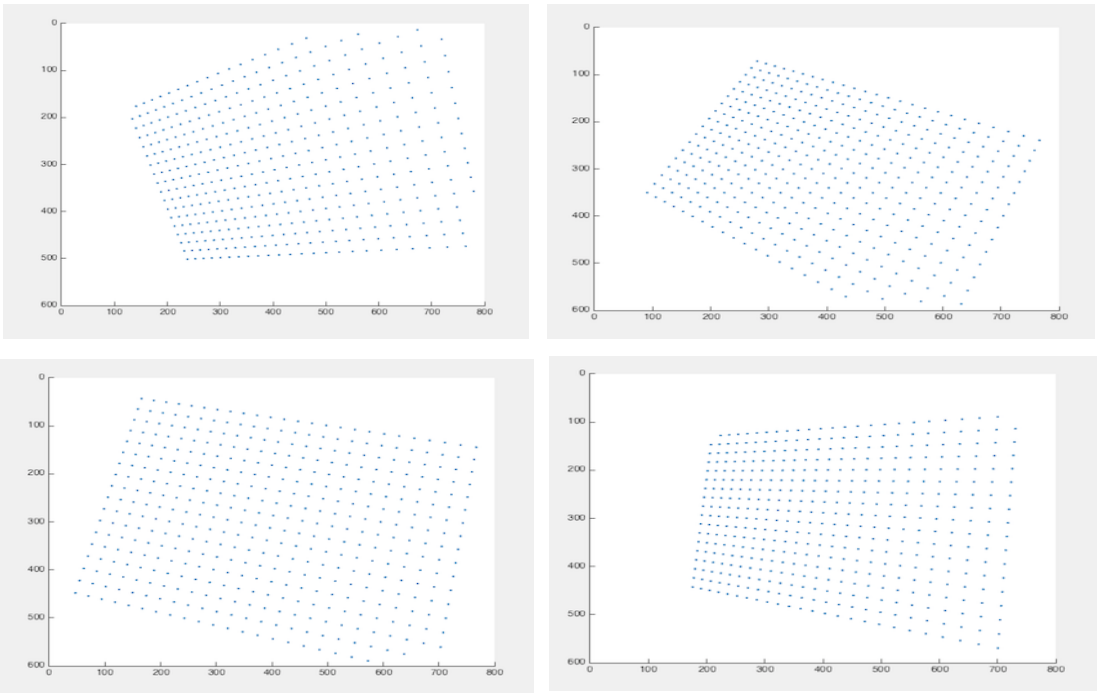


Figure2. 2D(u,v) points (3rd and 4th column values at each frame set)

(b) Extract the target coordinates from conics.csv (set of unique 3D points from the last three columns); make sure they can be indexed by id.

```
ctx.target_pts = unique(pts(:,[2,5:7]), 'rows');
ctx.target_pts(ctx.target_pts(:,1)==-1,:) = []; %clearn bad ids
```

Figure 3. Target 3D points read from .csv file

(c) Given an initial camera pose, project the target into an image. Display the image and the measured values for this frame.

