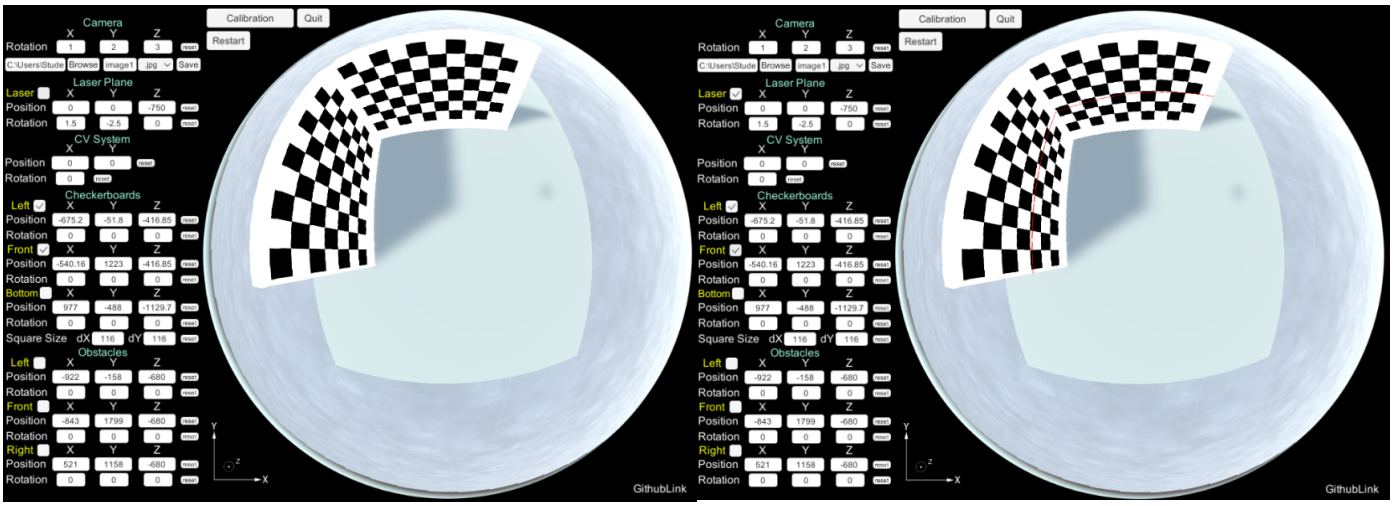
GETTING STARTED ADVANCE

This tutorial is based on our paper “Model of an Omnidirectional Vision System with Laser Illumination and its Calibration Based on Simulation” [1]. So firstly, it’s highly recommended to read this paper. The value of this tutorial is that you can understand the importance of the Vision System calibration, plus you can implement it to the real systems.
We will set up configuration of our vision system just in order to compare our experiment results with the following ones:

- Camera is rotated around X-axis by 1°, around Y-axis by 2° and around Z-axis by 3°;
- Laser plane is rotated around X-axis by 1.5° and around Y-axis by -2.5°;
- Distance between Camera and Laser plane is 750 mm;

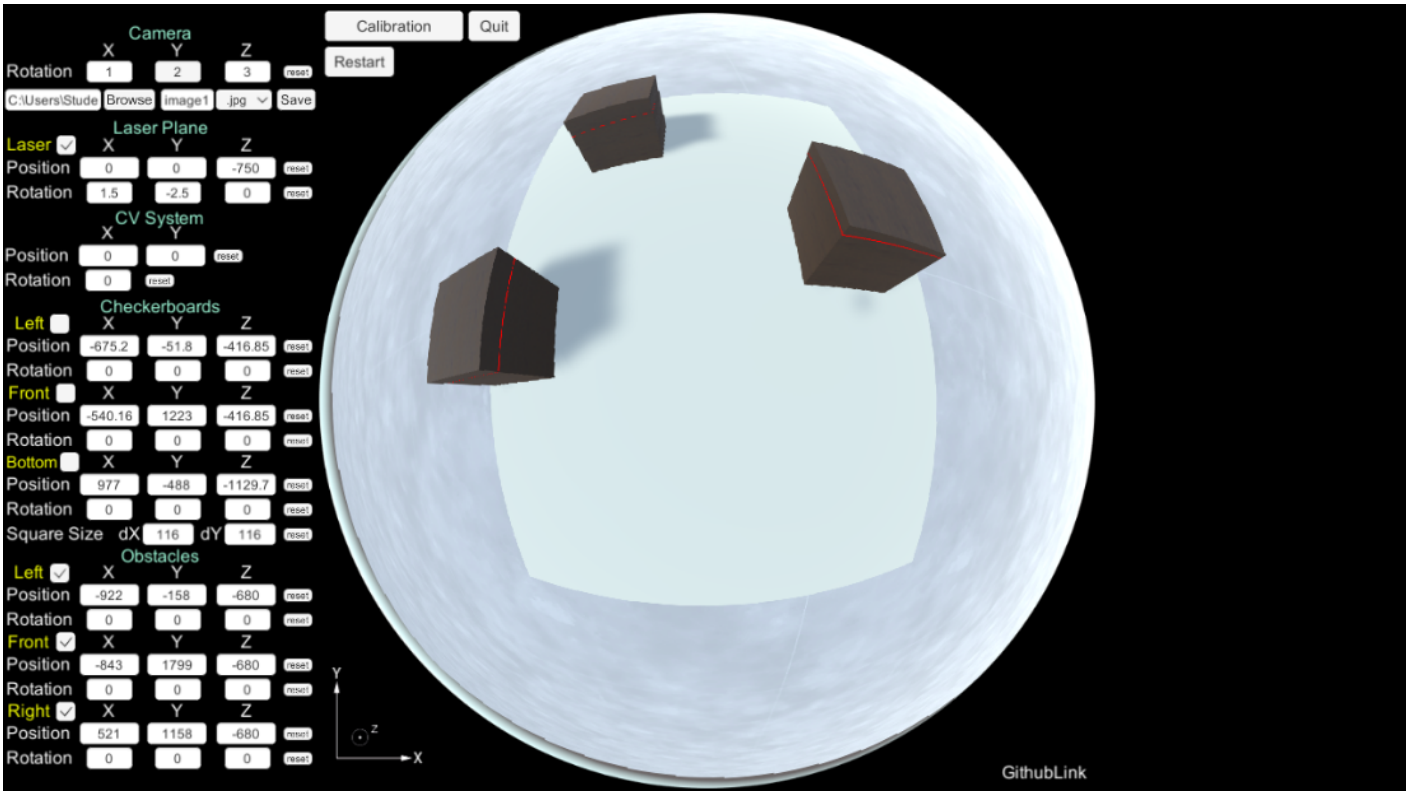
In order to calibrate our Vision System we need to place two checkerboards to our environment (activate them by main panel). And now we need to prepare two images: one without laser strip (for Camera calibration) and one with (for Laser Plane calibration and mapping).

