Signature Cams

William Ard, Hector Arredondo Ramirez, Miranda Pepper

ME 4133: Machine Design I

Louisiana State University

Abstract

This project presents the early stage of the development of a cam-driven automatic signature mechanism. The MATLAB package included with this project allows for the generation and analysis of the mechanism's cams. Users can input an image of their signature and generate points. The points can then be used to create custom cam profiles for a signature and analyze that profile. The software package generates cam displacement, velocity, acceleration, and jerk diagrams, in addition to radius of curvature and pressure angle diagrams; the generated cam profiles are saved in a SolidWorks compatible format that can be used to create solid models for manufacturing purposes.

Table of Contents

Introduction

The goal of this paper is to present a feasible design for a cam-driven automatic signature device. Major design constraints in development of the design were that the device be small enough to fit on a table and that any person should be able to use the code generated for cam profile synthesis to generate a 2”x1” replication of their own signature. The device presented in this report also incorporates pen lifting between each initial. This report is divided into two sections: the first section outlines the signature mechanism itself, and the second section discusses cam profile synthesis.

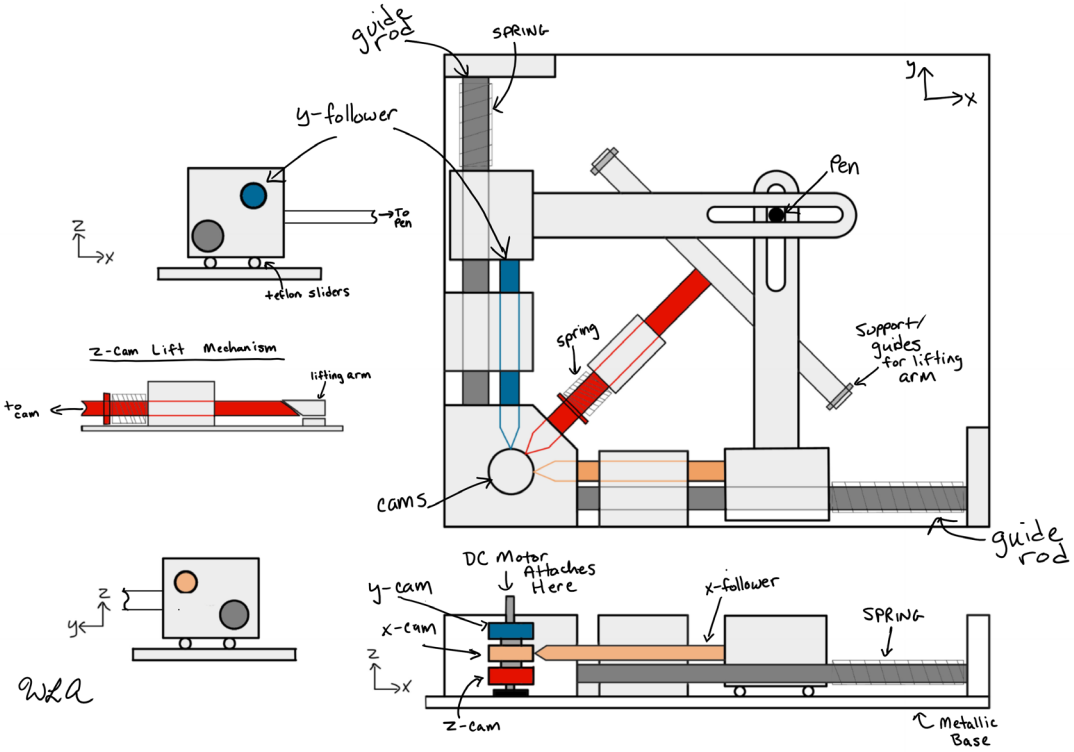

Figure 1: Mechanism Sketch (not to scale)

Mechanism Design

The automatic signature mechanism uses three cams on a single camshaft, driven by a variable speed, electric DC motor. Each cam controls displacement along a single axis, at a 1:1 ratio, within a standard cartesian coordinate system. The pen is attached in a slot inside of two guided arms. Guide rods ensure that each arm follows a straight path, and springs in compression ensure that each knife-edge follower maintains constant contact with each cam surface. Pen lifting is accomplished through the use of a lifting arm that sits beneath the other arms. When the z follower is actuated beyond the z cam’s base radius, the lifting arm is pushed upwards, raising the x and y arms from the paper. The entire mechanism is mounted on a 24”x24” metallic base, and paper is secured to the base through the use of magnets. More design sketches and a preliminary solid model are shown in Appendix G.

Cam Profile Synthesis

For each set of initials, three cam profiles are generated. Two MATLAB scripts are for profile synthesis—one for defining precision points for the signature, and another for the generation of cam profiles. The point generation script uses MATLAB's `getpts` function. Precision points are manually designated by clicking on an image loaded into MATLAB, following the signature path. This defines a path for cam function generation. At the end of each initial, the point set is completed by pressing the return key. Once six x-y parametric point-paths have been generated (one for each initial, and one for each lifted section), the paths are read by a second MATLAB script.

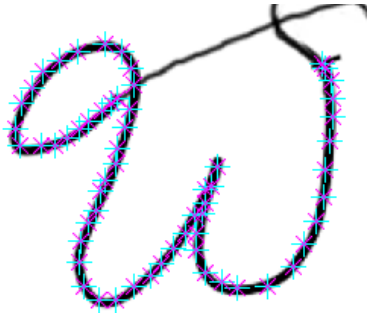

Figure 2: Point/Path Generation

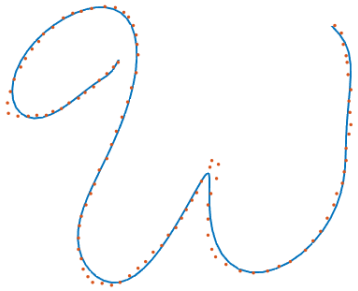

Figure 3: Fourier Series Fitting

The second script is responsible for curve fitting the defined path, generating cam functions, and producing plots for cam analysis. For each cam, *s v a j*, radius of curvature, and pressure angle plots are automatically generated. At the top of the script, cam base radiuses and maximum z follower displacement can be specified. Code was written with the assumption that an in-line, knife edge follower would be used, so formulations for radius of curvature and pressure angle assume a follower radius and cam eccentricity of zero; instant center analysis was not necessary for this cam-follower configuration.

X and Y cam functions for each letter are generated through the use of MATLAB's fitting routines. Each letter, and each lifted region, are approximated by separate 8^{th} order Fourier series equations. Since the fits use Fourier series approximations, the path functions defined do not pass through every precision point, but instead produce smooth, close approximations. The high-order differentiability, continuity, and smoothness of the Fourier series functions allow for viable cam profiles to be generated. If no path smoothing is used, the cam profiles tend to be very jagged, and discontinuities in the displacement function lead to poorly generated cams.

Once the Fourier series functions are generated, they are evaluated through the full range of the cam. The output of these functions is then splined together through MATLAB's `smoothingspline` fit. This ensures continuity between each path approximated by the Fourier series fit. It was found that the `smoothingspline` by itself is inadequate for automated cam function generation, this is why both the Fourier series and smoothing spline fits are used. The coefficient matrix for the functions generated is accessible through MATLAB, and can be called in the program through the `x_fit` and `y_fit` structures. Due to the number of equations that the fit represents (one 4th order equation for each spline, with 300-400 splines per cam), they are not included in this report. The z cam is generated through the use of harmonic functions: $\frac{1}{2Lh}(1 - cos\frac{2\pi(x-st)}{\Delta\theta})$ where Lh = maximum lift height, st = starting point of lift, and $\Delta\theta$ = range of lifted section.

Once the cam functions, profile, and analysis plots are generated, it's possible to check the *s, v, a,* and *j* plots for discontinuities, verify that pressure angle and radius of curvature is within the necessary range, and adjust the base radius of the cam at the beginning of the script if necessary. The MATLAB script also outputs the profile of the cams generated in a format that can be loaded into SolidWorks and used to generate solid models for manufacturing.

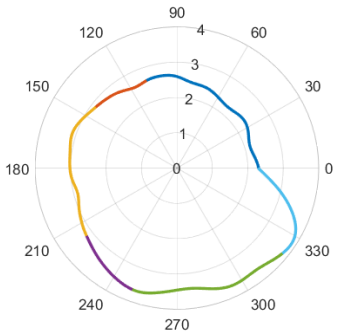

Figure 4: X Cam Profile "W"

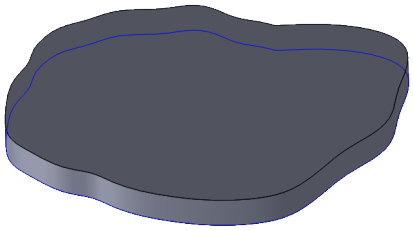

Figure 5: SolidWorks Model of "W" X Cam

Conclusion

This report lays out the basic plan for the development of a signature cam mechanism. The next phase for the development would be to complete a dynamic analysis of the cams and mechanism proposed. One potential issue with this proposed method of cam profile generation is that of point selection. Quality of fit for the curves is dependent on user input. Automation of path generation could produce better cam profiles. The development of a mechanism that allows for the scaling of signatures could also be considered. Alternate curve fitting techniques could be explored as well, but the design and method presented does provide a first-generation model.

Appendix A: Point Generation Code

```
% Point Generator (William)

% Click points on image, tracing signature in the process. Press enter key
% at the end of the initial. Continue clicking through the path while
% tracing the connection between the 1st and 2nd initial. Press enter to
% record the path. Repeat this process through the entire image. This will
% generate 6 sets of x-y data: one for each initial and one for each
% connecting path.

% Points will be saved in a .mat file. This file can be loaded in the
% camgen.m file to generate the cams.


sig = imread('signature_miranda.png'); % Must be 2:1 (x:y) Resolution!

signew = imresize(sig, [600 1200]);
imshow(signew)

[x1,y1]=getpts(); % 1st Initial
[xa,ya]=getpts(); % Connection B/W 1st & 2nd Initial
[x2,y2]=getpts(); % 2nd Initial
[xb,yb]=getpts(); % Connection B/W 2st & 3rd Initial
[x3,y3]=getpts(); % 3rd Initial
[xc,yc]=getpts(); % Connection B/W 3rd & 1st Initial



%Store x-y points in structure

points.x1 = x1;
points.xa = xa;
points.x2 = x2;
points.xb = xb;
points.x3 = x3;
points.xc = xc;

points.y1 = y1;
points.ya = ya;
points.y2 = y2;
points.yb = yb;
points.y3 = y3;
points.yc = yc;

save('mcp.mat', 'points')
```

