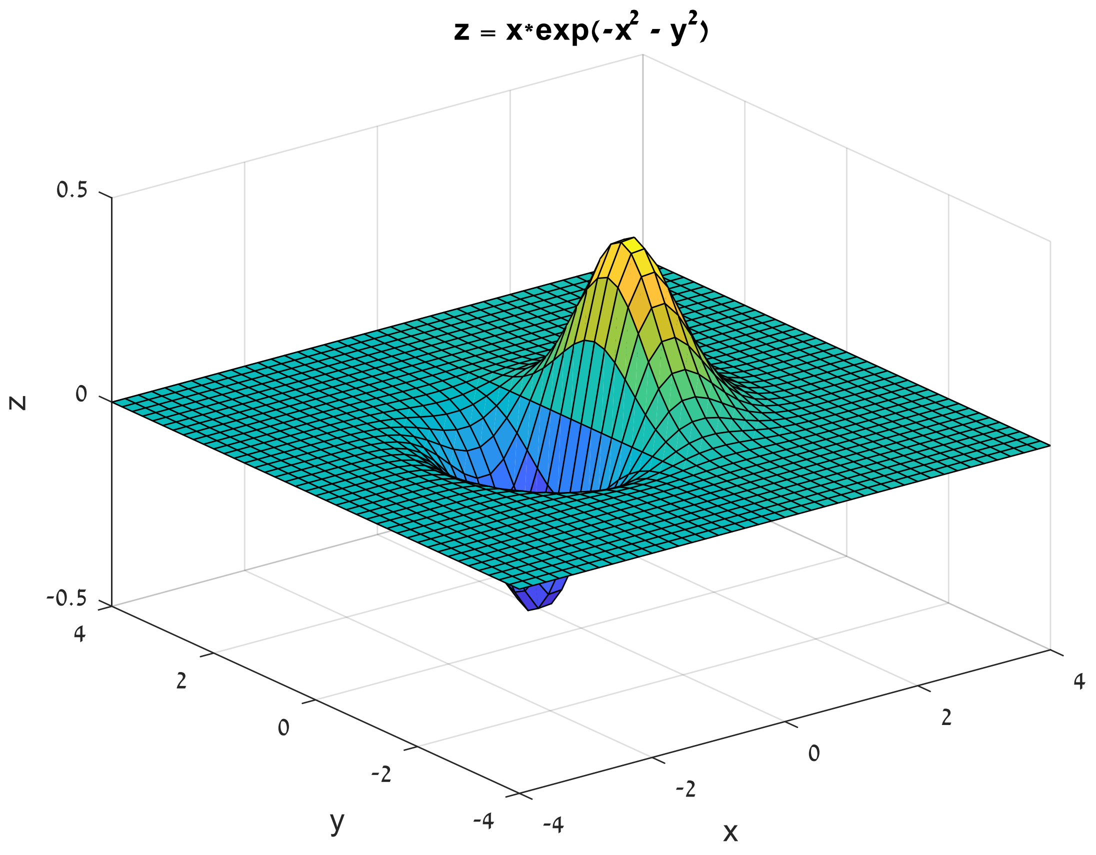
Computer Vision 236873 - HW1

Aram Gasparian, 310410865, aram89g@gmail.com
Hana Matatov, 203608302, hanama888@gmail.com

Q2 Photometry:

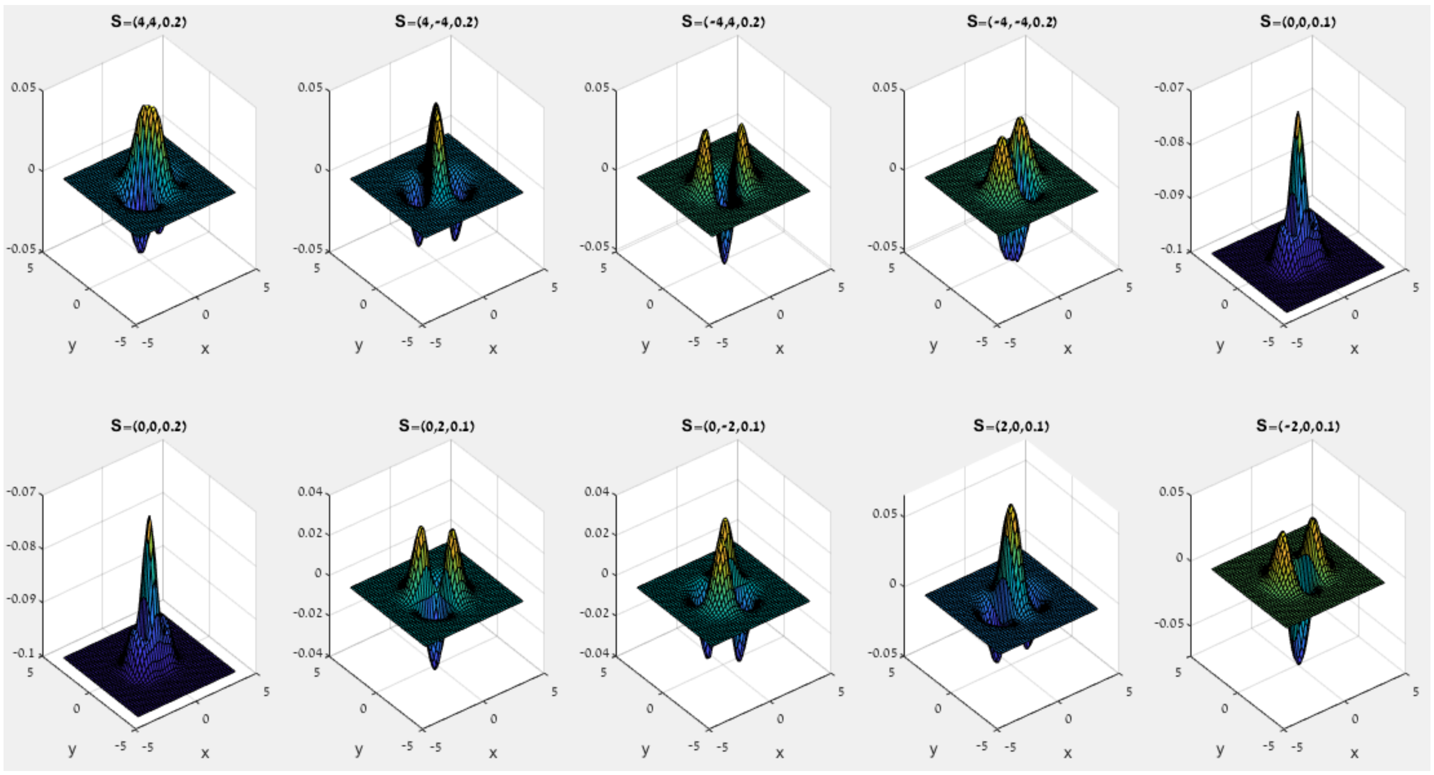
a. The following surface 3D in Matlab:

$$z = x \cdot \exp\left(-x^2 - y^2\right)$$



Explanation:
We defined X and Y axes as -4 to 4 with intervals of 0.2. We choose symmetric axes because the surface Z is not flat in the center of the grid. We choose intervals of 0.2 in order to have a smoother surface.

b. The irradiance I for 10 different illumination vectors $\hat{s}$:



<u>Explanation:</u>
We defined 10 different coordinates for s, each is trying to capture a different direction point:

X	Y	Z
4	4	0.2
4	-4	0.2
-4	4	0.2
-4	-4	0.2
0	0	0.1
0	0	0.2
0	2	0.1
0	-2	0.1
2	0	0.1
-2	0	0.1

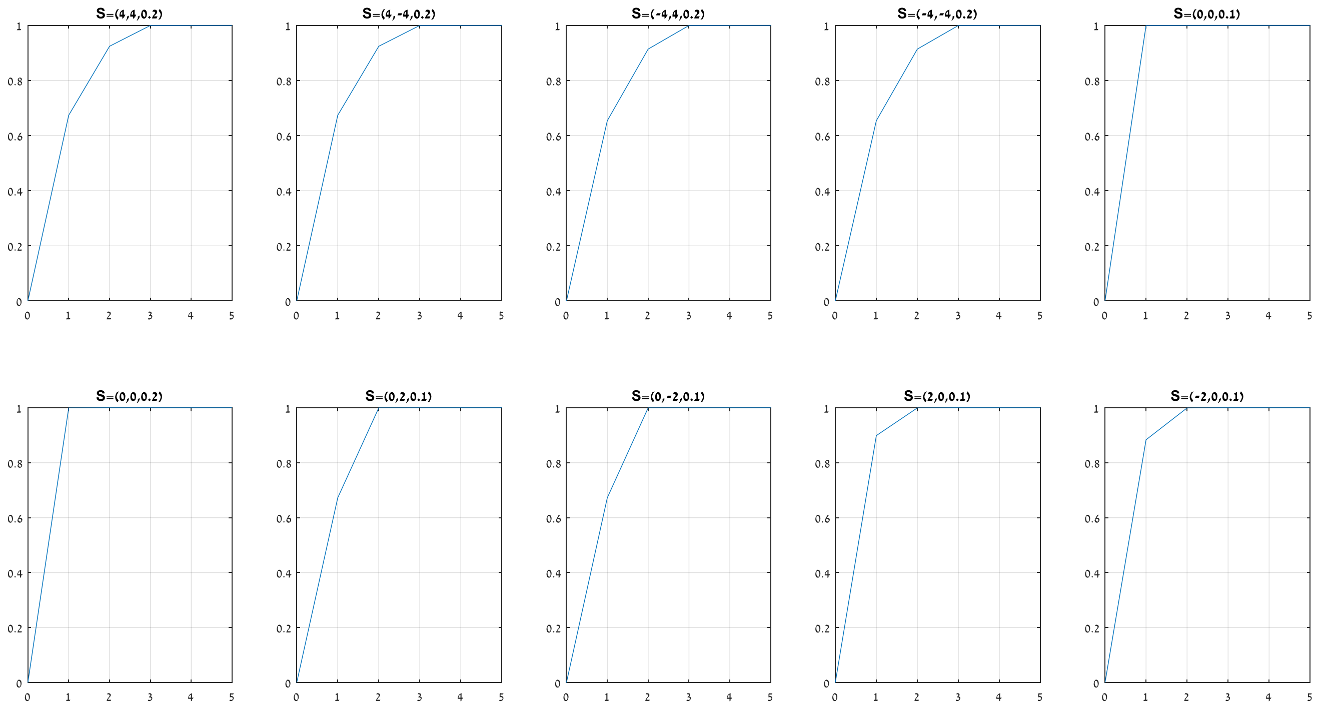
Then, we normalized the size of each source s to be 1 (a unit vector $\hat{s}$).

Finally, we calculated the formula for irradiance of Lambertian surface as:

$$I = \rho \hat{n}^T \cdot \hat{s}$$

When $\rho = 0.1$ as defined, $\hat{n}^T$ is the normal of each pixel on the surface and $\hat{s}$ from before.
We printed the 10 different surfaces, each represent the surface I as a function of X and Y, differ according to the s coordinates.

c. SVD factorization of each of the images for finding their effective dimension:



<u>Explanation:</u>
We calculated the SVD factorization of each of the images by using SVD factorization for each of the matrixes I (irradiance when using different source vectors s).
Then, we looked only on the middle-returned matrix which is the singular values matrix – the diagonal matrix.
The values on the diagonal of this matrix are the singular values of I.
The number of the non-zero singular values is the rank of the matrix I.
Also, the non-zero singular values are square roots of the non-zero eigenvalues of $I^T I$ and of II^T.
Therefore, we calculated the sum of squares of K non-zero singular values to get those eigenvalues.
Then, we calculated the result for every K divided by the value for all N:

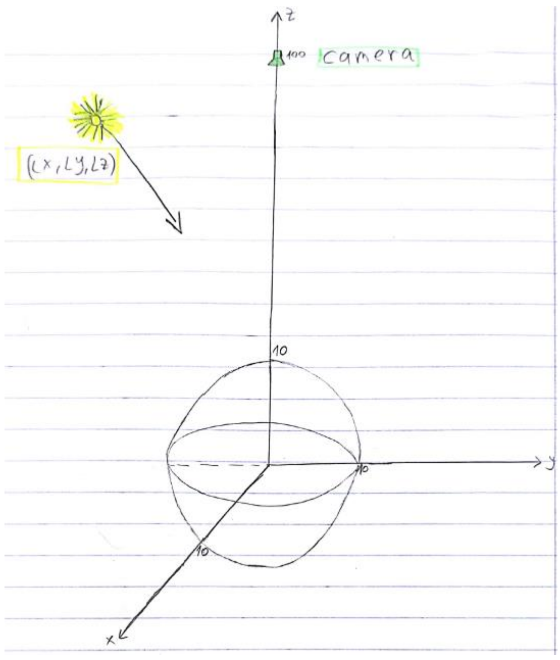
$$\frac{\sum_{i=0}^{K} \lambda_i^{\ 2}}{\sum_{i=0}^{N} \lambda_i^{\ 2}}$$

This is a value between 0 (when i=0) to 1 (when the numerator and denominator are equal).
We saw that this value gets to 1 when K<5, therefore we plot the graphs for $0 \le K \le 5$.
Results:
When S=(4,4,0.2) the effective dimension is 3.

When S=(4,-4,0.2) the effective dimension is 3.
When S=(0,0,0.2) the effective dimension is 1.
When S=(0,2,0.1) the effective dimension is 2.
And so on.

Q3:

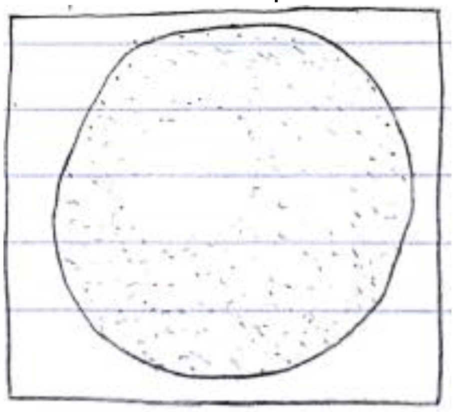
a. A schematic diagram showing the ball, the camera and the light source:



Explanation:

The ball is of radius 10, centered at the origin.
The ideal camera located at (0,0,100), above the ball - marked in green in the schematic diagram.
A point illumination source is located in some unknown place (Lx,Ly,Lz>0), far from the ball – assume it is in the point marked in yellow in the schematic diagram.

b. A schematic description of the image taken by the camera:



<u>Explanation:</u>

Since the illumination source is far from the ball, and it is Lambertian ball, we can assume that the direction of the illumination source is equal for every point on the ball, and that the illumination is approximately uniform on all points on the ball.
But, since the camera and the illumination source are both above the ball, in the image only the upper side of the ball will be seen, and it will look as a circle.
Because of the direction of light, the part of the upper-ball where the normal is in the same direction as the illumination source direction, will have more illumination (*this area looks brighter in the schematic description*).

c. An algorithm for finding the illumination source direction:
   1. Count the radius of the ball in the image (pixels units).
   2. Calculate the ratio between the radius of the ball in the world (centimeters units), to the radius of the ball in the image (pixels units) from step 1. It is possible since we know that the radius of the ball in the world is 10.
   3. Find the coordinates (x^*, y^*) of the brightest pixel of the image by:

      $$(x^*, y^*) = \underset{(x,y)\in image}{\arg\max}\, \tilde{I}(x, y)$$

      Where $\tilde{I}$ is the value of brightness on each point on the image.

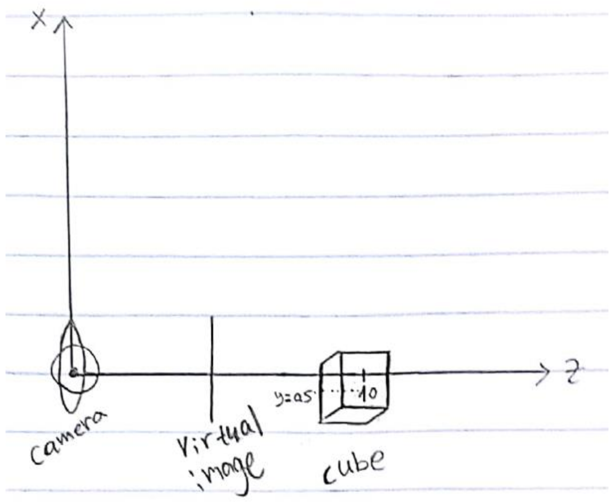
      We are using that $I = \rho \hat{n}^T \cdot \hat{s}$ for Lambertian ball and the approximation for far illumination sources. The brightest point is where $\hat{n}$ and $\hat{s}$ directions are the same (the vectors are parallel).
   4. Find $\left(x^{**}, y^{**}\right)$ in world's coordinates system by using the results of steps 2 and 3.
      We are using the distance between the center of the circle in the image to (x^*, y^*), as proportional to the distance between the center of the ball in the world to the projection of $\left(x^{**}, y^{**}\right)$ on XY surface (Z=0).