!!! Notice, that size of the checkerboard squares should be similar for both pre-calibration and calibration procedures (it’s better to do not change default values of the “Square Size” for this tutorial). Same for the image size.



And for calibration results validation let’s activate several objects placed to the virtual environment (left cube, front cube and right wall) as shown in image below. Distances of the

objects are different from those distances of the checkerboards which were used during the calibration step. This approach reveals the performance of the calibration results in the environment as a whole.



Pre-Calibration Procedure

First of all you need to calibrate your camera, in case you have not done this so far – go to the Calibration Tutorial first.

Preparation

Now we need to prepare image with the region of environment interested for us. So let us activate two checkerboards. Moreover, let's rotate our laser plane around X-axis about 1.5º and around Y-axis about -2.5º. One picture we need to take without laser, second one with and third one with active laser and all of the obstacles. As a result, our image will look as in the picture above.

This tutorial has some supplementing Matlab files which should be included to the project folder (this files can be found in the tutorial folder):

- C_calib_data
- Calibrate
- cam2world
- compose_rotation
- cornerfinder
- crop_borders
- draw_axes
- export_fig
- FindTransformMatrix

- FUNrho
- get_checkerboard_cornersUrban
- print2array
- using_hg2

Camera calibration

Idea

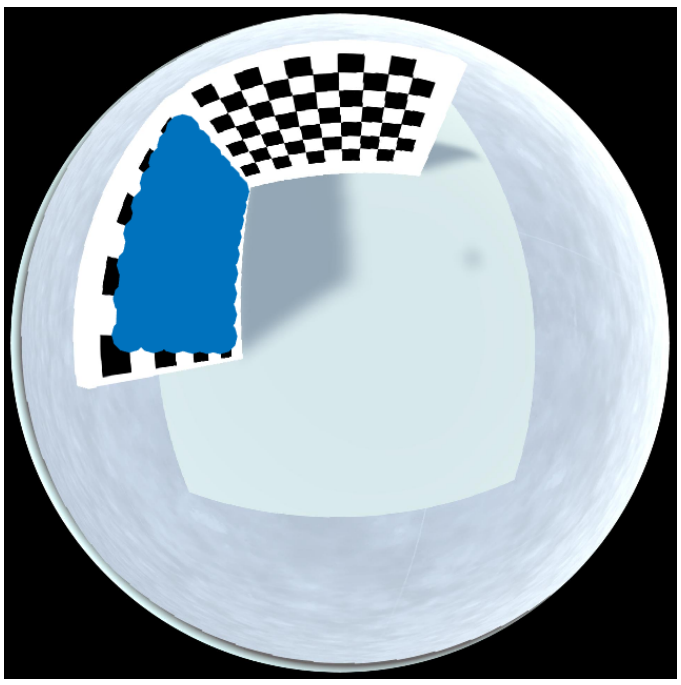
The configuration of two perpendicular checkerboards can give us an understanding of the camera orientation after calibration procedure, it means that in the ideal case we need to get our setup angles in degrees: [1;2;3].

Camera orientation

First of all let's load calibration parameters of our camera:

```
clc
clear all
load('Omni_Calib_Results.mat');
ocam_model = calib_data.ocam_model;
```

During this step we need to upload image without laser strip in order to extract image pixel coordinates of the checkerboard points (`ImagePoints` – for the left pattern and `ImagePoints1` – for the front one). In order to extract points of the second pattern simple function was written (`Get2ndPattern`). This function firstly obtains points of the first pattern then paints all of them and after that obtains points of the second pattern, therefore image will look the following way:



And `Get2ndPattern` function itself:

```
function [I,name] = Get2ndPattern(i,imagePoints,calib_data)
pause(0.8);
figure;
pause(0.8);
% set(gcf,'WindowState','fullscreen');
pause(0.8);
warning('off','Images:initSize:adjustingMag');
imshow(calib_data.L{i+1});
hold on
plot(imagePoints(:,1),imagePoints(:,2),'o-','LineWidth',19);
hold off
export_fig '2ndPattern.jpg' -native;
I = imread('2ndPattern.jpg');
name = '2ndPattern.jpg';
end
```

```
i = calib_data.n_ima;
calib_data.L(i+1)={'TestImages/image.jpg'};
use_corner_find=1;
[callBack,Xp_abs_,Yp_abs_] =
get_checkerboard_cornersUrban(i+1,use_corner_find,calib_data);
Xt = calib_data.Xt;
Yt = calib_data.Yt;
imagePoints = [Yp_abs_,Xp_abs_];
% second pattern
[II, name] = Get2ndPattern(i,imagePoints,calib_data);
calib_data.L(i+2)={name};
use_corner_find=1;
[callBack,Xp_abs_1,Yp_abs_1] =
get_checkerboard_cornersUrban(i+2,use_corner_find,calib_data);
imagePoints1 = [Yp_abs_1,Xp_abs_1];
```

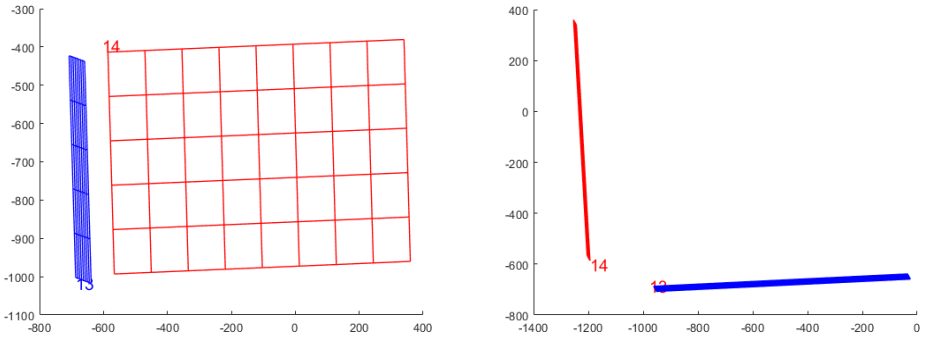
Once checkerboard points were found it becomes possible to compute location and orientation of the patterns itself towards camera. These extrinsic parameters are found by theory described in the paper [7] with Matlab functions “calibrate” and “FindTransformMatrix” which are included to the OcamCalib toolbox.

```
% first image extrinsic
if max(imagePoints(:,1)) > max(imagePoints1(:,1))
    imagePoints0 = imagePoints;
    imagePoints = imagePoints1;
    imagePoints1 = imagePoints0;
    Xp_abs_ = imagePoints(:,2);
    Yp_abs_ = imagePoints(:,1);
    Xp_abs_1 = imagePoints1(:,2);
    Yp_abs_1 = imagePoints1(:,1);
end
[RRfin,ss]=calibrate(Xt, Yt, Xp_abs_, Yp_abs_, ocam_model);
RRfin_=FindTransformMatrix(Xp_abs_, Yp_abs_, Xt, Yt, ocam_model, RRfin);
% second image extrinsic
[RRfin1,ss]=calibrate(Xt, Yt, Xp_abs_1, Yp_abs_1, ocam_model);
RRfin1_=FindTransformMatrix(Xp_abs_1, Yp_abs_1, Xt, Yt, ocam_model, RRfin);
```