Appendix B. Cam Profile Synthesis Code

```
% Cam Profile Generator (Coded by William)
% (SolidWorks Export & Pressure Angle coded by Hector)

% Specify points for signature to load below.
% Define cam base radiuses & z-follower displacement.
% Press F5 to run.


load('mcp.mat') % Specify point set to load here.

yDPI = 600; % Shoudn't have to change this if image was properly resized in pointgen.m


% Specify Base Radiuses & Max Z Follower Displacement (pen lift height)

Lh = .25;   % Max Z Follower Displacement (inches)
Brx = 2;  % X Cam Base Radius (inches)
Bry = 2;  % Y Cam Base Radius (inches)
Brz = 2;  % Z Cam Base Radius (inches)



%% Do not modify below this line

% Flip Coordinates of Y Values to Normal Coordinate System

points.y1 = yDPI - points.y1;
points.y2 = yDPI - points.y2;
points.y3 = yDPI - points.y3;
points.ya = yDPI - points.ya;
points.yb = yDPI - points.yb;
points.yc = yDPI - points.yc;


% Define span for each initial, and regions were pen is lifted

range.r1 = 1:numel(points.x1);
range.ra = 1:numel(points.xa);
range.r2 = 1:numel(points.x2);
range.rb = 1:numel(points.xb);
range.r3 = 1:numel(points.x3);
range.rc = 1:numel(points.xc);

span.s1 = 1 : length(range.r1);
span.sa = length(range.r1) : length(range.r1)+length(range.ra);
span.s2 = length(range.r1)+length(range.ra) :
length(range.r1)+length(range.ra)+length(range.r2);
span.sb = length(range.r1)+length(range.ra)+length(range.r2) :
length(range.r1)+length(range.ra)+length(range.r2)+length(range.rb);
span.s3 = length(range.r1)+length(range.ra)+length(range.r2)+length(range.rb) :
length(range.r1)+length(range.ra)+length(range.r2)+length(range.rb)+length(range.r3);
span.sc = length(range.r1)+length(range.ra)+length(range.r2)+length(range.rb)+length(range.r3)
:
length(range.r1)+length(range.ra)+length(range.r2)+length(range.rb)+length(range.r3)+length(ra
nge.rc);


% Define a vector of all displacement points for X & Y
```

```
x_disp_points = [points.x1; points.xa; points.x2; points.xb; points.x3; points.xc];
y_disp_points = [points.y1; points.ya; points.y2; points.yb; points.y3; points.yc];

range_full = 1:numel(x_disp_points);    % Length of vectors
range_full = range_full';               % Convert to column vector



% Build Smoothing Functions for Data

% Generate Fourier Series fits for each initial and lifted region

fsfit.x1fit = fit( range.r1', points.x1,'fourier8');    % 1st Initial
fsfit.y1fit = fit( range.r1', points.y1,'fourier8');
fsfit.xafit = fit( range.ra', points.xa,'fourier2');    % Lift
fsfit.yafit = fit( range.ra', points.ya,'fourier2');
fsfit.x2fit = fit( range.r2', points.x2,'fourier8');    % 2nd Initial
fsfit.y2fit = fit( range.r2', points.y2,'fourier8');
fsfit.xbfit = fit( range.rb', points.xb,'fourier2');    % Lift
fsfit.ybfit = fit( range.rb', points.yb,'fourier2');
fsfit.x3fit = fit( range.r3', points.x3,'fourier5');    % 3rd Initial
fsfit.y3fit = fit( range.r3', points.y3,'fourier5');
fsfit.xcfit = fit( range.rc', points.xc,'fourier3');    % Lift
fsfit.ycfit = fit( range.rc', points.yc,'fourier3');


% Combine the FS fits for each section into a single column vector

x_disp_fsfit = [fsfit.x1fit(range.r1)
                fsfit.xafit(range.ra)
                fsfit.x2fit(range.r2)
                fsfit.xbfit(range.rb)
                fsfit.x3fit(range.r3)
                fsfit.xcfit(range.rc)];

y_disp_fsfit = [fsfit.y1fit(range.r1)
                fsfit.yafit(range.ra)
                fsfit.y2fit(range.r2)
                fsfit.ybfit(range.rb)
                fsfit.y3fit(range.r3)
                fsfit.ycfit(range.rc)];


% Refit the data set with a smoothing spline, to eliminate discontinuities
% between each fit

x_fit = fit(range_full, x_disp_fsfit, 'smoothingspline', 'SmoothingParam',.02);
y_fit = fit(range_full, y_disp_fsfit, 'smoothingspline', 'SmoothingParam',.02);


x_disp_smoothfit = x_fit(range_full)/yDPI;  % Used to convert to inches!
y_disp_smoothfit = y_fit(range_full)/yDPI;
%
% x_disp_smoothfit(end) = x_disp_smoothfit(1);
% y_disp_smoothfit(end) = y_disp_smoothfit(1);


% Plot Points & Fit Data

figure(1)
```

```
    plot(x_disp_smoothfit(span.s1),y_disp_smoothfit(span.s1),...
         x_disp_smoothfit(span.sa),y_disp_smoothfit(span.sa),...
         x_disp_smoothfit(span.s2),y_disp_smoothfit(span.s2),...
         x_disp_smoothfit(span.sb),y_disp_smoothfit(span.sb),...
         x_disp_smoothfit(span.s3),y_disp_smoothfit(span.s3),...
         x_disp_smoothfit(span.sc),y_disp_smoothfit(span.sc),'LineWidth',1); hold
         scatter(x_disp_points/yDPI, y_disp_points/yDPI,25,'.','LineWidth',2); hold
    grid
    axis([0 2 0 1])
    title('Input Points & Generated Parametric Fit')
    xlabel('(inches)')
    ylabel('(inches)')
    legend('1st Initial', 'Lift', '2nd Initial', 'Lift', '3rd Initial', 'Lift')


%% Generate Cam Profiles

theta = linspace(0,2*pi,length(range_full))';
thetad = theta/pi*180;


% Define Angular Displacement for Lifted Zones
angdisp.sa = theta(span.sa(end))-theta(span.sa(1));
angdisp.sb = theta(span.sb(end))-theta(span.sb(1));
angdisp.sc = theta(span.sc(end))-theta(span.sc(1));


% Harmonic Rise & Fall Function for z-axis cam
lift = @(x,angdisp,starting_point) (1-cos(2*pi*(x-starting_point)/angdisp))/2*Lh;

% Define Cam Profile
z_points = zeros(1,length(range_full));
z_points(span.s1)= 0;
z_points(span.sa)= lift(theta(span.sa),angdisp.sa,theta(span.sa(1)));
z_points(span.s2)= 0;
z_points(span.sb)= lift(theta(span.sb),angdisp.sb,theta(span.sb(1)));
z_points(span.s3)= 0;
z_points(span.sc)= lift(theta(span.sc),angdisp.sc,theta(span.sc(1)));

% Points are already smooth--used Harmonic Function. No need to interpolate
% or fit data.

z_disp_smooth = z_points';



% Generate Final Cam Profiles.

x_cam_profile = x_disp_smoothfit + Brx;
y_cam_profile = y_disp_smoothfit + Bry;
z_cam_profile = z_disp_smooth    + Brz;



% Draw Cams

figure(2)
subplot(1,3,1)
    polarplot(theta(span.s1),x_cam_profile(span.s1),...
              theta(span.sa),x_cam_profile(span.sa),...
```

```
            theta(span.s2),x_cam_profile(span.s2),...
            theta(span.sb),x_cam_profile(span.sb),...
            theta(span.s3),x_cam_profile(span.s3),...
            theta(span.sc),x_cam_profile(span.sc),'LineWidth',2)
    title('X Cam Profile')
subplot(1,3,2)
    polarplot(theta(span.s1),y_cam_profile(span.s1),...
            theta(span.sa),y_cam_profile(span.sa),...
            theta(span.s2),y_cam_profile(span.s2),...
            theta(span.sb),y_cam_profile(span.sb),...
            theta(span.s3),y_cam_profile(span.s3),...
            theta(span.sc),y_cam_profile(span.sc),'LineWidth',2)
    title('Y Cam Profile')
subplot(1,3,3)
    polarplot(theta(span.s1),z_cam_profile(span.s1),...
            theta(span.sa),z_cam_profile(span.sa),...
            theta(span.s2),z_cam_profile(span.s2),...
            theta(span.sb),z_cam_profile(span.sb),...
            theta(span.s3),z_cam_profile(span.s3),...
            theta(span.sc),z_cam_profile(span.sc),'LineWidth',2)
    title('Z Cam Profile')
    legend('1st Initial', 'Lift', '2nd Initial', 'Lift', '3rd Initial',
'Lift','Location','Best')



%% Analyze Cams

% Kinematic Coefficients / s,v,a,j

 x_s = x_disp_smoothfit;
 y_s = y_disp_smoothfit;
 z_s = z_disp_smooth;

x_k1 = diff(x_cam_profile)./diff(theta);
y_k1 = diff(y_cam_profile)./diff(theta);
z_k1 = diff(z_cam_profile)./diff(theta);

x_k2 = diff(x_k1)./diff(theta(1:length(theta)-1));
y_k2 = diff(y_k1)./diff(theta(1:length(theta)-1));
z_k2 = diff(z_k1)./diff(theta(1:length(theta)-1));

x_j = diff(x_k2)./diff(theta(1:length(theta)-2));
y_j = diff(y_k2)./diff(theta(1:length(theta)-2));
z_j = diff(z_k2)./diff(theta(1:length(theta)-2));

% Radius of Curvature and Pressure Angle

x_rho = ((x_s(1:end-2)+Brx).^2 + x_k1(1:end-1).^2).^(3/2) ./ ...
        ((x_s(1:end-2)+Brx).^2 + 2.*x_k1(1:end-1).^2 - x_k2.*(x_s(1:end-2)+Brx));
x_pa = atan(x_k1./(x_s(1:end-1)+Brx))*180/pi;

y_rho = ((y_s(1:end-2)+Bry).^2 + y_k1(1:end-1).^2).^(3/2) ./ ...
        ((y_s(1:end-2)+Bry).^2 + 2.*y_k1(1:end-1).^2 - y_k2.*(y_s(1:end-2)+Bry));
y_pa = atan(y_k1./(y_s(1:end-1)+Bry))*180/pi;

z_rho = ((z_s(1:end-2)+Brz).^2 + z_k1(1:end-1).^2).^(3/2) ./ ...
        ((z_s(1:end-2)+Brz).^2 + 2.*z_k1(1:end-1).^2 - z_k2.*(z_s(1:end-2)+Brz));
z_pa = atan(z_k1./(z_s(1:end-1)+Brz))*180/pi;
```

```
% Plot Data

figure(3)
set(gcf, 'Position',  [100, 00, 1500, 700])
subplot(2,4,1)
    plot(thetad,x_cam_profile-Brx,'LineWidth',1)
    title('X Cam Displacement vs. Theta')
    ylabel('Displacement (inches)')
    xlabel('\theta (deg)')
    axis([0 360 0 2])
    xticks([0 60 120 180 240 300 360])
    grid
subplot(2,4,2)
    plot(thetad(1:length(thetad)-1),x_k1,'LineWidth',1)
    title('X Cam: 1st Order KC')
    xlabel('\omega (deg/s)')
    axis([0 360 -4 2])
    xticks([0 60 120 180 240 300 360])
    grid
subplot(2,4,3)
    plot(thetad(1:length(thetad)-2),x_k2,'LineWidth',1)
    title('X Cam: 2nd Order KC')
    xlabel('\alpha (deg/s^2)')
    axis([0 360 -20 20])
    xticks([0 60 120 180 240 300 360])
    grid
subplot(2,4,4)
    plot(thetad(1:length(thetad)-3),x_j,'LineWidth',1)
    title('X Cam: Jerk')
    xlabel('d\alpha (deg/s^3)')
%     axis([0 360 -20 20])
    xticks([0 60 120 180 240 300 360])
    xlim([0 360])
    grid
subplot(2,4,6)
    plot(thetad(1:end-2),x_rho,'LineWidth',1)
    title('X Cam: Radius of Curvature')
    ylabel('R.O.C. (inches)')
    xlabel('\theta (deg)')
    axis([0 360 -20 20])
    xticks([0 60 120 180 240 300 360])
    grid
subplot(2,4,7)
    plot(thetad(1:end-1),x_pa,'LineWidth',1)
    title('X Cam: Pressure Angle')
    ylabel('Pressure Angle (deg)')
    xlabel('\theta (deg)')
    xlim([0 360])
%     axis([0 360 -3 3])
    xticks([0 60 120 180 240 300 360])
    grid

figure(4)
set(gcf, 'Position',  [100, 00, 1500, 700])
subplot(2,4,1)
    plot(thetad,y_cam_profile-Bry,'LineWidth',1)
    title('Y Cam Displacement vs. Theta')
    ylabel('Displacement (inches)')
    xlabel('\theta (deg)')
```

```
    axis([0 360 0 1])
    grid
subplot(2,4,2)
    plot(thetad(1:length(thetad)-1),y_k1,'LineWidth',1)
    title('Y Cam: 1st Order KC')
    xlabel('\omega (deg/s)')
    axis([0 360 -4 2])
    xticks([0 60 120 180 240 300 360])
    grid
subplot(2,4,3)
    plot(thetad(1:length(thetad)-2),y_k2,'LineWidth',1)
    title('Y Cam: 2nd Order KC')
    xlabel('\alpha (deg/s^2)')
    axis([0 360 -20 20])
    xticks([0 60 120 180 240 300 360])
    grid
subplot(2,4,4)
    plot(thetad(1:length(thetad)-3),y_j,'LineWidth',1)
    title('Y Cam: Jerk')
    xlabel('d\alpha (deg/s^3)')
%     axis([0 360 -20 20])
    xlim([0 360])
    xticks([0 60 120 180 240 300 360])
    grid
subplot(2,4,6)
    plot(thetad(1:end-2),y_rho,'LineWidth',1)
    title('Y Cam: Radius of Curvature')
    ylabel('R.O.C. (inches)')
    xlabel('\theta (deg)')
    axis([0 360 -20 20])
    xticks([0 60 120 180 240 300 360])
    grid
subplot(2,4,7)
    plot(thetad(1:end-1),y_pa,'LineWidth',1)
    title('Y Cam: Pressure Angle')
    ylabel('Pressure Angle (deg)')
    xlabel('\theta (deg)')
%     axis([0 360 -3 3])
    xlim([0 360])
    xticks([0 60 120 180 240 300 360])
    grid



figure(5)
set(gcf, 'Position',  [100, 00, 1500, 700])
subplot(2,4,1)
    plot(thetad,z_cam_profile-Brz,'LineWidth',1)
    title('Z Cam Displacement vs. Theta')
    ylabel('Displacement (inches)')
    xlabel('\theta (deg)')
    axis([0 360 -.4 .4])
    xticks([0 60 120 180 240 300 360])
    grid
subplot(2,4,2)
    plot(thetad(1:length(thetad)-1),z_k1,'LineWidth',1)
    title('Z Cam: 1st Order KC')
    xlabel('\omega (deg/s)')
    axis([0 360 -3 3])
    xticks([0 60 120 180 240 300 360])
    grid
```

```
subplot(2,4,3)
    plot(thetad(1:length(thetad)-2),z_k2,'LineWidth',1)
    title('Z Cam: 2nd Order KC')
    xlabel('\alpha (deg/s^2)')
    axis([0 360 -20 20])
    xticks([0 60 120 180 240 300 360])
    grid
subplot(2,4,4)
    plot(thetad(1:length(thetad)-3),z_j,'LineWidth',1)
    title('Z Cam: Jerk')
    xlabel('d\alpha (deg/s^3)')
    axis([0 360 -20 20])
    xticks([0 60 120 180 240 300 360])
%     xlim([0 360])
    grid
subplot(2,4,6)
    plot(thetad(1:end-2),z_rho,'LineWidth',1)
    title('Z Cam: Radius of Curvature')
    ylabel('R.O.C. (inches)')
    xlabel('\theta (deg)')
    axis([0 360 -10 10])
    xticks([0 60 120 180 240 300 360])
    grid
subplot(2,4,7)
    plot(thetad(1:end-1),z_pa,'LineWidth',1)
    title('Z Cam: Pressure Angle')
    ylabel('Pressure Angle (deg)')
    xlabel('\theta (deg)')
%     axis([0 360 -3 3])
    xlim([0 360])
    xticks([0 60 120 180 240 300 360])
    grid



% X-Y (Cartesian) Cam Profile for Solid Model Generation (Hector)
% Note: 3rd columns in the below matricies are due to SolidWorks coordinate
% file loading requirements. (Need X Y Z)

x_cam_cart(:,1) = x_cam_profile.*cos(theta);
x_cam_cart(:,2) = x_cam_profile.*sin(theta);
x_cam_cart(:,3) = .75;

y_cam_cart(:,1) = y_cam_profile.*cos(theta);
y_cam_cart(:,2) = y_cam_profile.*sin(theta);
y_cam_cart(:,3) = 1.5;

z_cam_cart(:,1) = z_cam_profile.*cos(theta);
z_cam_cart(:,2) = z_cam_profile.*sin(theta);
z_cam_cart(:,3) = 0;

% Export Data to .txt files loadable by CAD software   (Hector)

save('x_cam_cart.txt', 'x_cam_cart','-ascii', '-tabs');
save('y_cam_cart.txt', 'y_cam_cart','-ascii', '-tabs');
save('z_cam_cart.txt', 'z_cam_cart','-ascii', '-tabs');
```

