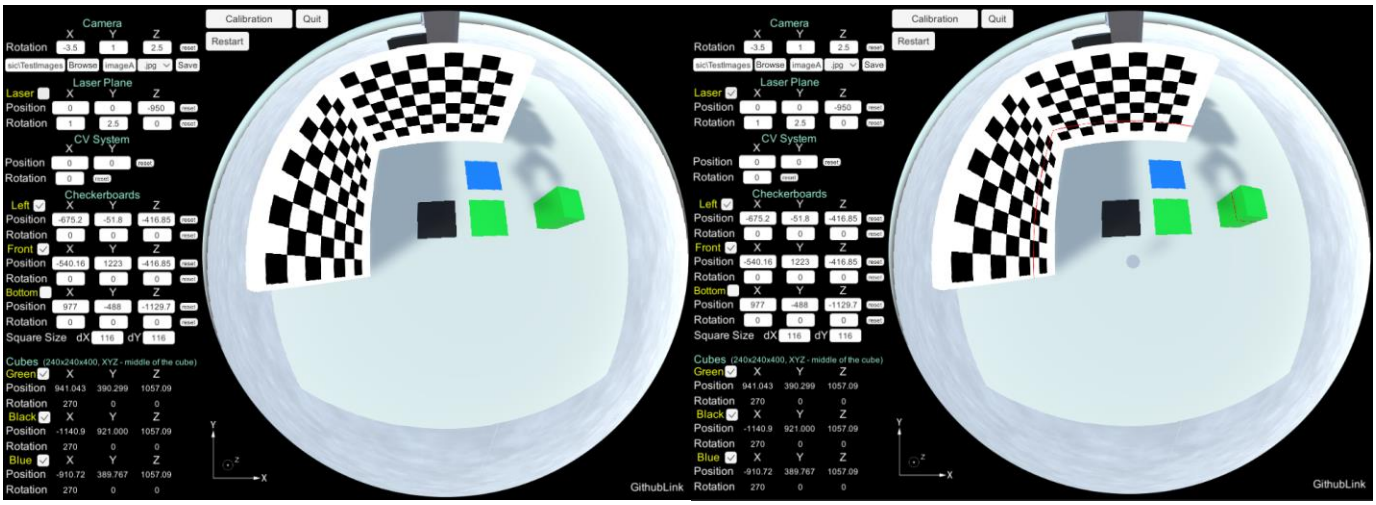
ВНЕШНЯЯ КАЛИБРОВКА

Это руководство основано на статье «Model of an Omnidirectional Vision System with Laser Illumination and its Calibration Based on Simulation» [1]. Во-первых, настоятельно рекомендуется прочитать эту статью. Цель этого руководства заключается в том, что вы поймете важность калибровки системы компьютерного зрения, а также можете применить ее к реальным системам.
Мы настроим конфигурацию нашей системы компьютерного зрения только для того, чтобы сравнить результаты нашего эксперимента со следующими:

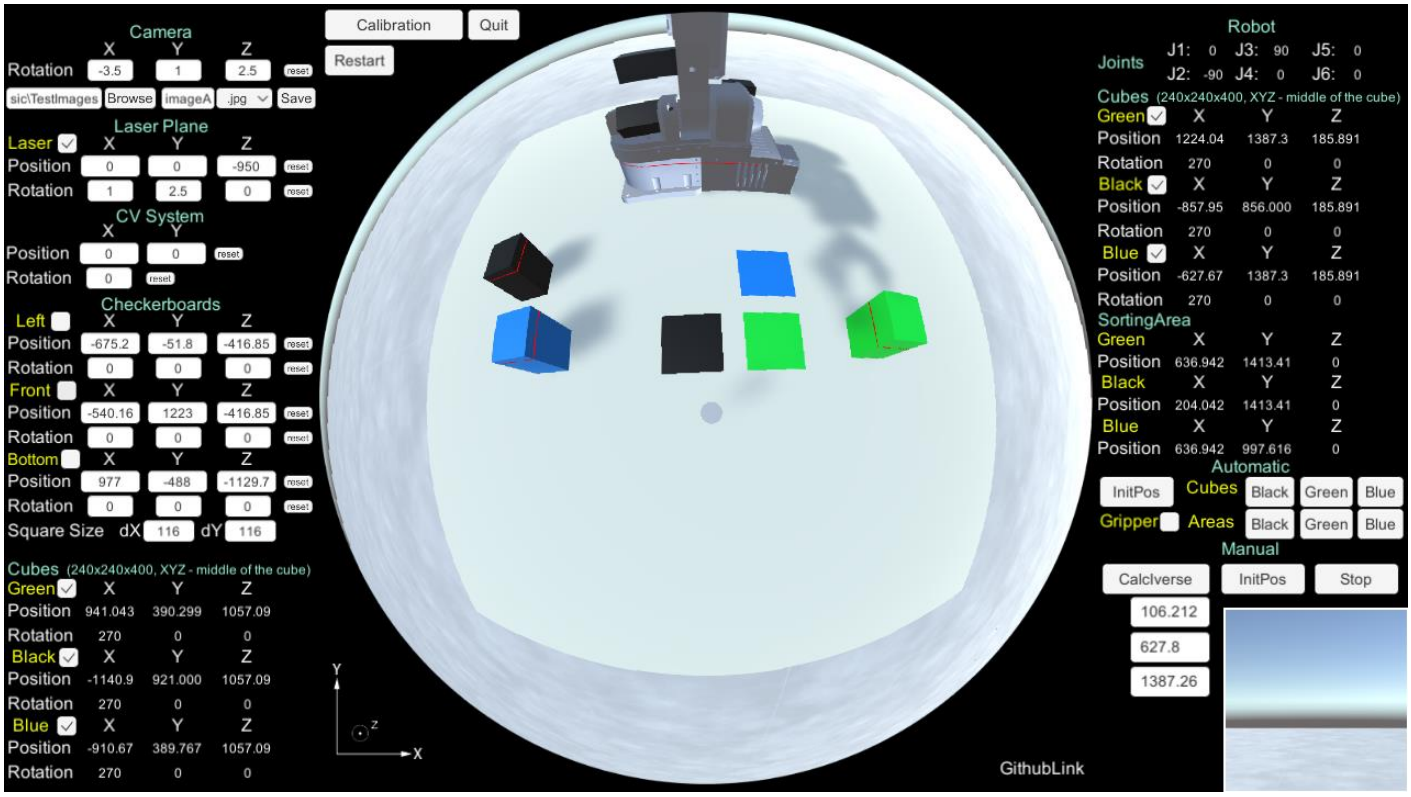
- Камера поворачивается вокруг оси X на -3,5^{o}, вокруг оси Y на 1^{o} и вокруг оси Z на 2,5^{o};
- Лазерная плоскость поворачивается вокруг оси X на 1 ^{o} и вокруг оси Y на 2,5^{o};
- Расстояние между камерой и плоскостью лазера 950 мм;

Чтобы откалибровать нашу систему, нам нужно разместить две шахматные доски (активировать их с помощью главной панели). Далее нам нужно подготовить два изображения: одно без лазера (для калибровки камеры) и одно с лазером (для калибровки системы).

!!! Обратите внимание, что размер квадратов шахматной доски должен быть одинаковым как для процедур внутренней калибровки, так и для внешней калибровки (лучше не изменять значения по умолчанию «Размер квадрата»). То же самое и с размером изображения.



Для оценки результатов калибровки будем использовать несколько объектов, размещенных в виртуальной среде (черный куб, синий куб и зеленый куб), как показано на рисунке ниже. Расстояния до объектов отличаются от расстояний до шахматных досок, которые использовались на этапе калибровки. Этот подход показывает эффективность результатов калибровки в окружающей среде в целом.



Процедура предварительной калибровки

Прежде всего, вам необходимо откалибровать камеру, если вы еще этого не сделали – сначала перейдите к руководству по калибровке.

Подготовка

Теперь нам нужно подготовить изображение с интересующей нас областью. Итак, давайте активируем две шахматные доски. Камера поворачивается вокруг оси X на -3,5°, вокруг оси Y на 1° и вокруг оси Z на 2,5°; Кроме того, повернем нашу лазерную плоскость вокруг оси X на 1° и вокруг оси Y на 2,5°. Один снимок нам нужно сделать без лазера, второй с активным лазером и третий с активным лазером и всеми препятствиями. В результате наше изображение будет выглядеть как на картинке выше.