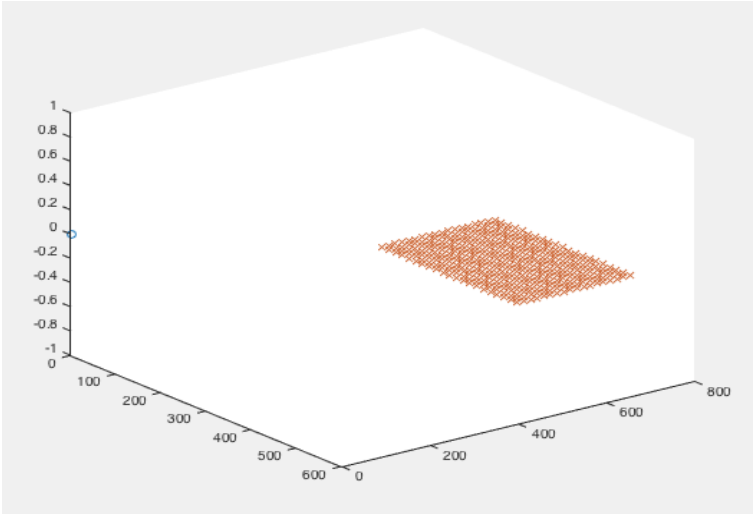


Figure 4. Initial points (pts columns 5:7) plotted into 3D world

(d) Write a function r=residual(x) to compute the residual vector which is the difference between all measured and predicted points.

```
function r = residual( x, ctx )

    global cmod;

    Twc = Cart2T(x);

    res = zeros(size(points3D,1), 5);
    res(:,2:3) = proj_3d_to_2d(Twc, ctx.target_pts(1:3), cmod);
    res(:,1) = ctx.cur_ids;

    for i = 0:ctx.nImg-1;
      hx = uv(uv(:,1) == i, 2:3);          % hx = proj_3d_to_2d(Twc, xcp, cmod);
      if hx
         res(res(:,1)==i, 4:5) = hx;
      else
         res(res(:,1)==i, :) = [];
      end
  end

  bad_id = isnan(res(:,4)) | isnan(res(:,5));
  res(bad_id ,:) = [];

  r(:,1:2) = res(:,4:5) - res(:,2:3);      % residual is difference btw original and
                                           % measured 2D points u,v from 3D points

end
```

Figure 5. function r = residual,
finding differences between calculated values and predicted value

(e) Use lsqnonlin to solve for the x that minimizes r. Plot the resulting pose, together with all the 3D target dots.

(f) Each time lsqnonlin calls your function, plot the current predicted measurements, based on x.

(g) Include all frames in one giant residual vector over all measurements – NB the dimension of x is now 6N ×1 if there are N frames.

```
    pose = [0 0 -1 pi/2 0 0]';

    for i = 0:ctx.nImg-1
        uv = ctx.target_pts(ctx.target_ptx(:,1) == i, 2:4);

        % 3-(e) use lsqnonlin() to minimize rediduals
        options = optimoptions('lsqnonlin', 'Algorithm', 'trust-region-reflective',
                               'TolFun', 1.0e-8, 'TolX', 1.0e-8);
        xwc = lsqnonlin(@residual, pose, [], [], options);

        pose = xwc;
        Twc = cart2t(pose);
        hx = proj_3d_to_2d(Twc, points3D, cmod);
```

```
        % 3-(f) plot predicted measurements every time lsqnonlin() called
        hold on
        axis ij; xlim([0 800]); ylim([0 600]);
        plot(hx(1,:), hx(2,:), 'o');

        pause(0.02);
        clf;

        % 3-(g) Include all frames in a giant residual vector 6N by 1
        camera_pose(:,i+1) = xwc;

end
```

Figure 6. lsqnonlin(@residual, xwc) to minimize residuals

(h) Plot the resulting poses (using plot_cf).