Appendix C. Alternate Point Generation Code

```
%% (Coded by Miranda)
% save signature as .png file with white text on black background for ease
% of signature outlining
%%
% Read image and display it.
I = imread('bwsig.png');
imshow(I)
%%
% Convert image to binary black&white
BW = im2bw(I);
imshow(BW)
%%
% following finds row/col coord of border pixel (AKA where the signature
% starts) this will be beginning of trace function
dim = size(BW)
col = round(dim(2)/2)-90;
row = min(find(BW(:,col)))
%%
% |bwtraceboundary| will trace all pixels starting at the point found above
% Call |bwtraceboundary| to trace the boundary from starting pt whose
% coords are defined above (these must be input to the function, must also
% input direction of first step (any cardinal direction)
boundary = bwtraceboundary(BW,[row, col],'E');
%%
% Show the original image first, then display the trace over this image
% (can check any points where image is not tracing properly)
imshow(I)
hold on;
plot(boundary(:,2),boundary(:,1),'g','LineWidth',3);
%%
% |bwboundaries| function traces all outlines.
% To ensure that|bwboundaries| only traces the outside use |imfill| to fill the area
% inside each letter. |bwboundaries| returns a cell array, where each cell
% contains the row/column coordinates for an object in the image.
BW_filled = imfill(BW,'holes');
boundaries = bwboundaries(BW_filled);
%%
% Plot the borders on the original grayscale image using
% the coordinates returned by |bwboundaries| .
for k=1:10
   b = boundaries{k};
   plot(b(:,2),b(:,1),'g','LineWidth',3);
end
```