В этом руководстве есть несколько дополнительных файлов Matlab, которые должны быть включены в папку проекта:

- C_calib_data
- Calibrate
- cam2world
- compose_rotation

- cornerfinder
- crop_borders
- draw_axes
- export_fig
- FindTransformMatrix
- FUNrho
- get_checkerboard_cornersUrban
- print2array
- using_hg2

Калибровка камеры

Идея

Конфигурация двух перпендикулярных шахматных досок может дать нам представление об ориентации камеры после процедуры калибровки, это означает, что в идеальном случае нам нужно получить наши заданные углы в градусах: [-3,5;1;2,5].

Ориентация камеры

Прежде всего загрузим калибровочные параметры нашей камеры:

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

На этом шаге нам нужно загрузить изображение без лазера, чтобы извлечь пиксельные координаты точек шахматной доски изображения (`ImagePoints` — для левой мешени и `ImagePoints1` — для передней мешени). Для извлечения точек второй мешени была написана простая функция (`Get2ndPattern`). Эта функция сначала получает точки первой мешени, затем закрашивает их все, после чего получает точки второй мешени, изображение будет выглядеть следующим образом:

И сама функция `Get2ndPattern` :

```
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];
```

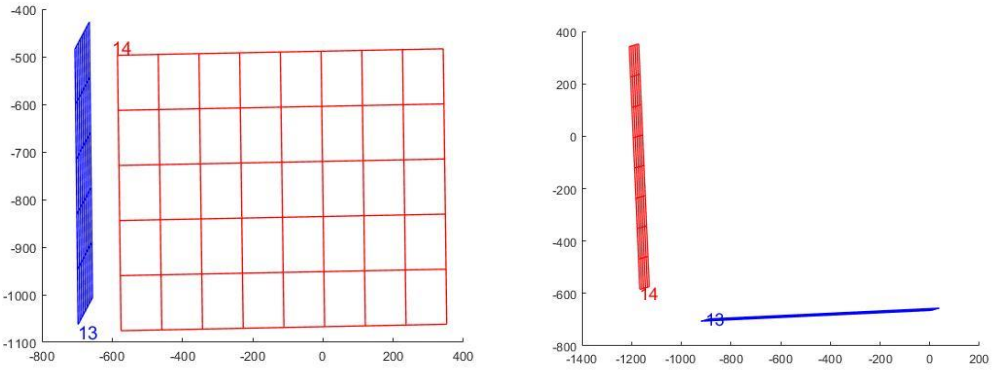
Как только точки шахматной доски были найдены, становится возможным вычислить местоположение и ориентацию самих мишеней по отношению к камере. Эти внешние параметры находятся с помощью теории, описанной в статье [7] с помощью функций Matlab «calibrate» и «FindTransformMatrix», которые включены в набор инструментов OcamCalib .

```
% 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);
```

Зная матрицу перехода, мы можем перепроецировать точки мишени с плоскости изображения в координаты пространства. Для этой процедуры создадим функцию (`M` – простанственные точки левой мешени и `M1` – простанственные точки передней мишени):

```
% 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)
```

В результате мы получим следующий график:



С помощью последней фигуры мы можем вычислить поворот мишеней геометрически.

```
% 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
```

Поздравляем! Мы только что определили поворот нашей камеры (`camX`, `camY`, `camZ`) и можем сравнить его с реальными значениями:

	X, градусов	Y, градусов	Z, градусов
Реальные значения	-3,50	1,00	2,50
Экспериментальные значения	-3,60	0,86	2,58
Ошибка	0,10	0,14	0,08

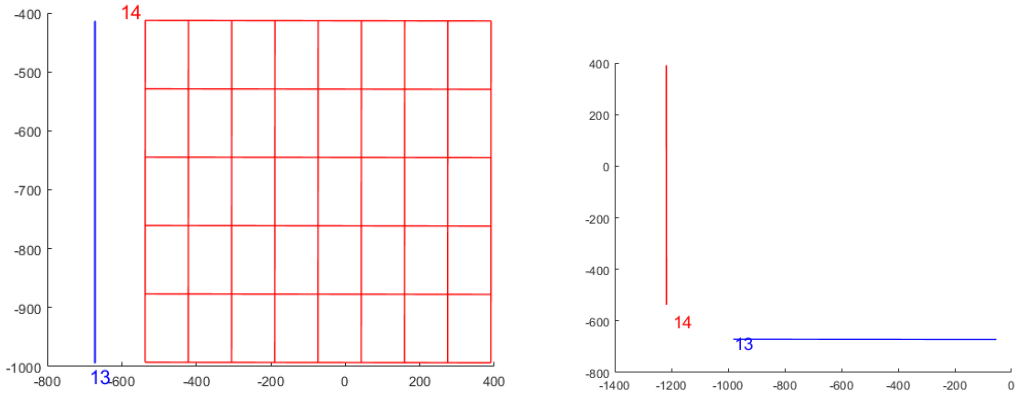
Как видите, ориентация камеры может быть точно получена с помощью калибровки!