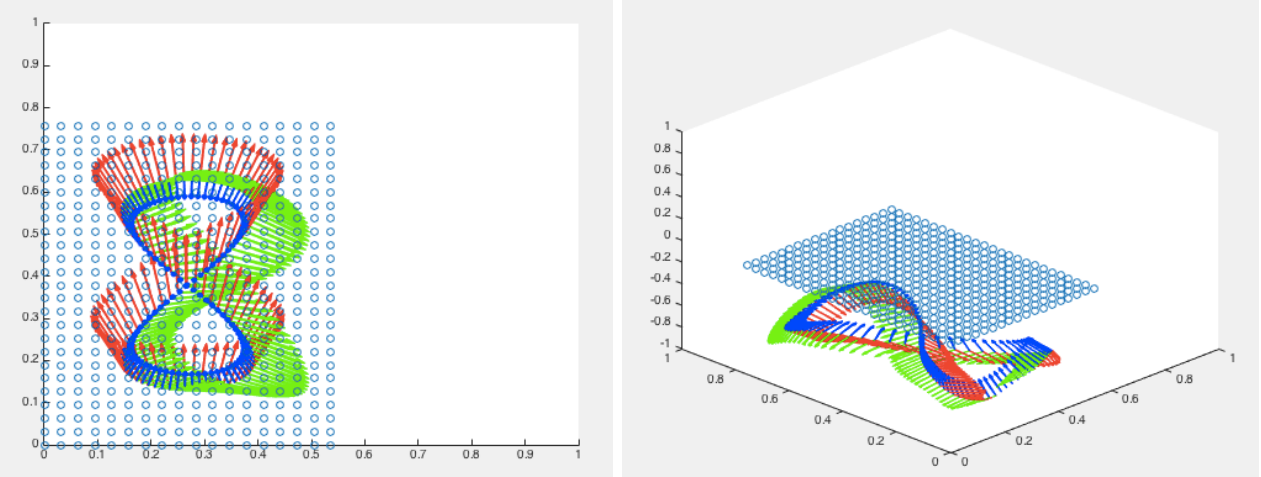


Figure 7. Resulting poses from top view (left), side view (right) at each frame

4. Matlab programing:

(a) Include the camera calibration parameters in the vector x. How many parameters are there now? Run the optimization again. Does it converge?

```
... (same as above)

poseN = [0 0 -1 pi/2 0 0]';                          % initial pose
pose = [poseN; cmod.fx; cmod.fy; cmod.cx; cmod.cy]; % 10 parameters including camera calibration

... (same as above)

    for i = 0:ctx.nImg-1
        uv = conics(conics(:,1) == i, 2:4);
        options = optimoptions('lsqnonlin', 'Algorithm', 'trust-region-reflective',
                               'TolFun', 1.0e-8, 'TolX', 1.0e-8);
        xwc = lsqnonlin(@residual, pose, [], [], options);
```

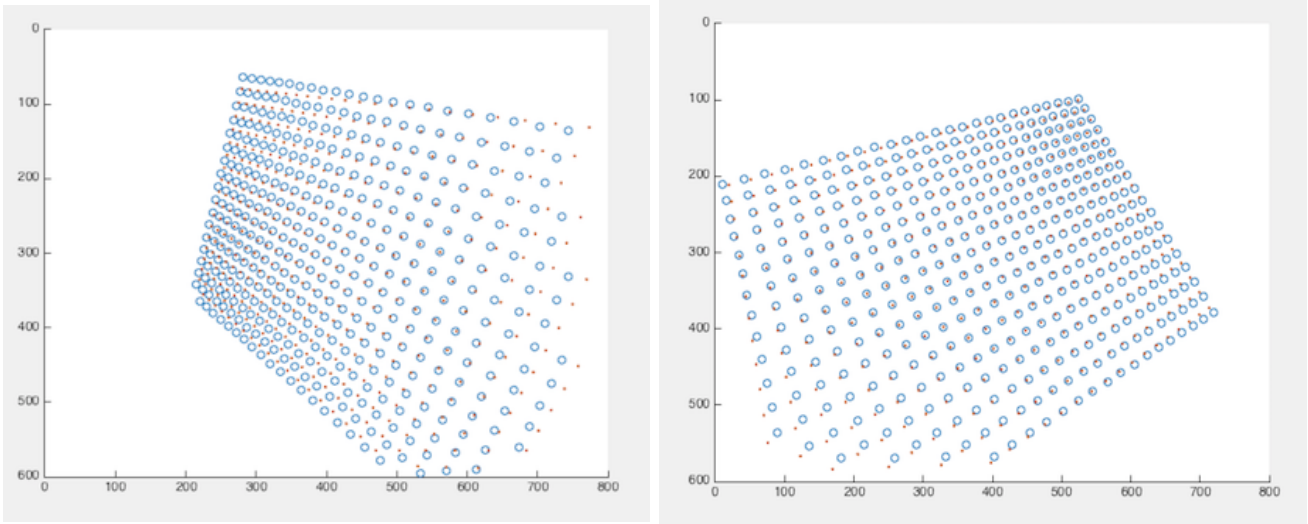
```
        pose = xwc;
        Twc = cart2t(pose(1:6)); % retrieve only pose(6 by 1) information, and pass for next run

... (same as above)

    end % end of for loop
```

```
function r = residual( x, ctx )

... (same as above)

    Twc = Cart2T( x(1:6) ); % pass only pose(6 by 1) to get twc

    cmod.fx = x(7);       % renew camera intrinsic to call proj_3d_to_2d() each time
    cmod.fy = x(8);
    cmod.cx = x(9);
    cmod.cy = x(10);

... (same as above)

end
```

```
plot_cf2(camera_pose(1:6,i+1));
```

Figure 8. Programs and functions to pass 10 by 1 parameter x = [xwc; cmod.fx; cmod.fy; cmod.cx; cmod.cy] everytime to guess camera pose along with camera intrinsic