Appendix D: “MCP” Cam Generation Results

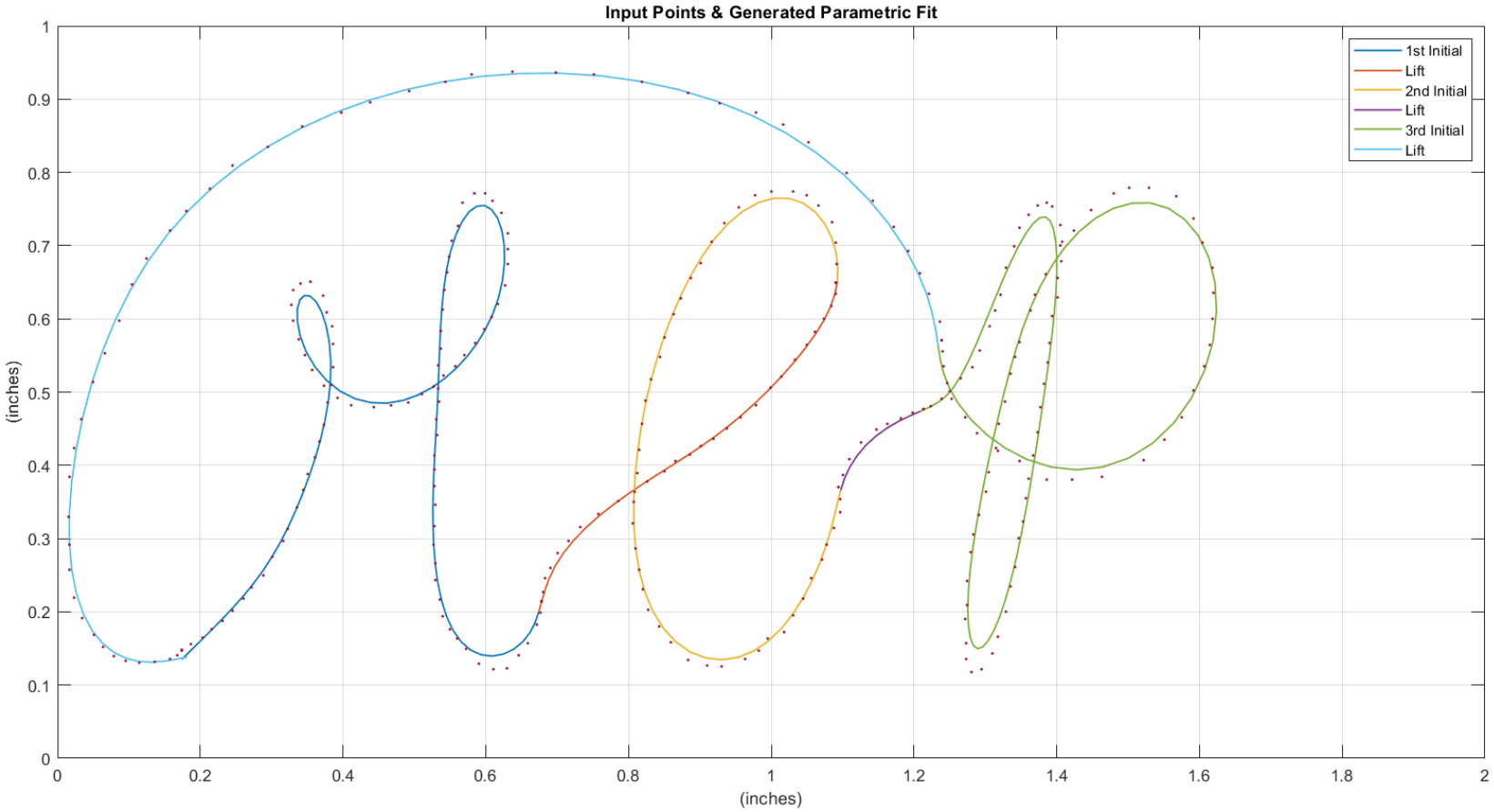

Figure 6: MCP Input Points & Fit

MCP Cam Profiles

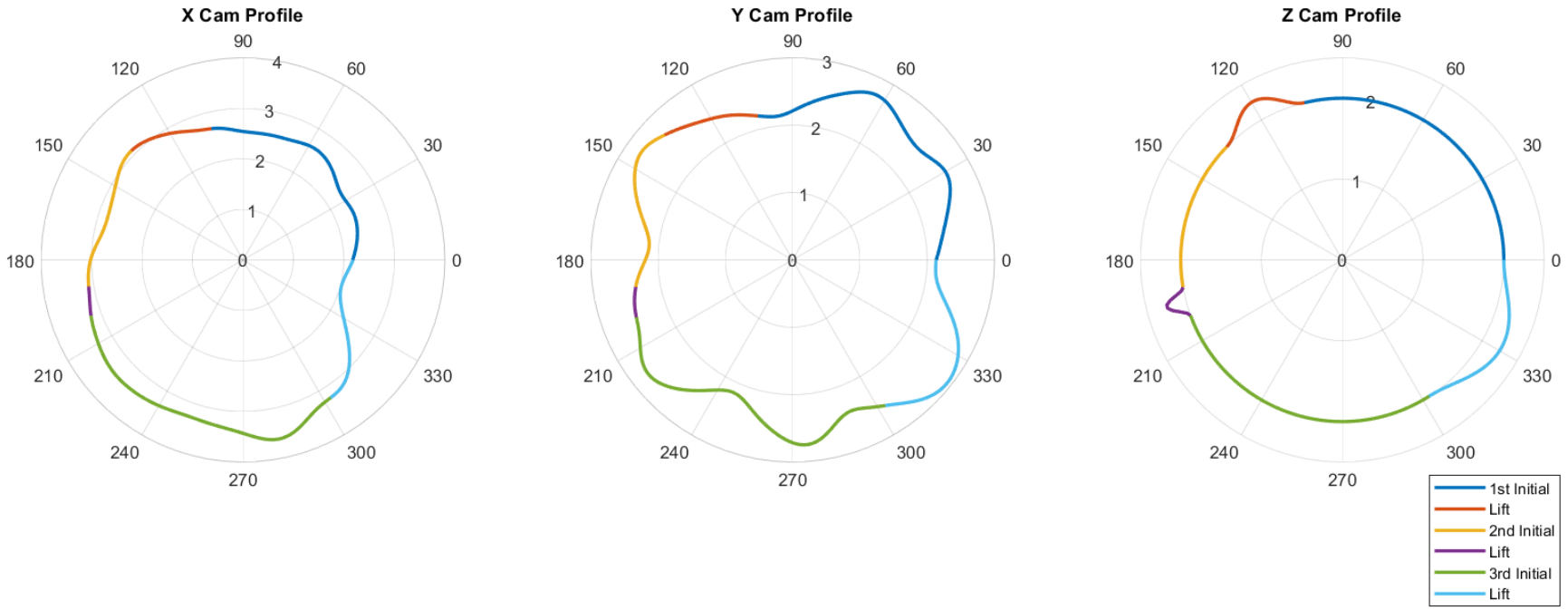

Figure 7: MCP Cam Profiles

MCP: X Cam Analysis

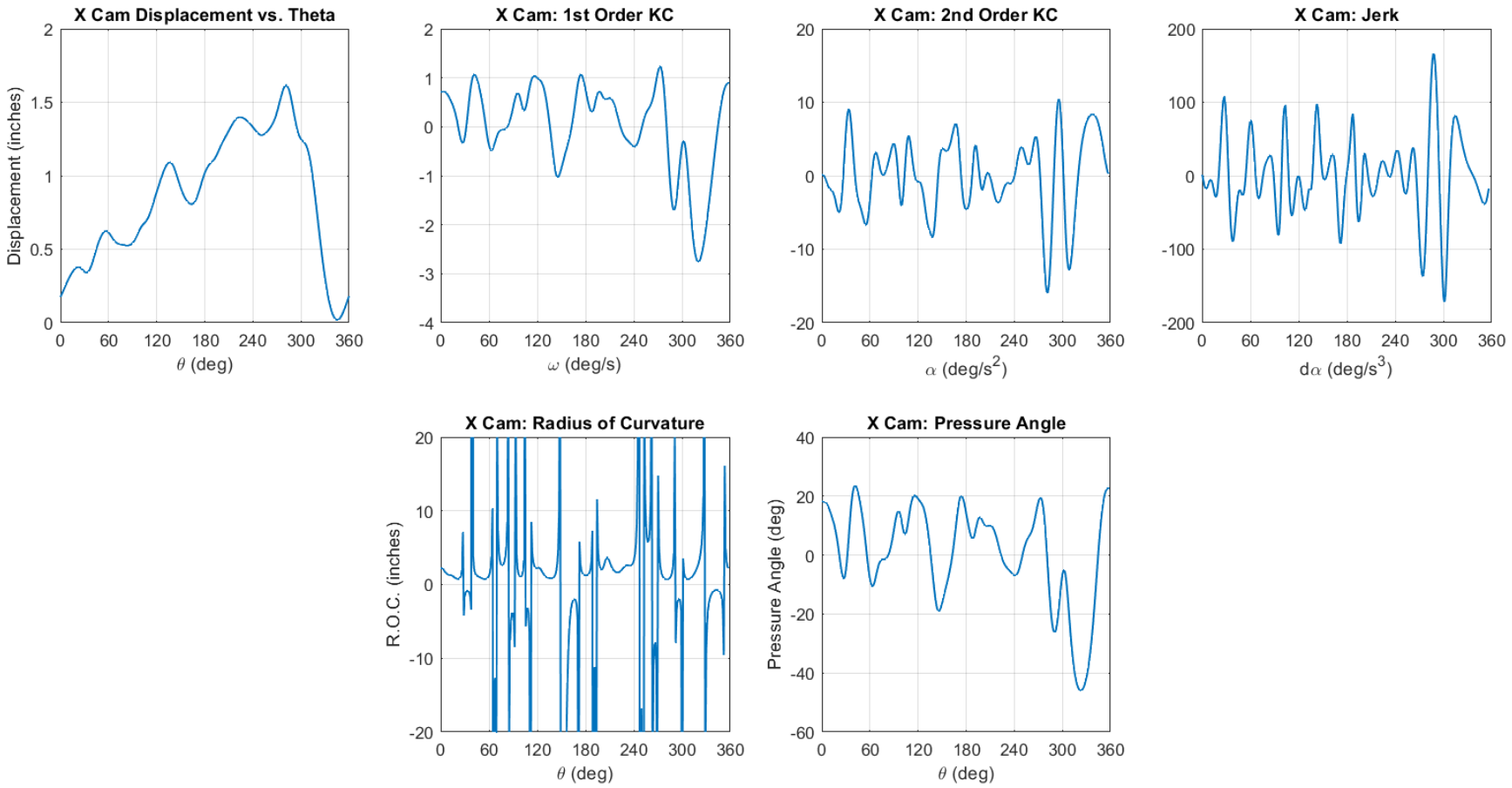

Figure 8: MCP X Cam Analysis

MCP: Y Cam Analysis

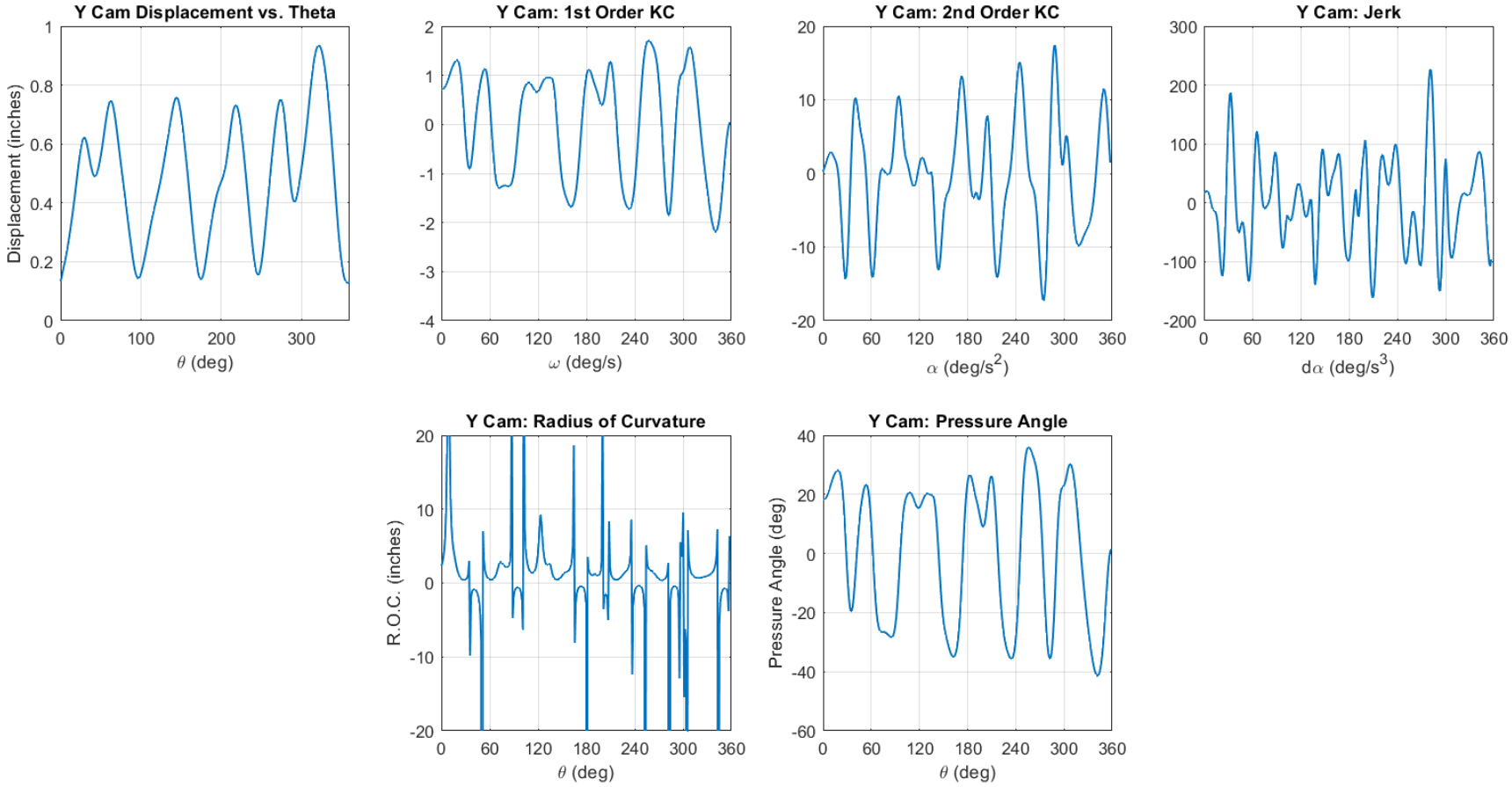

Figure 9: MCP Y Cam Analysis

MCP: Z Cam Analysis