By knowing transformation matrix, we can reproject pattern points from image plane to the world space. For this procedure let's create function (`M` – world points of the left pattern and `M1` – world points of the front one):

```
% obtain camera orientation and distance to patterns from camera
[M,M1] = show_patterns(0,0,0,i,RRfin_,RRfin1_,calib_data); % 0,0,0 - means that camera is not rotated (we don't know its rotation)
```

As a result we will get the following plot:



With the aid of the last figure we can calculate rotation of the patterns by simple geometry.

```
% first pattern
angle_y = atan(diff([M(3,9),M(3,1)])/diff([M(1,9),M(1,1)]))*180/pi;
angle_z = atan(diff([M(2,9),M(2,1)])/diff([M(1,9),M(1,1)]))*180/pi;
angle_x = atan(diff([M(2,46),M(2,1)])/diff([M(3,46),M(3,1)]))*180/pi;
% second pattern
angle_y1 = atan(diff([M1(1,46),M1(1,1)])/diff([M1(3,46),M1(3,1)]))*180/pi;
angle_z1 = atan(diff([M1(1,9),M1(1,1)])/diff([M1(2,9),M1(2,1)]))*180/pi;
angle_x1 = atan(diff([M1(3,9),M1(3,1)])/diff([M1(2,9),M1(2,1)]))*180/pi;
% mean value with regards to the two patterns
camX = sign(angle_x)*(abs(angle_x)+abs(angle_x1))/2;
camY = sign(angle_y)*(abs(angle_y)+abs(angle_y1))/2;
camZ = sign(angle_z)*(abs(angle_z)+abs(angle_z1))/2;
```

```
function [Mcc_,Mcc1_] = show_patterns(x,y,z,i,RRfin_,RRfin1_,calib_data)
a=i+1;
b=1;
calib_data.RRfin(:,:,i+1)= RRfin_;
calib_data.RRfin(:,:,i+2)= RRfin1_;
ddX=16;
ddY=16;
colors = 'brgkcm';
figure;
for a=i+1:i+2

    M=[calib_data.Xt';calib_data.Yt';ones(size(calib_data.Xt'))];
    Mc=rotz(-z)*roty(y)*rotx(x)*calib_data.RRfin(:,:,a)*M;
    if b==1
        Mcc_=Mc;
    else
        Mcc1_=Mc;
    end
    %Show extrinsic
          uu = [-ddX;-ddY;1];
          uu = calib_data.RRfin(:,:,a) * uu;
          YYx = zeros(calib_data.n_sq_x+1,calib_data.n_sq_y+1);
          YYy = zeros(calib_data.n_sq_x+1,calib_data.n_sq_y+1);
          YYz = zeros(calib_data.n_sq_x+1,calib_data.n_sq_y+1);

          YYx=reshape(Mc(1,:),calib_data.n_sq_y+1,calib_data.n_sq_x+1)';
          YYy=reshape(Mc(2,:),calib_data.n_sq_y+1,calib_data.n_sq_x+1)';
          YYz=reshape(Mc(3,:),calib_data.n_sq_y+1,calib_data.n_sq_x+1)';

          hold on;
          hhh= mesh(YYx,YYy,YYz);
          % axis equal;
          set(hhh,'edgecolor',colors(rem(a-1,6)+1),'linewidth',1); ...
              %,'facecolor','none');
          text(uu(1),uu(2),uu(3),num2str(a),'fontsize',14,'color',...
              colors(rem(a-1,6)+1));
          b=b+1;
end
hold off
end
```

Congratulations! We've just determined rotation of our Camera (`camX, camY, camZ`) and we can compare it with the real values:

	X, degrees	Y, degrees	Z, degrees
Real values	1	2	3
Experiment values	0.98	2.03	3.03
Error (Real – Experiment)	0.02	0.02	0.03

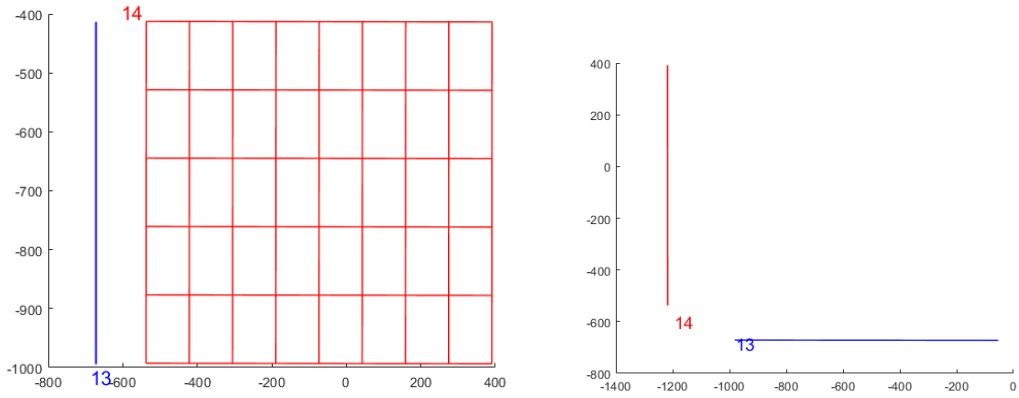
As you can see orientation of the camera can accurately be obtained with the aid of calibration!