Total 10 parameters are passed to function r = residual(x), not including image width and image height. Yes, it converge into prospective uv points (ctx.z)



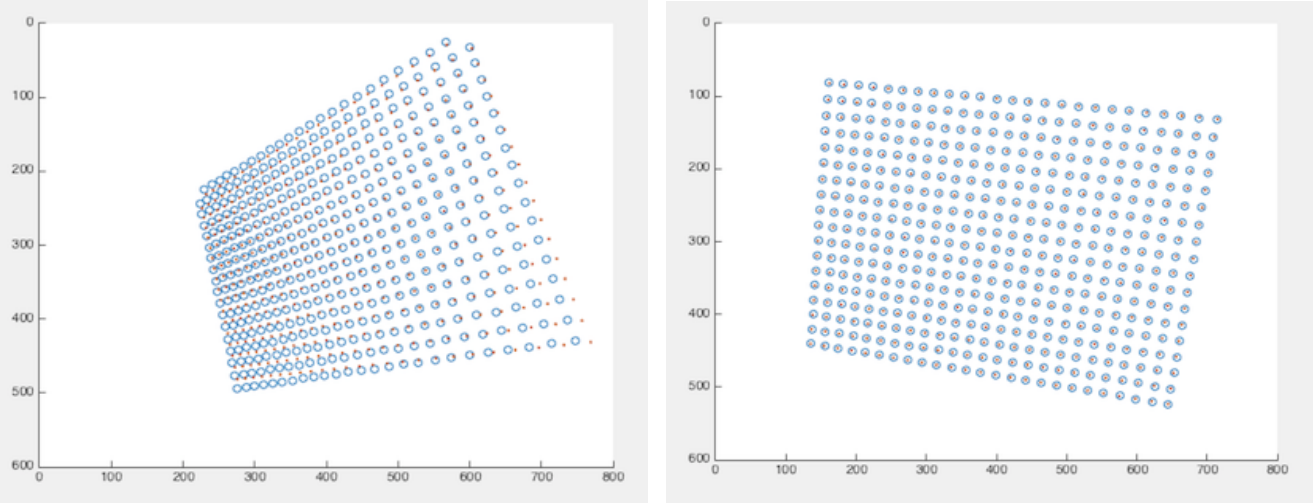


Figure 9. Plotting original uv(ctx.z, blue circles)
and projected 3D points(ctx.target_pts, red dots) into 2D plane at the same time, which is converging.

5. Modify your residual function to also return the Jacobian of the residual vector [r, J] = residual (x).

(a) What is the dimension of J
2 by 6

(b) Write a generic function FiniteDiff.m that takes function f(x) and returns the residual and Jacobian J

(c) Write the Gauss-Newton method, `x = GaussNewton(@(x)residual(x,ctx), x0)`, to solve for

argmin residual(x)

```
function [r, J] = residual_Jacobian(x, ctx)

    global cmod;

    r = residual(x, ctx);
    J = FiniteDiff(@residual, x, ctx);

End
```

Figure 10. Additional residual function that takes pose and target points, and return both residual r and Jacobian

```
function D = FiniteDiff( varargin )

    f   = varargin{1};        % residual function handler
    x   = varargin{2};        % initial 6 by 1 pose
    ctx = varargin{3};

    dEps = 1e-4;              % should be tweaked

    y = feval(f, x, ctx);     % y is residual of u,v at each frame, 2 by 1
    J = zeros(numel(y), numel(x) );
```

```
    for ii = (1:numel(x))
        xPlus = x;
        xPlus(ii) = xPlus(ii) + dEps;
        yPlus = feval(f, xPlus, ctx);  % yPlus = 2 by 1

        xMinus = x;
        xMinus(ii) = xMinus(ii) - dEps;
        yMinus = feval(f, xMinus, ctx); % yMinus = 2 by 1

        J(:,ii) = (yPlus-yMinus)./(2*dEPS);
    end

end
```

6. Download https://arpg.colorado.edu/wp-content/uploads/conics2.csv
(a) What are the camera intrinsics for this dataset?

$$K = \begin{bmatrix} 376.5758 & 0 & 399.4792 \\ 0 & 375.4656 & 299.4964 \\ 0 & 0 & 1 \end{bmatrix}$$