MonteCarlo Simulation

"**Monte Carlo methods**, or **Monte Carlo experiments**, are a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results. The underlying concept is to use randomness to solve problems that might be deterministic in principle. They are often used in physical and mathematical problems and are most useful when it is difficult or impossible to use other approaches."

https://en.wikipedia.org/wiki/Monte_Carlo_method

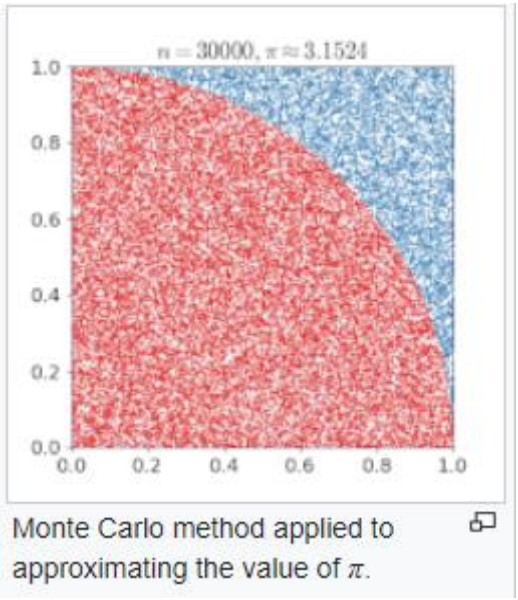


Figure1. Throwing random darts on the square and counting the number in the red area

The basic idea is to create random variables to the inputs of your experiment, in order to simulate your experiment, and get the variations respect to the theoretical values. The most common way to define random variables is to use the normal distribution.

```
close all
clear all
clc

tiledlayout(2,2) % Requires R2019b or later

nexttile;
x = randn(100000,1); % Centered and sigma = 1
nbins = 300;
h = histogram(x,nbins);

% ======================
nexttile;
x = 3 * randn(100000,1); % centered and sigma = 3
nbins = 300;
h = histogram(x,nbins);

% ======================
nexttile;
x = 10 + randn(100000,1); % sigma = 1 displaced by 10
nbins = 300;
h = histogram(x,nbins);

% ======================
```

```
nexttile;
x = 10 + 3 * randn(100000,1); % sigma = 3 displaced by 10
nbins = 300;
h = histogram(x,nbins);
```

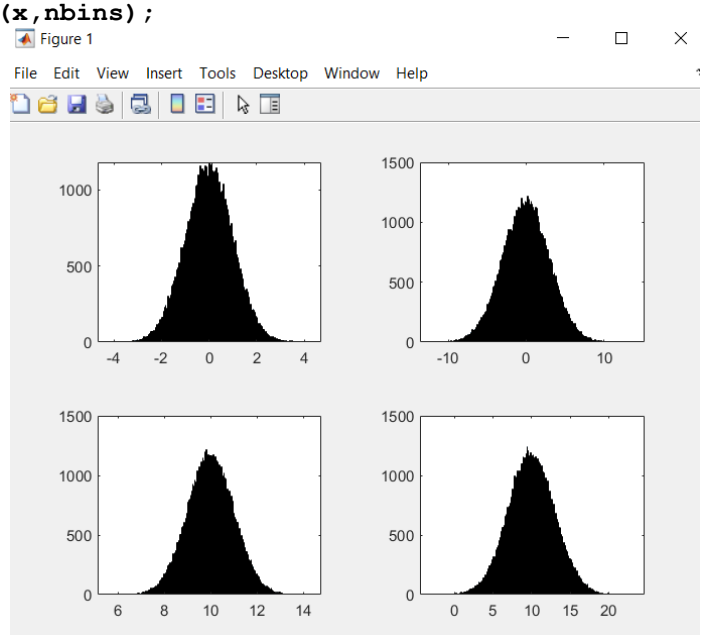


Figure 2- Normal Probabilities with different mean and sigma

Application to the analysis of Laser camera Triangulation

We like to analyze the errors in reconstruction in function of the errors in the subpixel precision of the laser extraction, and the error in the physical points of the cone. We assume that the H matrix (homography matrix) obtained from the calibration is correct.