Checkerboards location

Let's reproject pattern points again, but now we will do it with already known camera rotation. After that it'll be possible to calculate distances from camera to the patterns (this is needed for the next step which is relater with the Laser Plane calibration):

```
% find distance
[M,M1] = show_patterns(camX,camY,camZ,i,RRfin_,RRfin1_,calib_data); %
camX,camY,camZ - means that camera is rotated (we know its rotation)
Y1 = mean(M(2,:));
Y2 = mean(M1(1,:));
t1 = [0;Y1;0];
r1 = [1,0,0;0,1,0;0,0,1];
r1 = [r1(:,1),r1(:,3),t1];
r1_ = compose_rotation(-angle_x, -angle_y, angle_z);
r1 = r1_*r1;
t2 = [Y2;0;0];
r2 = [1,0,0;0,1,0;0,0,1];
r2 = [r2(:,2),r2(:,3),t2];
r2_ = compose_rotation(angle_x1, angle_y1, -angle_z1);
r2 = r2_*r2;
```

As a result we will get the following plot:



`r1` and `r2` are transformation matrix of the checkerboard patterns with regards to the camera. And we can also compare how accurate we obtained distances to the patterns by comparing values `Y1` and `Y2` with the real ones:

	Y1, mm	Y2, mm
Real values	675.2	1223.0
Experiment values	671.7	1217.8
Error (Real – Experiment)	3.5	5.2

Laser Plane calibration

Idea

Calculate incline of each laser strip belonging to the checkerboard patterns.

Laser Segmentation

During this step we need to upload image with laser strip in order to calibrate Laser Plane itself.

Our first Matlab function here will be related with the laser strip extraction. We will use simple image segmentation. After laser extraction we will apply morphological operation which is known as skeletonization.
Let's load our image for further operation.

```
I1 = imread('TestImages/image1.jpg');
```

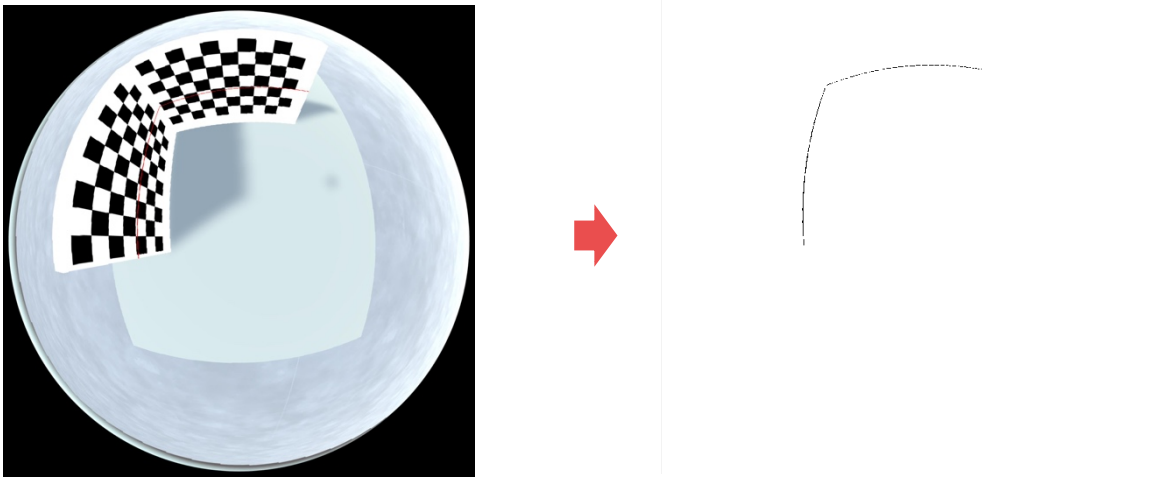
Now we need to create the new function for laser strip extraction.

```
I = las_segm(I1);
```

```
function BW = las_segm(img)
% http://matlabtricks.com/post-35/a-simple-image-segmentation-example-in-
warning('off','Images:initSize:adjustingMag');
image = img;     % read image
[height, width, planes] = size(image);
rgb = reshape(image, height, width * planes);
imagesc(rgb);                    % visualize RGB planes
r = image(:, :, 1);              % red channel
g = image(:, :, 2);              % green channel
b = image(:, :, 3);              % blue channel
threshold = 100;                 % threshold value
imagesc(r < threshold);          % display the binarized image
blueness = double(r) - max(double(g), double(b));
imagesc(blueness);               % visualize RGB planes
mask = blueness < 78;
imagesc(mask);
% labels = bwlabel(mask);
R(~mask) = 255;
G(~mask) = 255;
B(~mask) = 255;
J = cat(3,R,G,B);
BW = ~mask;
% Skeletonization
% BW = im2bw(J,0.4);
% BW = bwmorph(BW,'skel',Inf);
imshow(~BW);
end
```

!!! Notice, that "mask" value blueness < 78 was set for particular width of the laser, so if you're planning to change laser width, change "threshold" value as well.

After applying the above-mentioned code we will have only extracted laser strip part which is shown in the picture below:



Laser segmentation within patterns

Image above contains laser strip belonging to the whole environment, however we are only interested in the regions lying within checkerboard patterns. Therefore, we need to write a function of the region mask creation in order to extract laser points belonging to the patterns. Moreover, extracted points might contain some noise, so we will eliminate it by the least square curve fitting. As a result, we will get two arrays with the laser pixel coordinates within our patterns (`Cent` – points belonging to the left pattern, `Cent1` – points belonging to the front one).