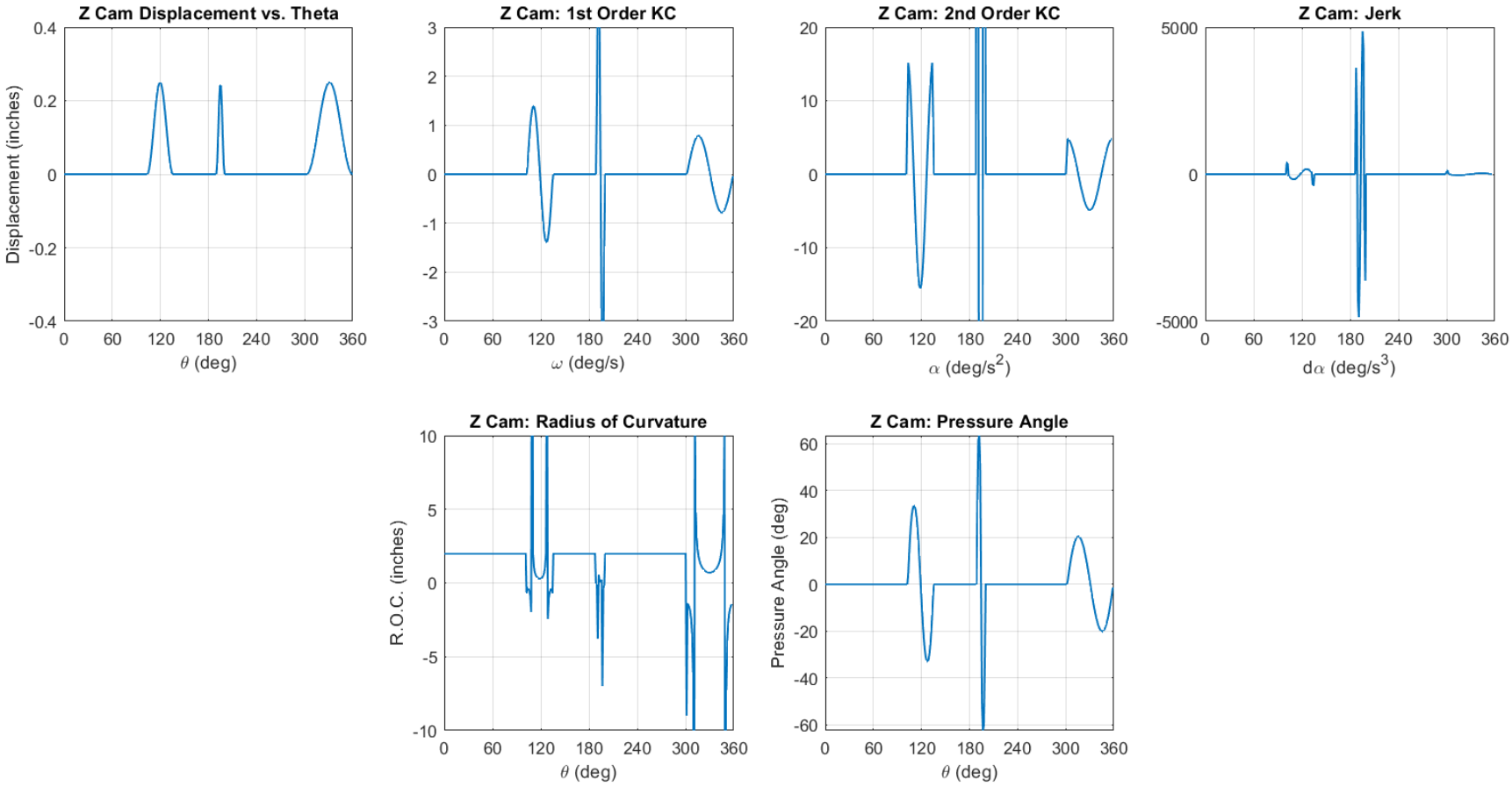

Figure 10: MCP Z Cam Analysis

Appendix E: “WLA” Cam Generation Results

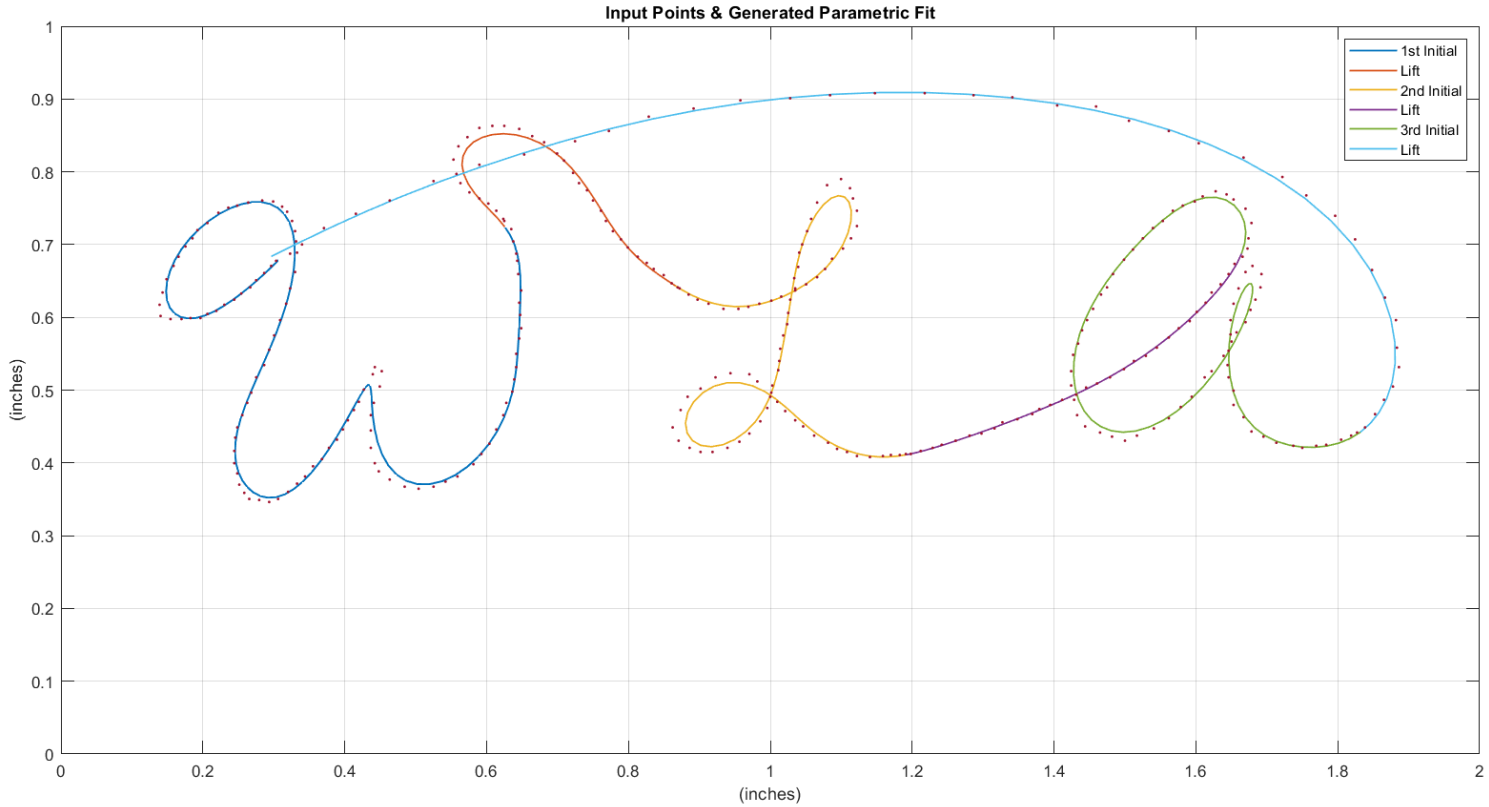

Figure 11: WLA Input Points & Fit

WLA Cam Profiles

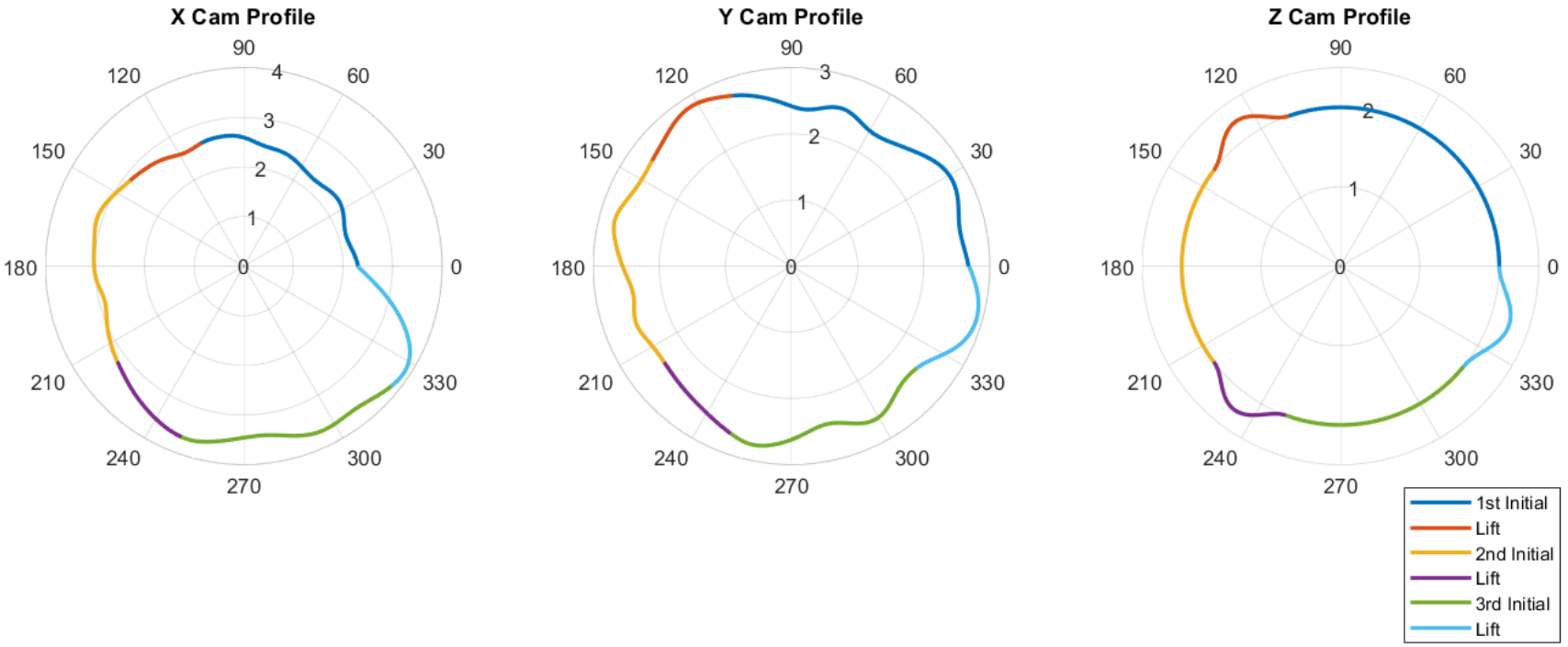

Figure 12: WLA Cam Profiles

WLA: X Cam Analysis

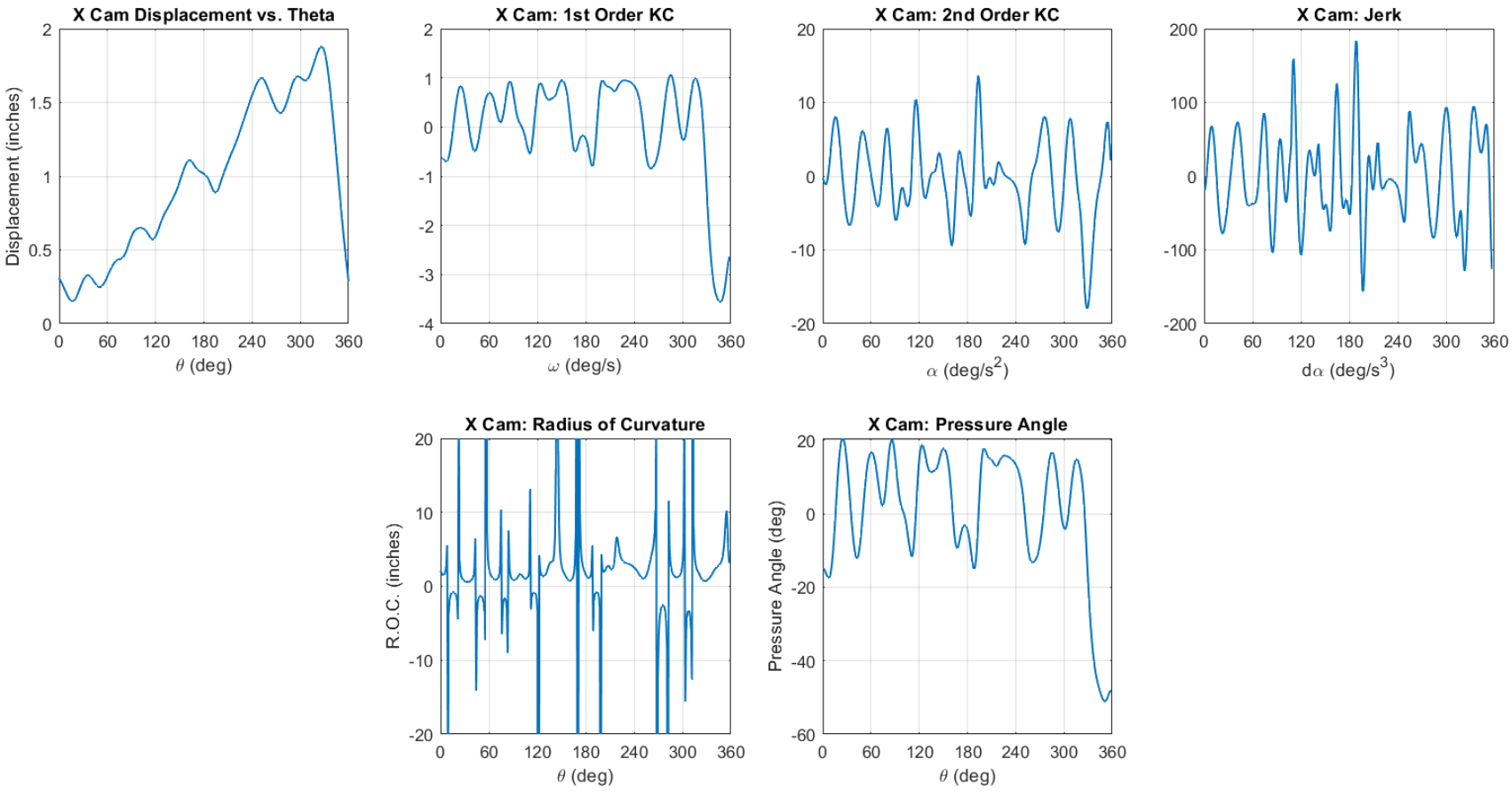

Figure 13: WLA Input Points & Fit

WLA: Y Cam Analysis

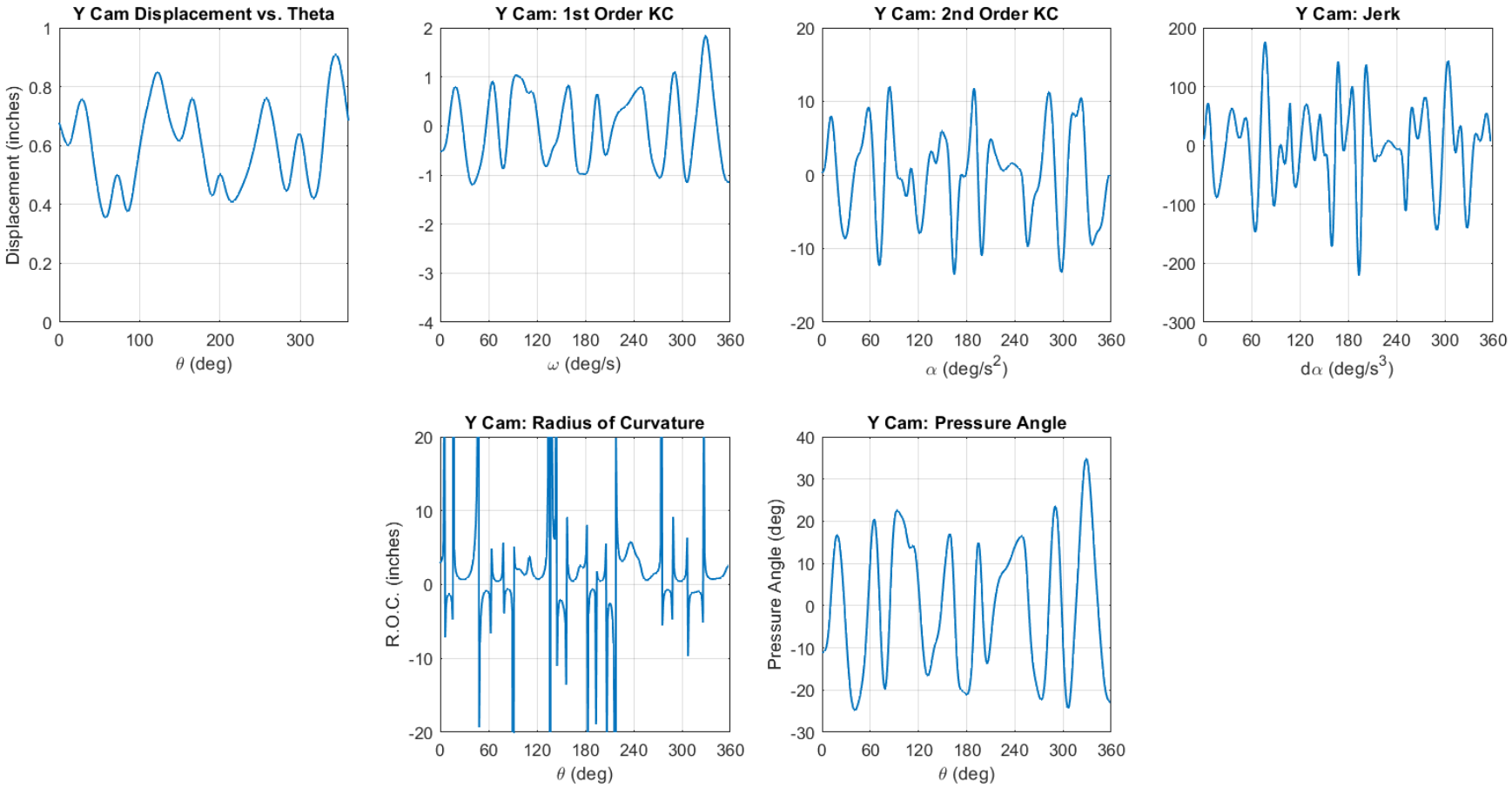