Расположение шахматной доски

Перепроецируем точки мишени, но уже с уже известным поворотом камеры. После этого можно будет рассчитать расстояния от камеры до мишени (это необходимо для следующего шага, связанного с калибровкой лазерной плоскости):

```
% 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;
```

В результате мы получим следующий график:



`r1` и `r2` — матрицы преобразования шахматной доски относительно камеры. И мы также можем сравнить, насколько точно мы получили расстояния до мишеней, сравнив значения `Y1` и `Y2` с реальными:

	Y1, мм	Y2, мм
Реальные значения	675,2	1223,0
Экспериментальные значения	669,6	1220,2
Ошибка	5,6	2,8

Калибровка лазерной плоскости

Идея

Рассчитайте наклон лазерной плоскости, вписав в нее плоскость в простанстве.

Лазерная сегментация

На этом этапе нам нужно загрузить изображение с лазером, чтобы откалибровать саму лазерную плоскость.

Наша первая функция Matlab здесь будет связана с извлечением лазера. Мы будем использовать простую сегментацию изображения. Загрузим наше изображение для дальнейшей работы.

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

Теперь нам нужно создать новую функцию для извлечения лазера.

```
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
```

Обратите внимание, что значение «маски» blueness <78 было установлено для определенной интенсивности лазера, поэтому, если вы планируете изменить интенсивность лазера (другой источник), измените также данное значение.

После применения вышеупомянутого кода у нас будет только извлеченная часть лазерной полосы, которая показана на рисунке ниже:

Лазерная сегментация с области мишеней

Изображение выше содержит лазерную полосу, принадлежащую всей окружающей среде, однако нас интересуют только области, находящиеся на шахматных досках. Поэтому нам нужно написать функцию создания маски, чтобы выделить лазерные точки, принадлежащие мишеням. Кроме того, выделенные точки могут содержать некоторый шум, поэтому мы удалим его методом вписания кривой наименьших квадратов. В результате мы получим два массива с координатами лазерных пикселей внутри наших мишеней (`Cent` – точки, принадлежащие левой мишени, `Cent1` – точки, принадлежащие передней мишени).

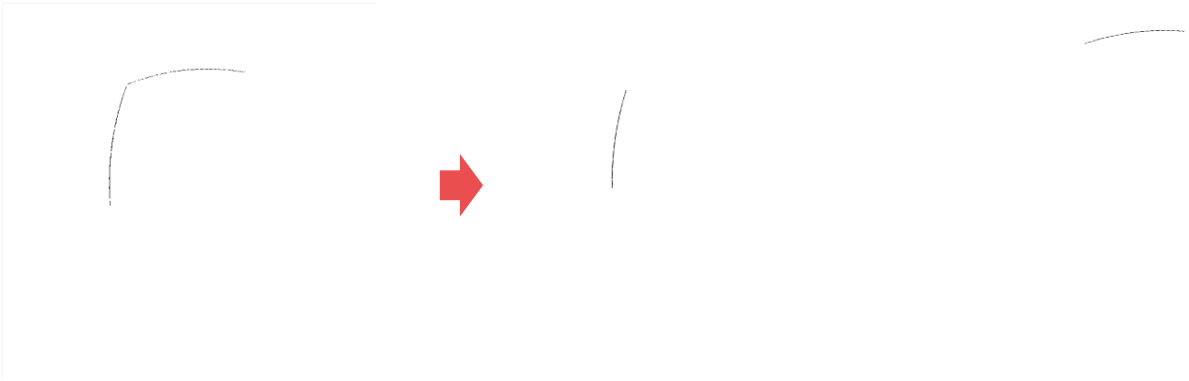
```
[Cent,Cent1] = intersections_2img_curve(I,imagePoints,imagePoints1);
```

```
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
```