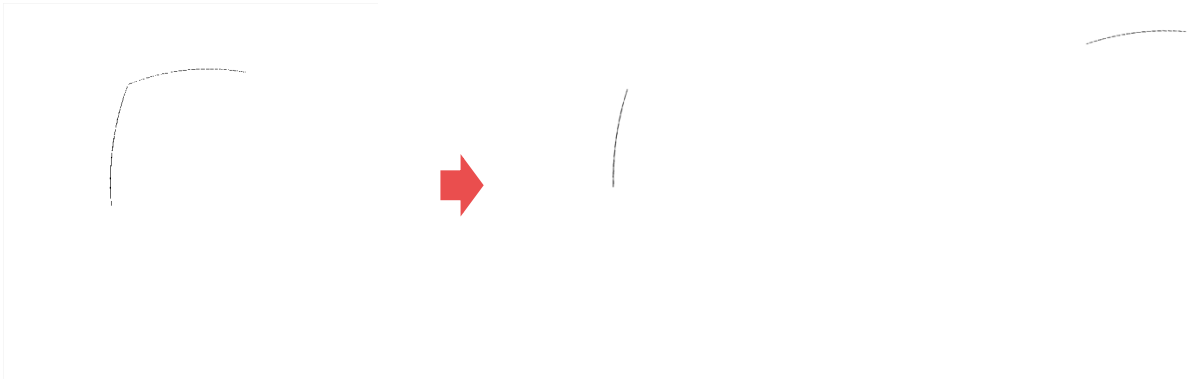
```
[Cent,Cent1] = intersections_2img_curve(I,imagePoints,imagePoints1);
```

```
function [Cent,Cent1] = intersections_2img_curve(I,imgPoints0,imgPoints1)
[height,width] = size(I);
% first image
% Change matrix of the checkherboard
b=1;
d=1;
for c=1:9
    d = c;
    for i=1:6
        Im(b,:) = imgPoints0(d,:);
        b = b+1;
        d = d+9;
    end
end
Xp_abs_1 = Im(:,2);
Yp_abs_1 = Im(:,1);
imgPoints0 = [Yp_abs_1,Xp_abs_1];
% mask http://qaru.site/questions/13446091/creating-a-mask-using-a-binary-
image-matlab
% poly mask https://ww2.mathworks.cn/help/images/ref/poly2mask.html
img = I;
img = im2double(img);
y=Yp_abs_1;
x=Xp_abs_1;
xi = [y(1);y(2);y(3);y(4);y(5);y(6);y(12);y(18);y(24);y(30);y(36);y(42);...
      y(48);y(54);y(53);y(52);y(51);y(50);y(49);y(43);y(37);y(31);y(25);...
      y(19);y(13);y(7)];
yi = [x(1);x(2);x(3);x(4);x(5);x(6);x(12);x(18);x(24);x(30);x(36);x(42);...
      x(48);x(54);x(53);x(52);x(51);x(50);x(49);x(43);x(37);x(31);x(25);...
      x(19);x(13);x(7)];
bw = poly2mask(xi,yi,height,width);
a1=round(min(Xp_abs_1));
a2=round(max(Xp_abs_1));
b1=round(min(Yp_abs_1));
b2=round(max(Yp_abs_1));
res2 = img.*bw;
res2 = ~res2;
imshow(res2);
export_fig Test.png -native;
RGB = imread('Test.png');
RGB = ~RGB;
imshow(RGB);
image = RGB;
```

```
a = 1;
x_1=[];
z_1=[];
for j = a1:a2     % working image region
    for i= b1:b2
        if image(j,i)>0
            m=[j;i];       % image pixels
            z_1(a) = j;
            x_1(a) = i;
            a=a+1;
        end
    end
end
plot(z_1,x_1,'mo');
hold on
% fitobject=fit(z_1',x_1','poly2','Normalize','on','Robust','Bisquare');
fitobject=polyfit(z_1',x_1',2); % f(x) = p1*x^2 + p2*x + p3
plot(fitobject);
v = 1;
z_1 = sort(z_1);
for i = z_1(1):z_1(length(z_1))
    x(v) = i;
    % y(v) = fitobject(x(v));
    y(v) = fitobject(1)*(x(v))^2+fitobject(2)*x(v)+fitobject(3);
    v = v+1;
end
plot(x,y,'bo');
hold off
Cent=[y,x];
b=1;
d=1;
for c=1:9
    d = c;
    for i=1:6
        Im(b,:) = imgPoints1(d,:);
        b = b+1;
        d = d+9;
    end
end
Xp_abs_1 = Im(:,2);
Yp_abs_1 = Im(:,1);
imgPoints1 = [Yp_abs_1,Xp_abs_1];
% mask http://qaru.site/questions/13446091/creating-a-mask-using-a-binary-
image-matlab
% poly mask https://ww2.mathworks.cn/help/images/ref/poly2mask.html
img = I;
img = im2double(img);
y=Yp_abs_1;
x=Xp_abs_1;
```

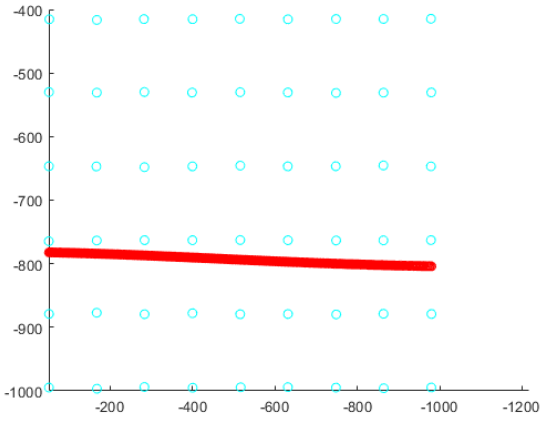
```
xi = [y(1);y(2);y(3);y(4);y(5);y(6);y(12);y(18);y(24);y(30);y(36);y(42);...
      y(48);y(54);y(53);y(52);y(51);y(50);y(49);y(43);y(37);y(31);y(25);...
      y(19);y(13);y(7)];
yi = [x(1);x(2);x(3);x(4);x(5);x(6);x(12);x(18);x(24);x(30);x(36);x(42);...
      x(48);x(54);x(53);x(52);x(51);x(50);x(49);x(43);x(37);x(31);x(25);...
      x(19);x(13);x(7)];
bw = poly2mask(xi,yi,height,width);
a1=round(min(Xp_abs_1));
a2=round(max(Xp_abs_1));
b1=round(min(Yp_abs_1));
b2=round(max(Yp_abs_1));
res2 = img.*bw;
res2 = ~res2;
imshow(res2);
export_fig Test.png -native;
RGB = imread('Test.png');
RGB = ~RGB;
imshow(RGB);
image = RGB;
a = 1;
x_1=[];
z_1=[];
for j = a1:a2     % working image region
    for i= b1:b2
        if image(j,i)>0
            m=[j;i];       % image pixels
            z_1(a) = j;
            x_1(a) = i;
            a=a+1;
        end
    end
end
plot(x_1,z_1,'mo');
hold on
% fitobject=fit(x_1',z_1','poly2','Normalize','on','Robust','Bisquare');
fitobject=polyfit(x_1',z_1',2); % f(x) = p1*x^2 + p2*x + p3
plot(fitobject);
v = 1;
x_1 = sort(x_1);
for i = x_1(1):x_1(length(x_1))
    x(v) = i;
    % y(v) = fitobject(x(v));
    y(v) = fitobject(1)*(x(v))^2+fitobject(2)*x(v)+fitobject(3);
    v = v+1;
end
plot(x,y,'bo');
hold off
Cent1=[x,y];
end
```