Figure 14: WLA Y Cam Analysis

WLA: Z Cam Analysis

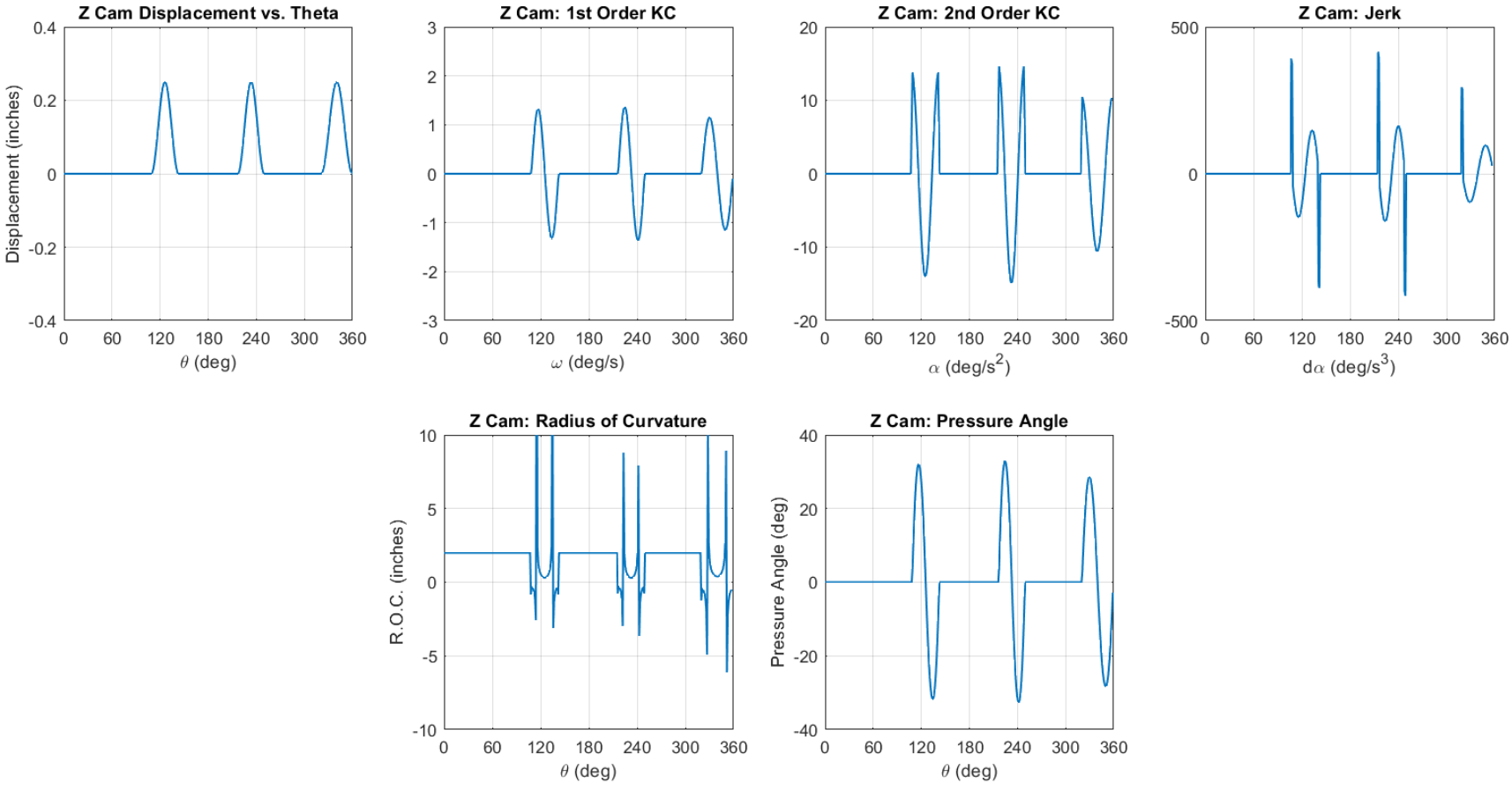

Figure 15: WLA Z Cam Analysis

Appendix F: “HAR” Cam Generation Results

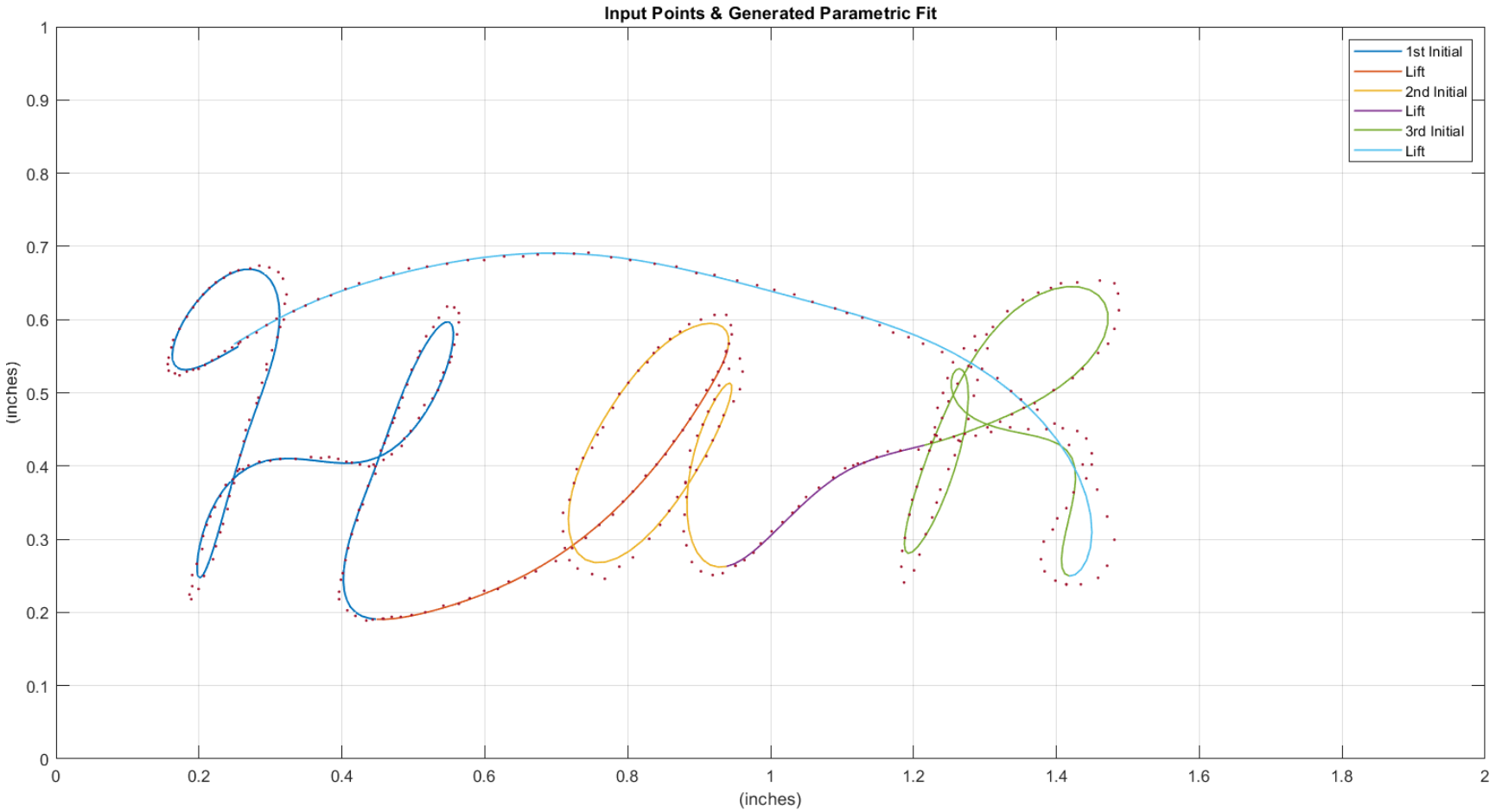

Figure 16: HAR Input Points & Fit

HAR Cam Profiles

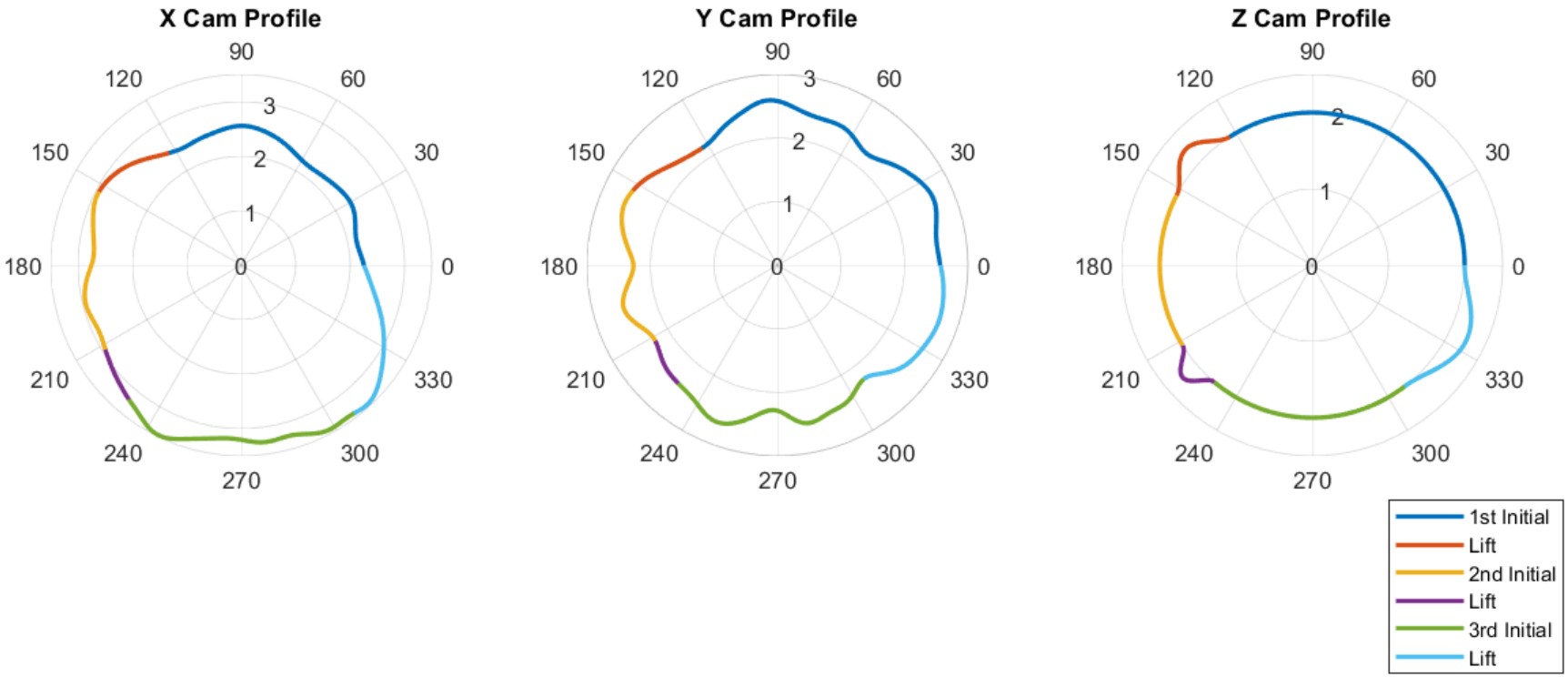

Figure 17: HAR Cam Profiles

HAR: X Cam Analysis

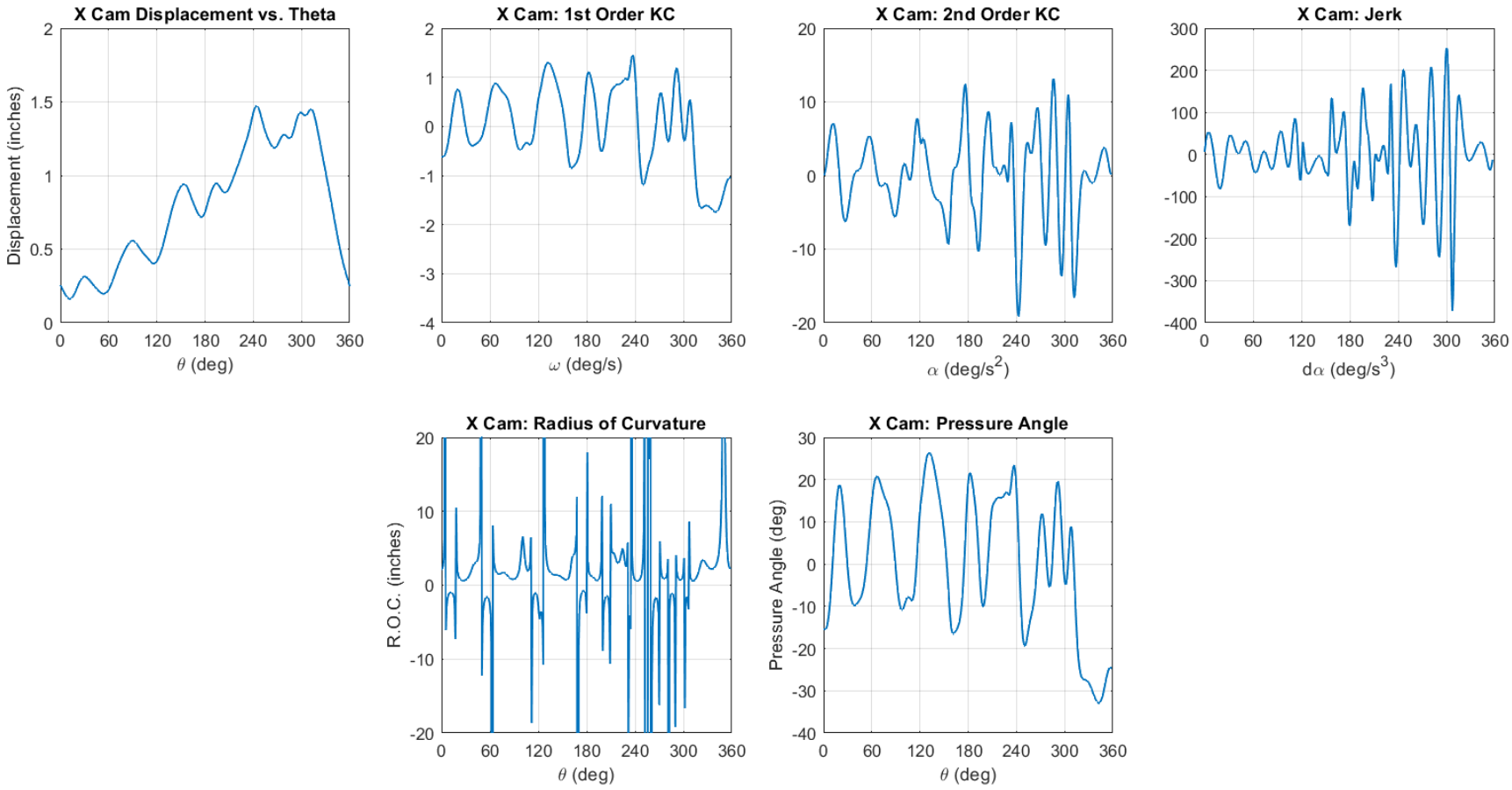