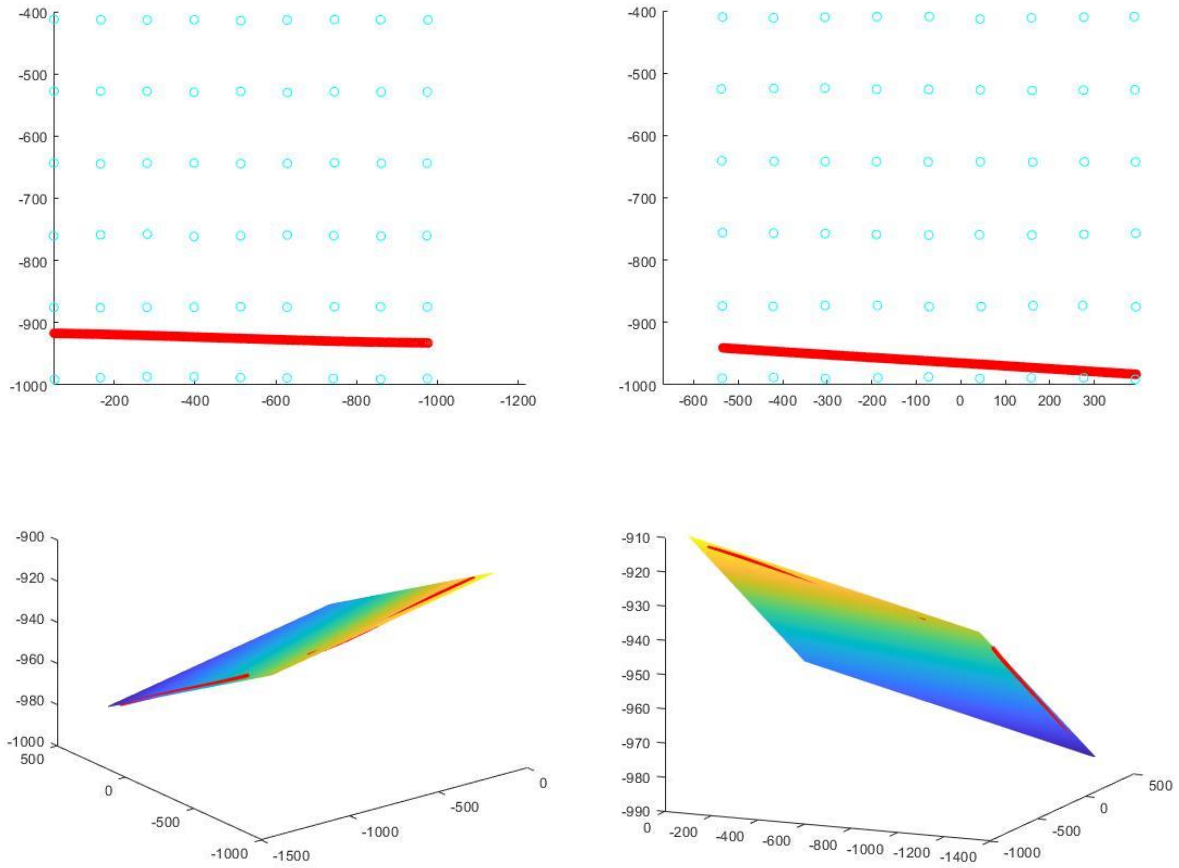
Извлеченная лазерная часть, принадлежащая областям шахматной доски, представлена ниже:



Наклон лазерной плоскости

Как только координаты пикселей изображения были найдены, мы можем перепроецировать их в координаты пространства. И для наглядности мы будем перепроецировать лазерные точки вместе с точками шахматной доски. В лазерные точки мы впишем плоскость и найдем геометрически ее наклон. В результате мы получим поворот нашей лазерной плоскости.

Шахматные доски вместе с лазером представлены ниже (вот что мы получим после этого шага):



```
% 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
```

Поздравляем! Мы только что определили поворот нашей лазерной плоскости (`lasX_`,`lasY_`), а также расстояние между камерой и лазерной плоскостью и можем сравнить их с реальными значениями:

	X, градусов	Y, градусов	z_lasdist, мм
Реальные значения	1,59	-2,59	950,00
Экспериментальные значения	1,59	-2,41	945,48
Ошибка	0,09	0,18	4,52

Здесь мы также получили точные результаты поворота, как это было с ориентацией камеры!

Функция построения карты

После того, как мы откалибровали нашу систему, проведем измерения до препятствий. Прежде всего, давайте загрузим новое изображение

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

Теперь нам нужно извлечь лазер, как мы это делали раньше.