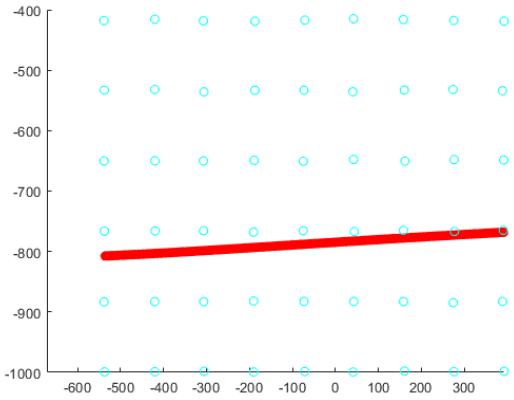
Extracted laser part belonging to the checkerboard regions are represented below:

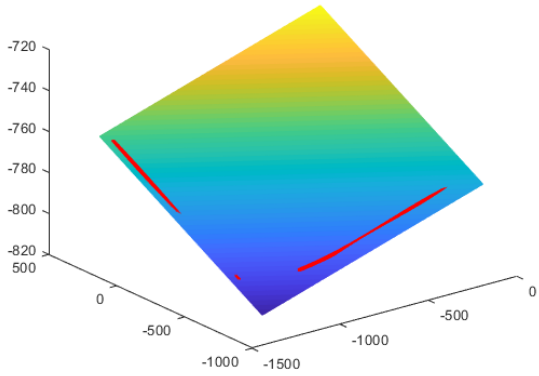


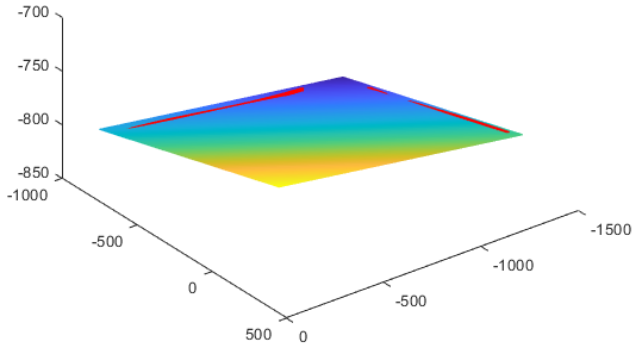
Laser incline

Once image pixel coordinates were found, we can re-project them to the world coordinates. And in order to facilitate understanding we will re-project laser points together with the checkerboard points. Among laser world points we will fit the plane and find its incline its geometry. As a result we will obtain orientation of our Laser Plane.
Figures of checkerboard patterns together with the laser strip are presented below (that's what we'll get after this step):










```
% world intersections
% first chess
[z_1,x_1] = intersections_world(Cent,r1,ocam_model);
```

```
[Z_1,X_1] = intersections_world(imagePoints,r1,ocam_model);
hh = figure;
hh = axes;
set(hh, 'Xdir', 'reverse');
hold on
plot(x_1,z_1,'ro');
xlim([Y2 inf]);
% h = lsline;
plot(X_1,Z_1,'co');
xlim([Y2 inf]);
hold off
% second chess
[z_2,x_2] = intersections_world(Cent1,r2,ocam_model);
[Z_2,X_2] = intersections_world(imagePoints1,r2,ocam_model);
figure;
hold on
plot(x_2,z_2,'ro');
xlim([Y1 inf]);
% h = lsline;
plot(X_2,Z_2,'co');
xlim([Y1 inf]);
hold off
%% fit plane to the world laser points
X_p1 = x_1;
X_p1 = reshape(X_p1,[length(X_p1),1]);
Y_p1 = [];
for j = 1:length(x_1)
    Y_p1(j) = Y1;
end
for j = 1:length(x_2)
    X_p1(length(x_1)+j) = Y2;
end
Y_p1 = [Y_p1, x_2];
Y_p1 = reshape(Y_p1,[length(Y_p1),1]);
Z_p1 = [z_1,z_2];
Z_p1 = reshape(Z_p1,[length(Z_p1),1]);
% https://youtu.be/U4eRSL16KzA
A = [X_p1, Y_p1, ones(size(X_p1))];
v = Z_p1;
result = A\v;
a = result(1);
b = result(2);
c = result(3);
% distance between camera and laser plane
z_las_dist = a*0 + b*0 + c;
lasX_ = -(atand(((a*0+b*0+c)-(a*0+b*300+c))/300));
lasY_ = (atand(((a*0+b*0+c)-(a*300+b*0+c))/300));
lasX = lasX_;
lasY = lasY_;
%% plot plane with the fitted laser points
figure;
plot3(X_p1,Y_p1,Z_p1,'r.');
hold on
[XXX YYY] = meshgrid(-1250:0,-750:450);
Zplot = a*XXX+b*YYY+c;
mesh(XXX,YYY,Zplot)
```

```
function [Z_1,X_1] = intersections_world(Cent,r,ocam_model)
if length(Cent(:,1)) > 1
    for i = 1:length(Cent)
        x = Cent(i,2);
        y = Cent(i,1);
        m = [x;y];
        M=cam2world(m, ocam_model);
        a1 = M(1)*r(2,1)-M(2)*r(1,1);
        b1 = M(1)*r(2,2)-M(2)*r(1,2);
        c1 = M(1)*r(2,3)-M(2)*r(1,3);
        a2 = M(3)*r(1,1)-M(1)*r(3,1);
        b2 = M(3)*r(1,2)-M(1)*r(3,2);
        c2 = M(3)*r(1,3)-M(1)*r(3,3);
        Z_1(i) = (a2*c1-a1*c2)/(a1*b2-a2*b1);
        X_1(i) = (-c1-b1*Z_1(i))/a1;
    end
end
end
```

Congratulations! We've just determined rotation of our Laser Plane (`lasX_, lasY_`), as well as the distance between camera and laser plane and we can compare it with the real values:

	X, degrees	Y, degrees	z_lasdist, mm
Real values	1.59	-2.59	750
Experiment values	1.59	-2.41	751
Error (Real – Experiment)	0.09	0.18	1

So here we also obtained accurate orientation results as it was with the camera orientation!