Figure 18: X Cam Analysis

HAR: Y Cam Analysis

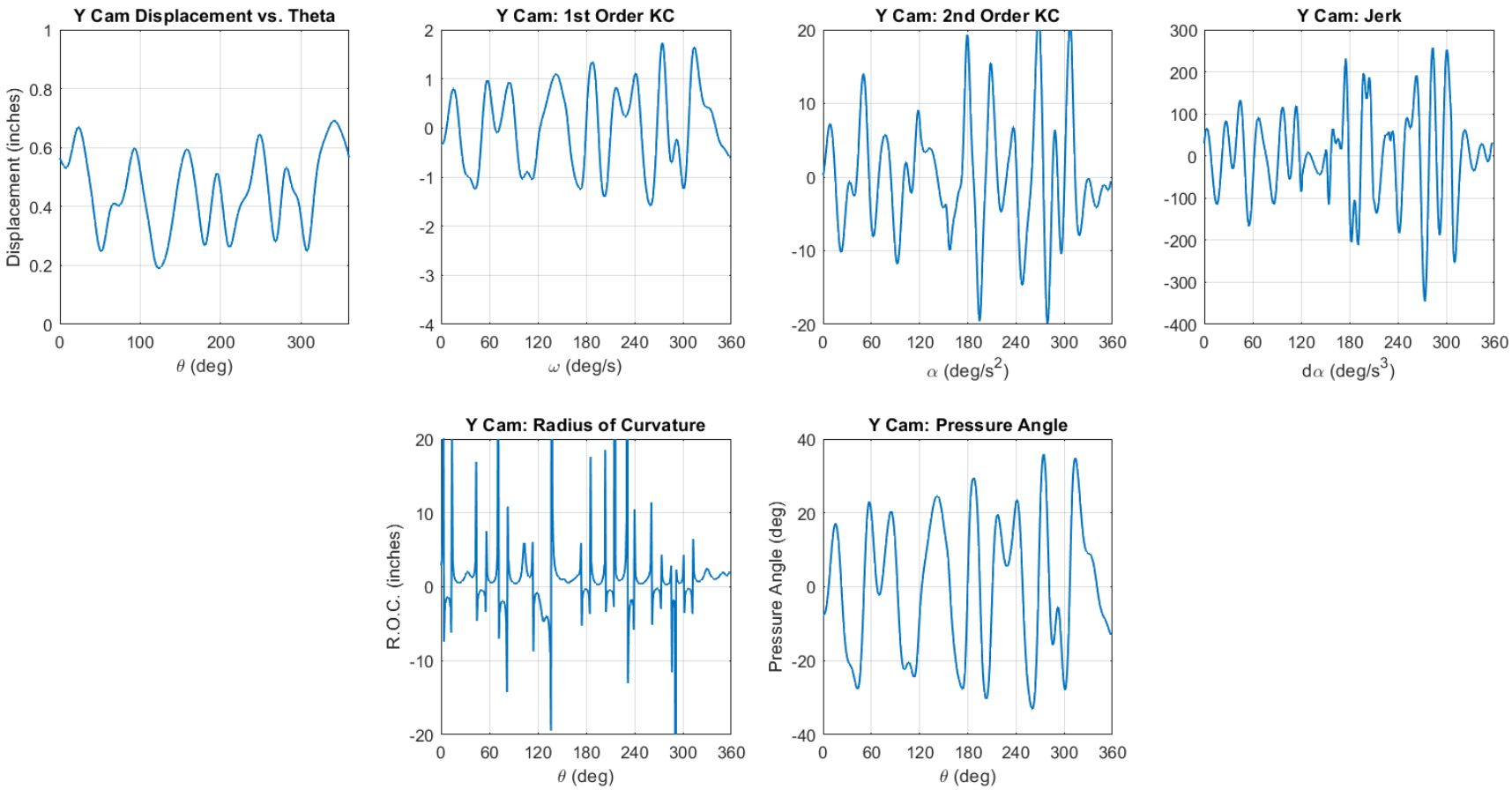

Figure 19: Y Cam Analysis

HAR: Z Cam Analysis

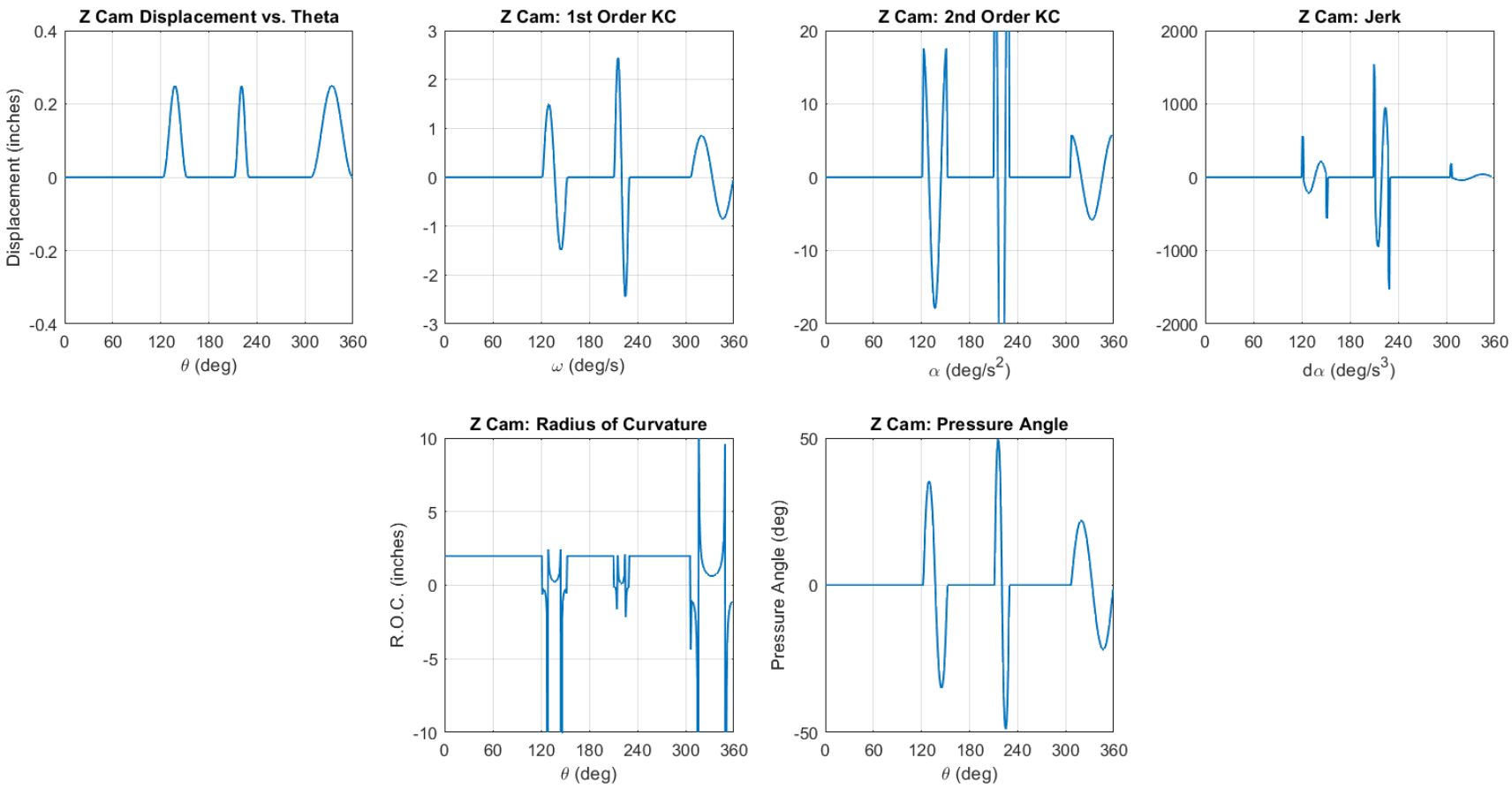

Figure 20: HAR Z Cam Analysis

Appendix G: Mechanism Sketches & Solid Model

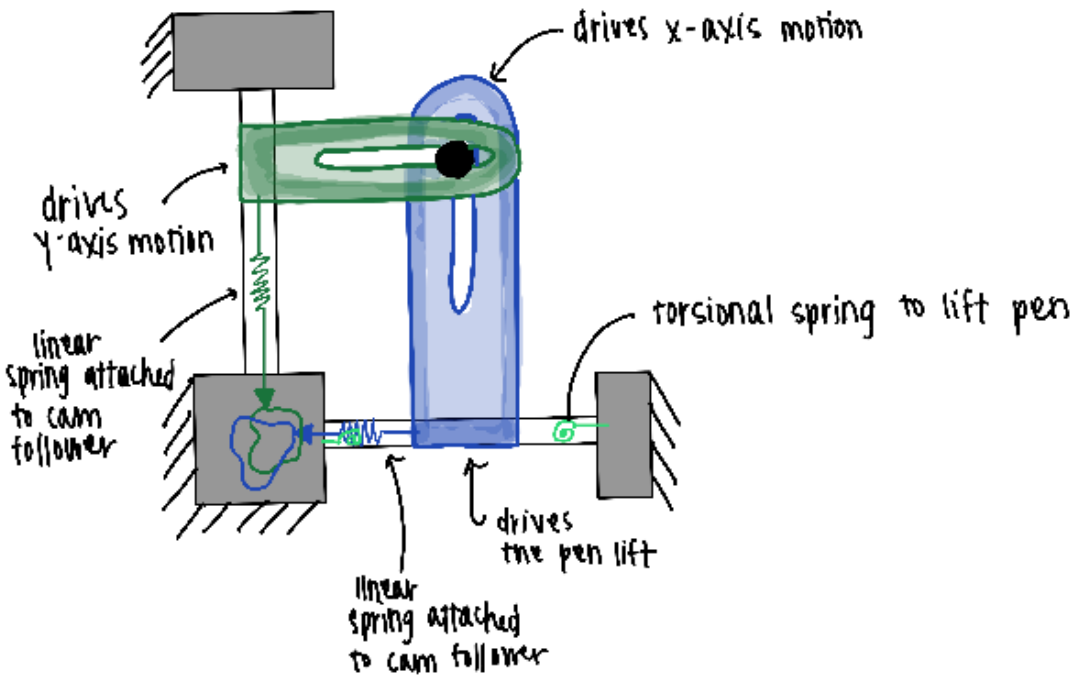

Figure 21: Early Mechanism Sketch

Credit: Miranda Pepper

Based on mechanism in Meriam's Dynamics Textbook (Problem 2/129)