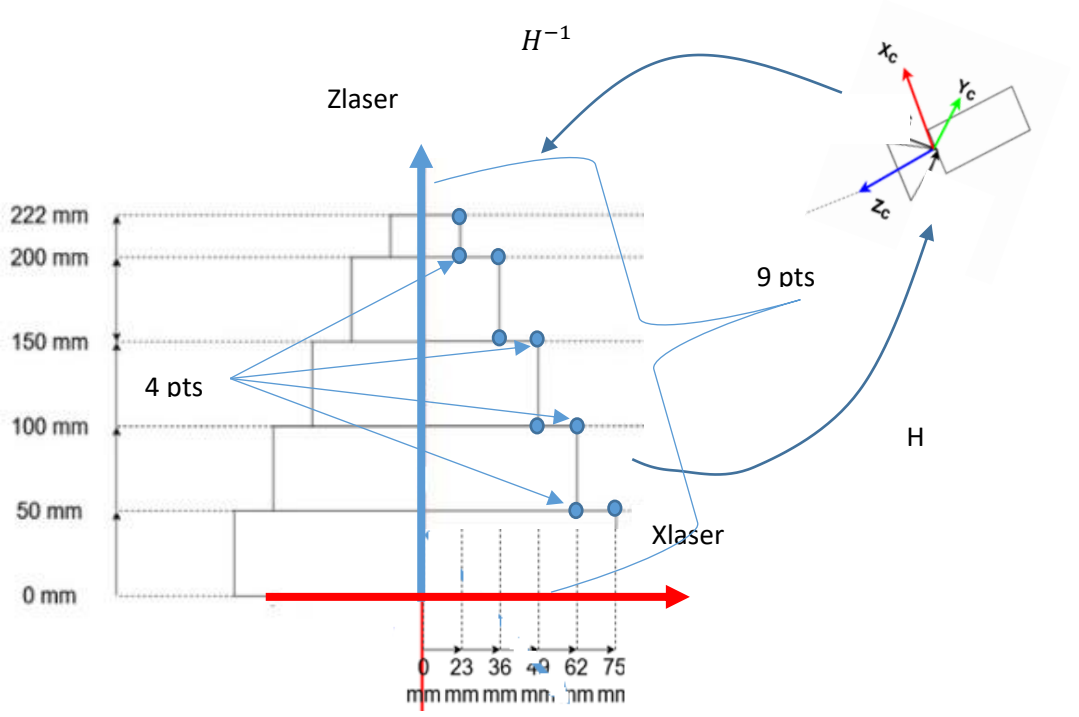


Figure 3. Point correspondence for MonteCarlo analysis

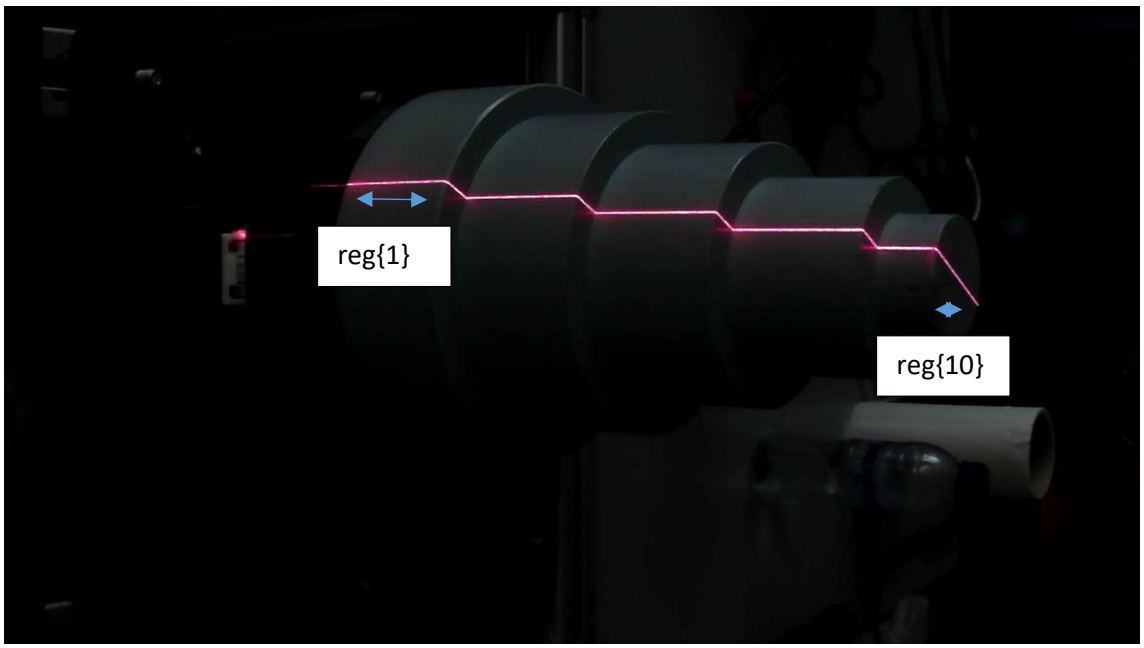
In order to do the Monte-Carlo Analysis, we project the set points in the cone (theoretical points plus a random noise) to the image. To the points in the image, we add a random noise. We find straight lines and intersections, as previously done, and re-project the image points to the laser plane. We calculate the errors between the theoretical points in the cone and the re-projected ones.