Mapping function

Once we calibrated our Vision System it makes sense to carried out mapping.
First of all, let's load new image

```
image = imread('TestImages/image2.jpg');
figure;
```

Now we need to extract laser strip as we did before

```
img = las_segm(image);
```

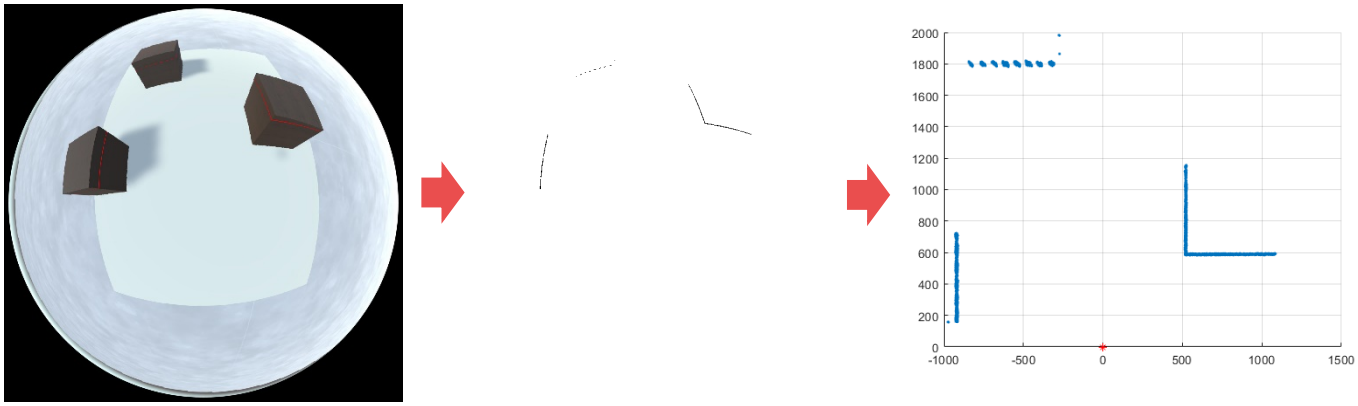
After that, we are ready to create mapping function:

```
function [x,y] =
mapping(image,camX,camY,camZ,lasX,lasY,las_dist,ocam_model)
[height,width] = size(image);
Z=las_dist;
a = 1;
x=[];
y=[];
t = [0;0;Z];
r = compose_rotation(lasX, lasY, 0);
% r = rotz(camZ)*roty(-camY)*rotx(-camX)*[r(:,1),r(:,2),t];
r1 = compose_rotation(-camX, -camY, camZ);
r = r1*[r(:,1),r(:,2),t];

for i=1:height      % working image region
    for j=1:width
        if image(i,j)>0
            m=[i;j];       % image pixels
            M = cam2world(m,ocam_model); % transform from image plane to
the camera plane
            a1 = M(1)*r(2,1)-M(2)*r(1,1);
            b1 = M(1)*r(2,2)-M(2)*r(1,2);
            c1 = M(1)*r(2,3)-M(2)*r(1,3);
            a2 = M(3)*r(1,1)-M(1)*r(3,1);
            b2 = M(3)*r(1,2)-M(1)*r(3,2);
            c2 = M(3)*r(1,3)-M(1)*r(3,3);

            Y = (a2*c1-a1*c2)/(a1*b2-a2*b1);
            X = (-c1-b1*Y)/a1;

            y(a)=X;
            x(a)=-Y;
            a=a+1;
        end
    end
end
end
```

Finally, let's set up our calibrated parameters and plot mapping results:

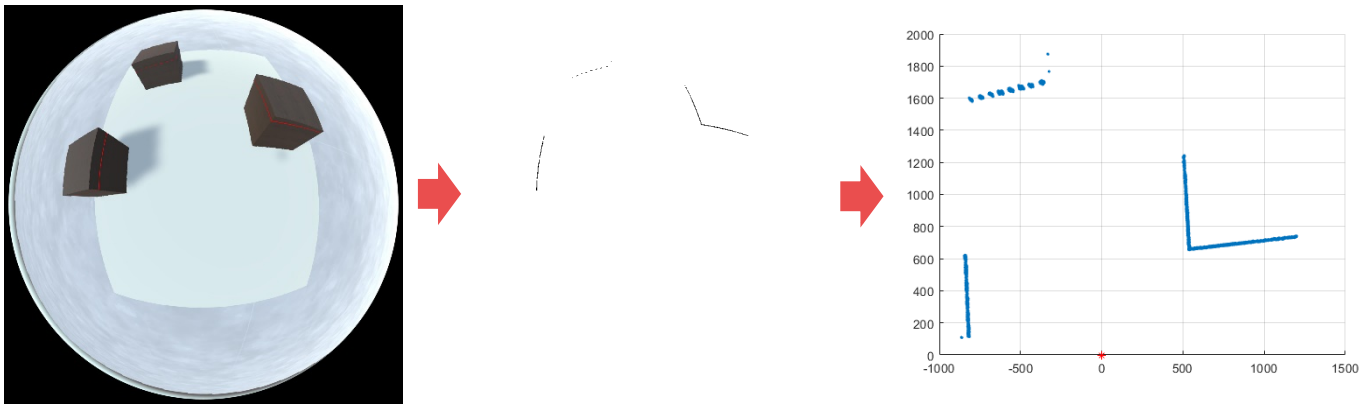
```
[x,y] = mapping(img,camX,camY,camZ,lasX_,lasY_,z_las_dist,ocam_model);
robot_x = 0; % Robot position
robot_y = 0; % Robot position
% Finally figure:
figure;
scatter(x,y,5,'filled'); % Laser intersections
hold on;
plot(robot_x,-robot_y,'r*'); % Robot location
grid on;
```

Now we can compare our input image with the output one. As you can see, output image contains only distance information of the laser strip intersection with the obstacles:



For comparing let's consider not calibrated case, how the map will look like if we set up angles related with the camera and laser plane orientations equal to zeros:

```
[x,y] = mapping(img,0, 0, 0, 0_,0_,750,ocam_model);
robot_x = 0; % Robot position
robot_y = 0; % Robot position
% Finally figure:
figure;
scatter(x,y,5,'filled'); % Laser intersections
hold on;
plot(robot_x,-robot_y,'r*'); % Robot location
grid on;
```