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

После этого мы готовы создать функцию для построения карты глубины:

```
function [x,y] =
mapping(image,camX,camY,camZ,lasX,lasY,las_dist,ocam_model)
[height,width] = size(image);
Z=las_dist;
a = 2;
x=[];
y=[];
t = [0;0;Z];
r = compose_rotation(-lasX, -lasY, 0);
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 = ?
            b1 = ?
            c1 = ?
            a2 = ?
            b2 = ?
            c2 = ?

            Y = ?
            X = ?

            y(a)= ?
            x(a)= ?
            a=a+1;
        end
    end
end
end
```

Наконец, давайте введем наши калибровочные параметры и выведем результаты карты глубины на график:

```
[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;
```

Теперь мы можем сравнить наше входное изображение с выходным. Как видите, выходное изображение содержит только информацию о расстоянии пересечения лазерной полосы с препятствиями:

Для сравнения рассмотрим неоткалиброванный случай, как будет выглядеть карта, если установить углы, связанные с ориентацией плоскости камеры и лазера, равными нулю:

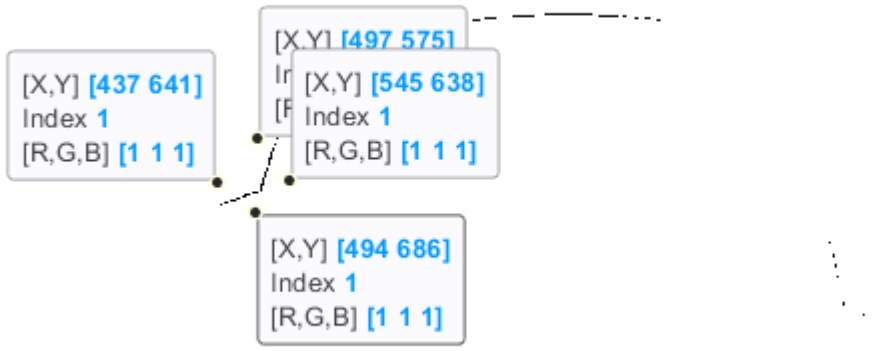
```
[x,y] = mapping(img,0, 0, 0, 0_,0_,950,ocam_model);
robot_x = 0; % Robot position
robot_y = 0; % Robot position
% Finally figure:
figure;
scatter(x,y,5,'filled'); % Laser intersections
hold on;
plot(robot_x,-robot_y,'r*'); % Robot location
grid on;
```

Из последнего рисунка видно, что даже небольшие отклонения могут привести к неправильным измерениям, поэтому важен процесс калибровки.

Проверка результатов

После построения карты появляется возможность сравнивать расстояния из эксперимента с реальными. Для этого можно использовать график и курсор мыши. Однако лучше брать среднее значение среди всех лазерных точек, принадлежащих желаемому препятствию, потому что это поможет вам устранить некоторые помехи. Рассмотрим функцию для второго случая.

Прежде всего нам нужно найти интересующие нас координаты пикселей, чтобы найти лазерные точки, принадлежащие этой области, см. рисунок ниже:



После того, как координаты были определены, мы можем написать функцию, которая проецирует пиксельные координаты в координаты пространства (`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);
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 = ?
            b1 = ?
            c1 = ?
            a2 = ?
            b2 = ?
            c2 = ?

            Y = ?
            X = ?

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

И, наконец, мы возьмем среднее значение каждой области

```
% Black Cube right
i=[497;575]; % working image region - column
j=[545;638]; % working image region - row
[Cube_L] = cube_dist(img,i,j,camX,camY,camZ,lasX_,lasY_,z_las_dist,ocam_model);
Cube_L = mean(Cube_L(:,1))-240/2;
% Black Cube bottom
i=[437;494]; % working image region - column
j=[641;686]; % working image region - row
[Cube_Up] =
cube_dist(img,i,j,camX,camY,camZ,lasX_,lasY_,z_las_dist,ocam_model);
Cube_Up = mean(Cube_Up(:,2))+240/2;
```

Теперь мы можем сравнить переменные:

	Черный куб - X, мм	Черный куб - Y, мм
Реальные значения	-1140,90	921,00
Экспериментальные значения	-1135,00	917,99
Погрешность	5,90	3,01

ИСПОЛЬЗОВАННАЯ ЛИТЕРАТУРА:

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.