Hint:

Define the same set of regions as in the previous example:

```
reg{1} = [600, 741];
reg{2} = [750, 780];
reg{3} = [788, 959];
reg{4} = [970, 995];
reg{5} = [1009, 1200];
reg{6} = [1205, 1232];
reg{7} = [1235, 1445];
reg{8} = [1456, 1473];
reg{9} = [1478, 1568];
reg{10} = [1581, 1627];
```

```
PtsCono = [75 62 62 49 49 36 36 23 23; ...
           50 50 100 100 150 150 200 200 222];
```



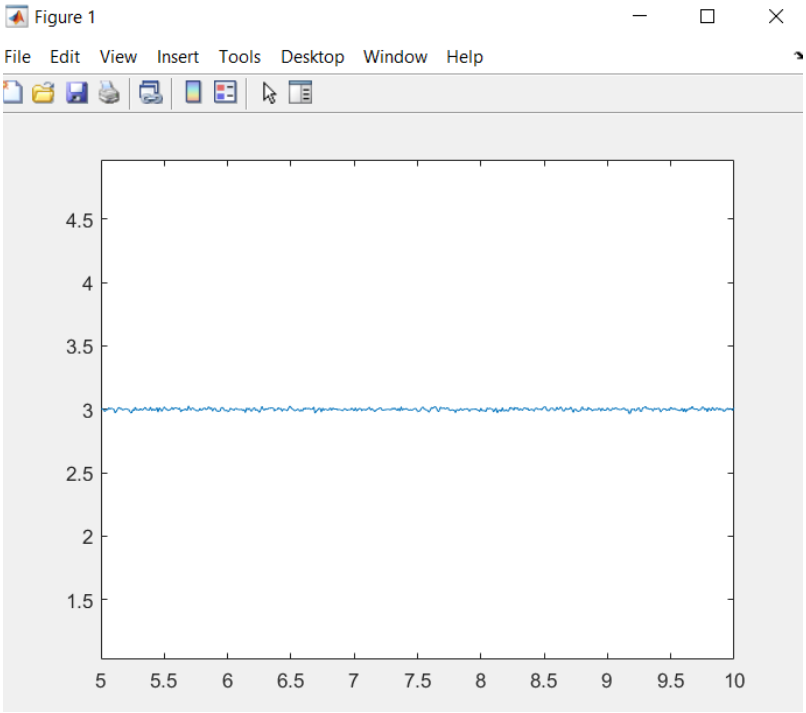
We calculate for each region the theoretical points in the image vertical line, calculating which theoretical point of the cone is the corresponding one. Add noise to the theoretical point in the cone, calculated the corresponding image, and add point to this image. Calculate the straight lines and intersection of the lines in order to calculate the modified homography. Re-project the points in all segments with the new homography and calculate the errors with the theoretical points in the cone. Calculate the statistics repeating the calculation n = 100 times. Calculate the statistics for the following sigma values (laser and images).

```
% Variances of added normal noises
sigmaPw = [0, 0.001, 0.005, 0.01, 0.05, 0.1];
sigmaPixel = [0, 0.01, 0.05, 0.1];
```

Note: The profile of the cone creates errors in the image lines. These errors can be model in three different ways:

a) Add the error in the theoretical point of the cone, and reproject the point into the camera. It gives a different profile.
b) Compute a cone spline, modifying the theoretical points obtained from the image profiles. Intersect the corresponding image profile in the cone with the spline. We show how to do it with one example:

```
clear all; clc; close all;

% intersect a spline with a horizontal line
 x = 5:0.01:10;
 y = 3 + 0.01 * randn(size(x));

 figure
 plot(x, y); axis equal
```

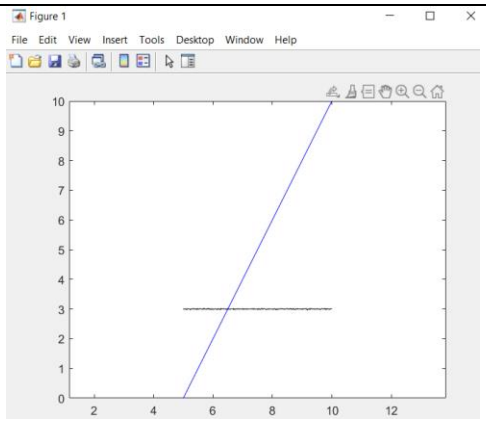


Create a 2d Spline passing though the points.

```
fhcsc = cscvn([x;y]);
 t = linspace(fhcsc.breaks(1),fhcsc.breaks(end),length(x) * 10);
 v = ppual(fhcsc,t);

 hold on;
 plot(v(1,:), v(2,:), 'k');
```