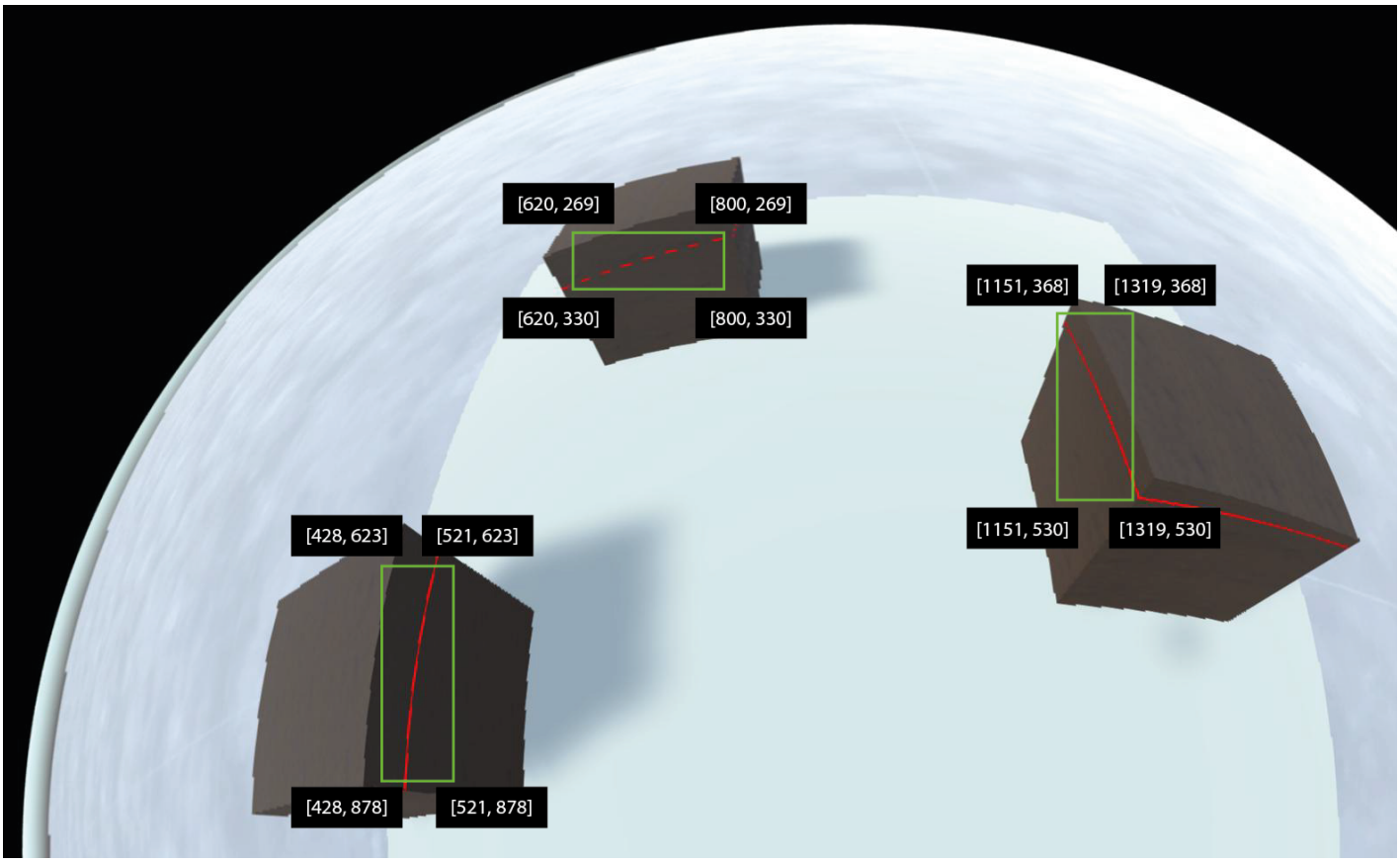


From the last figure we can see that even small deviations can lead to the wrong measurements, that's why calibration process is important.

Results validation

After mapping it becomes possible to compare distances from the experiment with the real ones. You can use plotted figure and mouse cursor for it. However, it's better to take mean value among all of the laser points belonging to the desirable obstacle, because it will help you to eliminate some noise. Let's consider function for the second case.

First of all we need to find interested pixel coordinates for us in order to find laser points belonging to that region, see picture below:



Once coordinates were determined we can write function, which will project pixel coordinates to the world ones (`Dist`).

```
function [Dist] =
cube_dist(I,i_,j_,camX,camY,camZ,lasX,lasY,las_dist,ocam)
Z=las_dist;
a = 1;
x1=[];
y1=[];
t = [0;0;Z];
r = compose_rotation(lasX, lasY, 0);
% r = rotz(camZ)*roty(-camY)*rotx(-camX)*[r(:,1),r(:,2),t];
r1 = compose_rotation(-camX, -camY, camZ);
r = r1*[r(:,1),r(:,2),t];

for i=i_(1):i_(2)     % working image region
    for j=j_(1):j_(2)
        if I(j,i)>0
            m=[j;i];       % image pixels
            M = cam2world(m,ocam); % transform from image plane to the
camera plane
            a1 = M(1)*r(2,1)-M(2)*r(1,1);
            b1 = M(1)*r(2,2)-M(2)*r(1,2);
            c1 = M(1)*r(2,3)-M(2)*r(1,3);
            a2 = M(3)*r(1,1)-M(1)*r(3,1);
            b2 = M(3)*r(1,2)-M(1)*r(3,2);
            c2 = M(3)*r(1,3)-M(1)*r(3,3);

            Y = (a2*c1-a1*c2)/(a1*b2-a2*b1);
            X = (-c1-b1*Y)/a1;

            y1(a)=Y;
            x1(a)=X;
            a=a+1;
        end
    end
end
Dist=[y1',x1'];
end
```

And finally we will take mean value of each region

```
% Right Cube
i=[1595;1720]; % working image region
j=[690;1240]; % working image region
[Cube_R] = cube_dist(img,i,j,camX,camY,camZ,lasX_,lasY_,z_las_dist,ocam_model);
[x,y] = mapping(img,camX,camY,camZ,lasX_,lasY_,z_las_dist,ocam_model);
Cube_R = mean(Cube_R(:,1));
% Left Cube
i=[565;690]; % working image region
j=[545;1340]; % working image region
[Cube_L] = cube_dist(img,i,j,camX,camY,camZ,lasX_,lasY_,z_las_dist,ocam_model);
Cube_L = mean(Cube_L(:,1));
% Front Cube
i=[690;1235]; % working image region
j=[345;455]; % working image region
[Cube_F] = cube_dist(img,i,j,camX,camY,camZ,lasX_,lasY_,z_las_dist,ocam_model);
Cube_F = mean(Cube_F(:,2));
```

Now we are able to compare variables:

	Left Cube (Cube_L), mm	Up Cube (Cube_F), mm	Right Cube (Cube_R), mm
Real values	-922.00	1799.00	521.00
Experiment values	-919.26	1799.70	521.31
Error (Real – Experiment)	2.74	0.70	0.31

REFERENCES:

1. I. Kholodilin, Y. Li, and Q. Wang, “Model of an Omnidirectional Vision System with Laser Illumination and its Calibration Based on Simulation,” IEEE Trans. on Instrument and Measurement, vol. 59, no. 7, pp. 1841–1849, Jul. 2010, 10.1109/TIM.2009.2028222.