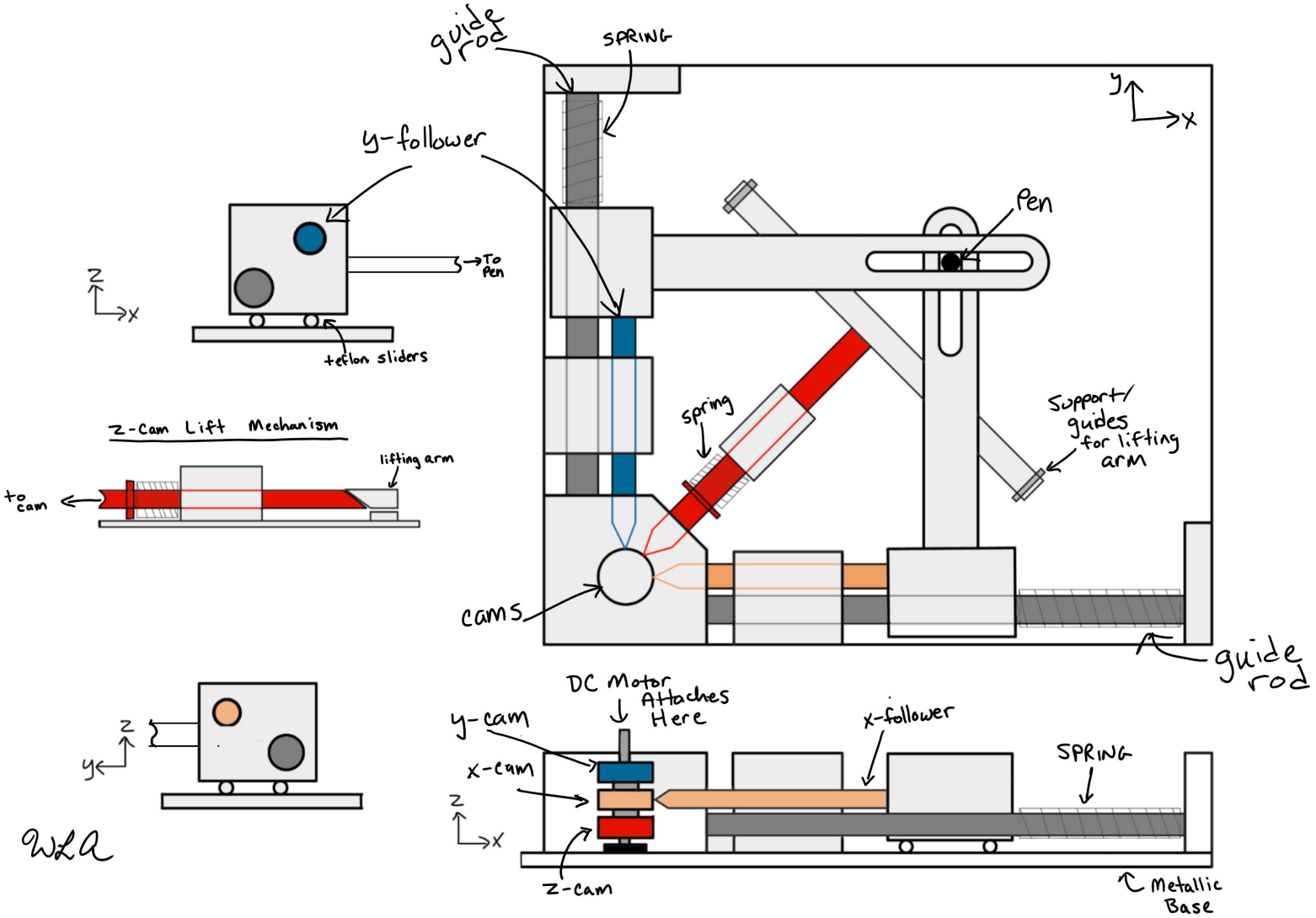

Figure 22: Mechanism Sketch (by William Ard)

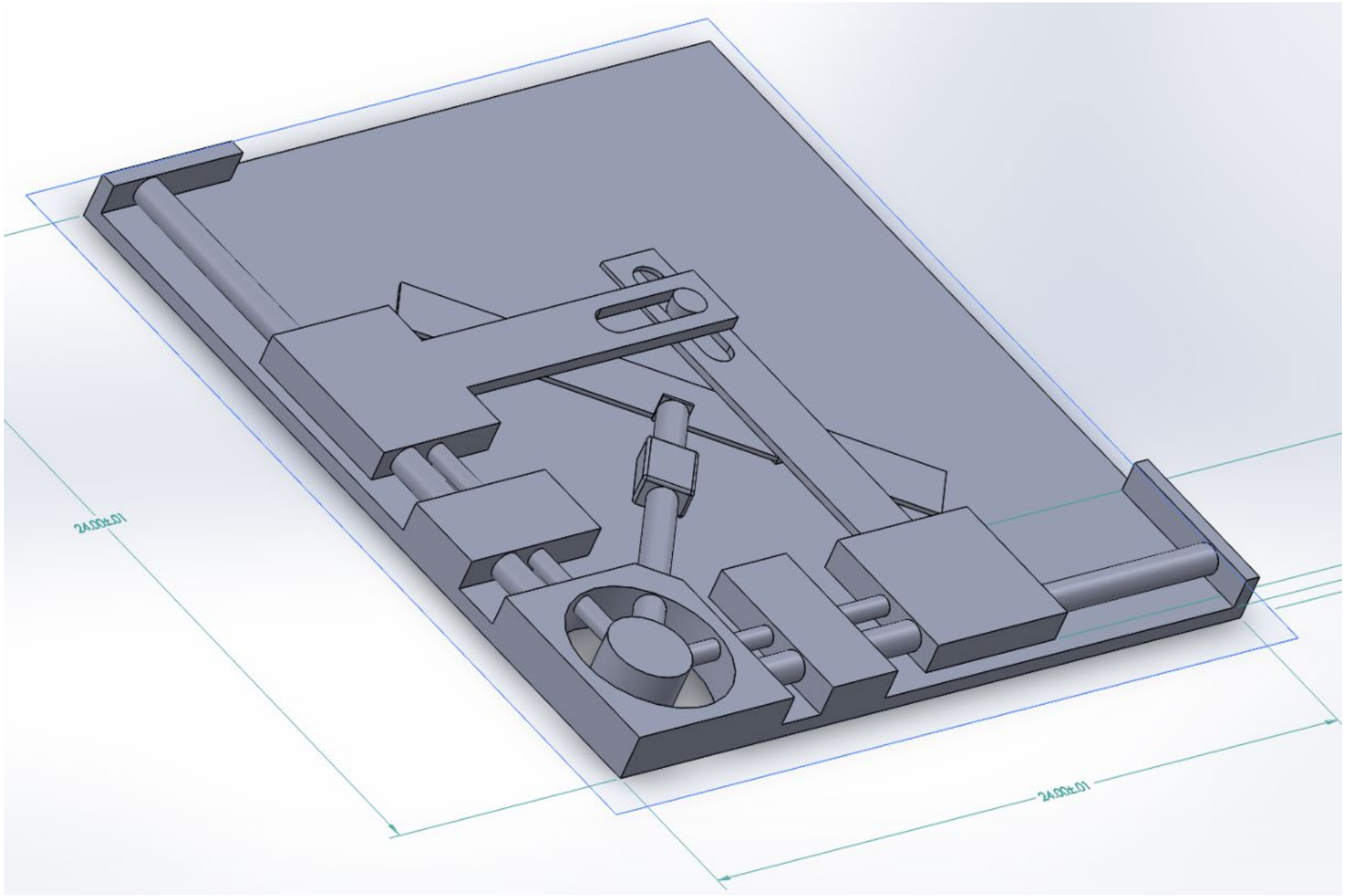

Figure 23: Mechanism Solid Model (Credit: Hector Arredondo)

Appendix H: References

[1] J. L. Meriam, L. G. Kraige and J. N. Bolton, Engineering Mechanics, Vol.2: Dynamics, Wiley & Sons, 2015.

[2] R. L. Norton, Cam Design and Manufacturing Handbook, New York: Industrial Press, 2009.