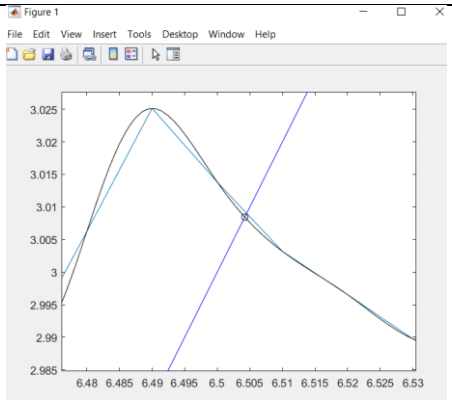
Create a line

```
% Create a line
 p1 = [6, 2];
 p2 = [7, 4];

 Recta = cross([p1, 1], [p2, 1]);

 plot(x, -(Recta(1) * x + Recta(3)) / Recta(2), 'b');
```



Intersect the line with a dicotonomic search:

```
[pintersect, n_steps] = intersectSpline(fhcsc, Recta, t(1), t(end),
'plot');
 plot(pintersect(1), pintersect(2), 'ok');
```

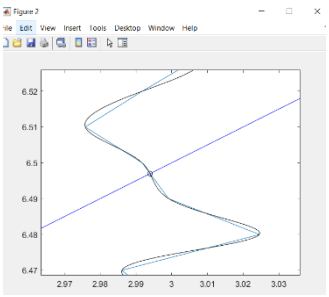


The same works for vertical lines:

```
% Try with a vertical line
 y = 5:0.01:10;
 x = 3 + 0.01 * randn(size(x));
 fvcsc = cscvn([x;y]);
 t = linspace(fvcsc.breaks(1),fvcsc.breaks(end),length(x) * 10);
 v = ppual(fvcsc,t);

  % Create a line
```

```
p1 = [2, 6];
p2 = [4, 7];

Recta = cross([p1, 1], [p2, 1]);

figure
plot(x, y); axis equal
hold on;
plot(v(1,:), v(2,:), 'k');
plot(-(Recta(2) * y + Recta(3)) / Recta(1), y, 'b');
[pintersect, n_steps] = intersectSpline(fvcsc, Recta, t(1), t(end),
'plot');
plot(pintersect(1), pintersect(2), 'ok');
```

```
%% Calculates the intersection by dicotonomical search
function [pt, nsteps] = intersectSpline(ft, Recta, t1, tf, varargin)
 % Calculate the signs at the beg and end
 pt1 = ppual(ft,t1);
 ptf = ppual(ft,tf);

 sign1 = sign(Recta(1) * pt1(1) + Recta(2) * pt1(2) + Recta(3));
 sign2 = sign(Recta(1) * ptf(1) + Recta(2) * ptf(2) + Recta(3));

 interval = [t1, tf];
 nsteps = 0;

 % Do dicotonomically until the sign is cero or the interval es small
 while(norm(interval(2) - interval(1)) > 1.0e-5)
   t = (interval(1) + interval(2)) / 2;
   nsteps = nsteps + 1;

   pt = ppual(ft,t);

   if (length(varargin) > 0 && strcmp(varargin{1}, 'plot') == true)
     plot(pt(1), pt(2), 'og');
     norm(interval(2) - interval(1))
   end

   signInt = sign(Recta(1) * pt(1) + Recta(2) * pt(2) + Recta(3));

   if (signInt == 0)
     break;
   end

   if (signInt == sign1)
     interval = [t, interval(2)];
   else
     interval = [interval(1), t];
   end
 end
end
```



Steps

1. Calculate for each region and for each vertical line (image) of the regions the point in the cone, and the point in the vertical line of the image.
2. Add noise to the points in the cone perpendicular to the direction of the line in the cone, in order to simulate the rugosity.
3. Use one of the 3 methods presented previously (either project to the image directy, finding a different vertical line in the image), or project the vertical line to the image and intersect with the calculated spline of the cone.
4. Add random noise in the image.
5. Calculate the new homography, and proect the modified image points into the cone.
6. Calculate the errors between the projected points and the modified points in the cone.
7. Do a table (Excel or document) in Matlab, with the Matlab program.
8. Draw conclusions.