It relies on the fact that the camera's center is directly above ball's center.

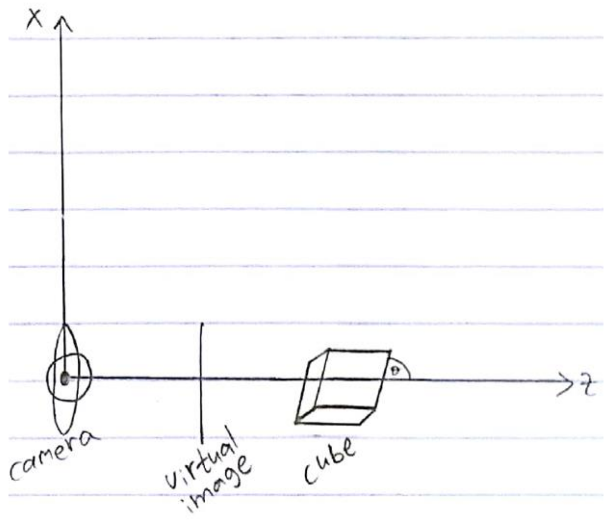
5. Find $\left(x^{**}, y^{**}, z^{**}\right)$ in world's coordinates system by using the result of step 4 and the ball's formula.
6. Find the normal to the ball surface at the point $\left(x^{**}, y^{**}, z^{**}\right)$.
7. Define the direction of the normal from step 6 as the direction of the illumination source.

<u>Q4:</u>

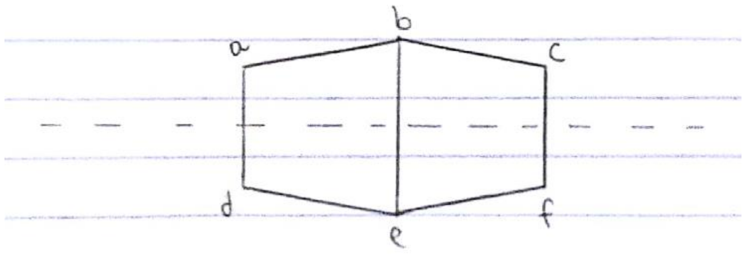
We can draw the axes so that axis y is coming to the viewer direction.
In this coordinates system the center of the cube located above the drawing (y=0.5):



After the rotation of the cube, which was defined as a rotation in the center of the cube over an axis which is parallel to axis y, one of the edges of the cube creates an angle θ with the z axis, and it looks:



When looking from the direction of the camera, the image plane which is parallel to XY plane looks:

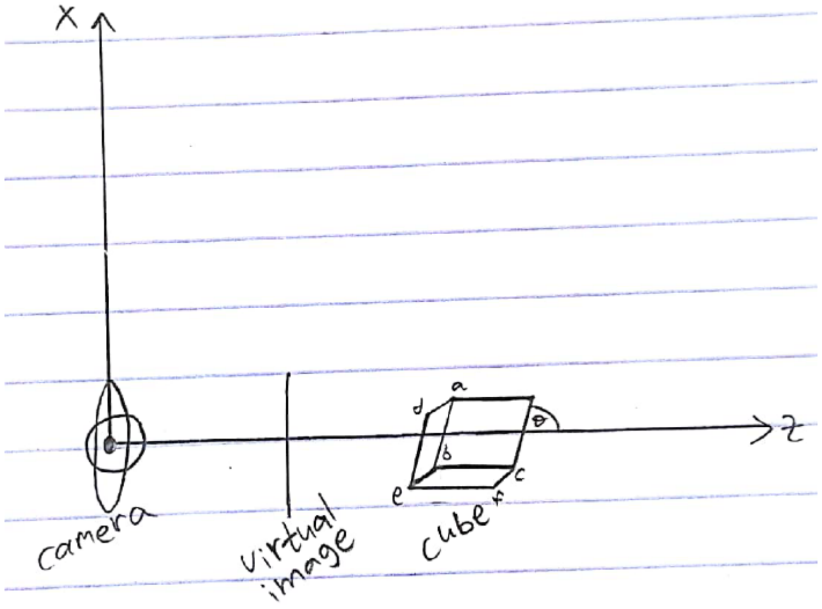


We have two locations of vanishing points:

1. Edge a-b is parallel to edge d-e and they have the vanishing point $\left(x_{im_1}, y_{im_1}\right)$.
2. Edge b-c is parallel to edge e-f and they have the vanishing point $\left(x_{im_2}, y_{im_2}\right)$.

Note that the edges a-d, b-e, c-f remain parallel on the image plane since they are parallel to the rotation axis (which defined as parallel to axis y), therefore we won't have an additional vanishing point.

In order to do the calculations, let's mark those edges on the rotated cube:



Now, let's use the rotation formula:

$$\begin{pmatrix} z' \\ x' \end{pmatrix} = \begin{pmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{pmatrix} \begin{pmatrix} z \\ x \end{pmatrix}$$

The image is in the XY plane.

Y coordinates didn’t change during the rotation since the rotation axis is parallel to y.

Therefore, we are interested in x’ and z’ for each of the relevant 6 points on the cube:

$$a_{x'} = -a_z \cdot \sin(\theta) + a_x \cdot \cos(\theta) = -9.5 \cdot \sin(\theta) + 0.5 \cdot \cos(\theta)$$
$$b_{x'} = -b_z \cdot \sin(\theta) + b_x \cdot \cos(\theta) = -9.5 \cdot \sin(\theta) - 0.5 \cdot \cos(\theta)$$
$$c_{x'} = -c_z \cdot \sin(\theta) + c_x \cdot \cos(\theta) = -10.5 \cdot \sin(\theta) - 0.5 \cdot \cos(\theta)$$
$$d_{x'} = -d_z \cdot \sin(\theta) + d_x \cdot \cos(\theta) = -9.5 \cdot \sin(\theta) + 0.5 \cdot \cos(\theta)$$
$$e_{x'} = -e_z \cdot \sin(\theta) + e_x \cdot \cos(\theta) = -9.5 \cdot \sin(\theta) - 0.5 \cdot \cos(\theta)$$
$$f_{x'} = -f_z \cdot \sin(\theta) + f_x \cdot \cos(\theta) = -10.5 \cdot \sin(\theta) - 0.5 \cdot \cos(\theta)$$

$$a_{z'} = a_z \cdot \cos(\theta) + a_x \cdot \sin(\theta) = 9.5 \cdot \cos(\theta) + 0.5 \cdot \sin(\theta)$$
$$b_{z'} = b_z \cdot \cos(\theta) + b_x \cdot \sin(\theta) = 9.5 \cdot \cos(\theta) - 0.5 \cdot \sin(\theta)$$
$$c_{z'} = c_z \cdot \cos(\theta) + c_x \cdot \sin(\theta) = 10.5 \cdot \cos(\theta) - 0.5 \cdot \sin(\theta)$$
$$d_{z'} = d_z \cdot \cos(\theta) + d_x \cdot \sin(\theta) = 9.5 \cdot \cos(\theta) + 0.5 \cdot \sin(\theta)$$
$$e_{z'} = e_z \cdot \cos(\theta) + e_x \cdot \sin(\theta) = 9.5 \cdot \cos(\theta) - 0.5 \cdot \sin(\theta)$$
$$f_{z'} = f_z \cdot \cos(\theta) + f_x \cdot \sin(\theta) = 10.5 \cdot \cos(\theta) - 0.5 \cdot \sin(\theta)$$

Now, we calculate the 4 edges:

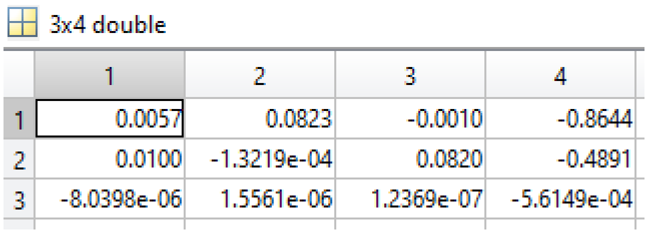
$$\left(a \leftrightarrow b\right)_x : a_{x'} - b_{x'} = -9.5 \cdot \sin(\theta) + 0.5 \cdot \cos(\theta) + 9.5 \cdot \sin(\theta) + 0.5 \cdot \cos(\theta) = \cos(\theta)$$
$$\left(a \leftrightarrow b\right)_z : a_{z'} - b_{z'} = 9.5 \cdot \cos(\theta) + 0.5 \cdot \sin(\theta) - 9.5 \cdot \cos(\theta) + 0.5 \cdot \sin(\theta) = \sin(\theta)$$
$$\left(a \leftrightarrow b\right)_y : a_{y'} - b_{y'} = 1 - 1 = 0$$
$$\left(d \leftrightarrow e\right)_x : d_{x'} - e_{x'} = -9.5 \cdot \sin(\theta) + 0.5 \cdot \cos(\theta) + 9.5 \cdot \sin(\theta) + 0.5 \cdot \cos(\theta) = \cos(\theta)$$
$$\left(d \leftrightarrow e\right)_z : d_{z'} - e_{z'} = 9.5 \cdot \cos(\theta) + 0.5 \cdot \sin(\theta) - 9.5 \cdot \cos(\theta) + 0.5 \cdot \sin(\theta) = \sin(\theta)$$
$$\left(d \leftrightarrow e\right)_y : d_{y'} - e_{y'} = 0 - 0 = 0$$
$$\left(b \leftrightarrow c\right)_x : b_{x'} - c_{x'} = -9.5 \cdot \sin(\theta) - 0.5 \cdot \cos(\theta) + 10.5 \cdot \sin(\theta) + 0.5 \cdot \cos(\theta) = \sin(\theta)$$
$$\left(b \leftrightarrow c\right)_z : b_{z'} - c_{z'} = 9.5 \cdot \cos(\theta) - 0.5 \cdot \sin(\theta) - 10.5 \cdot \cos(\theta) + 0.5 \cdot \sin(\theta) = -\cos(\theta)$$
$$\left(b \leftrightarrow c\right)_y : b_{y'} - c_{y'} = 1 - 1 = 0$$
$$\left(e \leftrightarrow f\right)_x : e_{x'} - f_{x'} = -9.5 \cdot \sin(\theta) - 0.5 \cdot \cos(\theta) + 10.5 \cdot \sin(\theta) + 0.5 \cdot \cos(\theta) = \sin(\theta)$$
$$\left(e \leftrightarrow f\right)_z : e_{z'} - f_{z'} = 9.5 \cdot \cos(\theta) - 0.5 \cdot \sin(\theta) - 10.5 \cdot \cos(\theta) + 0.5 \cdot \sin(\theta) = -\cos(\theta)$$
$$\left(e \leftrightarrow f\right)_y : e_{y'} - f_{y'} = 0 - 0 = 0$$

Therefore, the 2 vanishing points are:

1. Edge a-b is parallel to edge d-e and they have the vanishing point (for $t \to \infty$):
$$\left(x_{im_1}, y_{im_1}\right) = \left(-f \cdot \frac{v_x}{v_z}, -f\frac{v_y}{v_z}\right) = \left(-1\cdot\frac{\cos(\theta)}{\sin(\theta)}, -1\cdot\frac{0}{\sin(\theta)}\right) = \left(\frac{-\cos(\theta)}{-\sin(\theta)}, 0\right)$$
2. Edge b-c is parallel to edge e-f and they have the vanishing point (for $t \to \infty$):
$$\left(x_{im_2}, y_{im_2}\right) = \left(-f \cdot \frac{v_x}{v_z}, -f\frac{v_y}{v_z}\right) = \left(-1\cdot\frac{\sin(\theta)}{-\cos(\theta)}, -1\cdot\frac{0}{-\cos(\theta)}\right) = \left(\frac{-\sin(\theta)}{\cos(\theta)}, 0\right)$$

Q5 – Camera Calibration

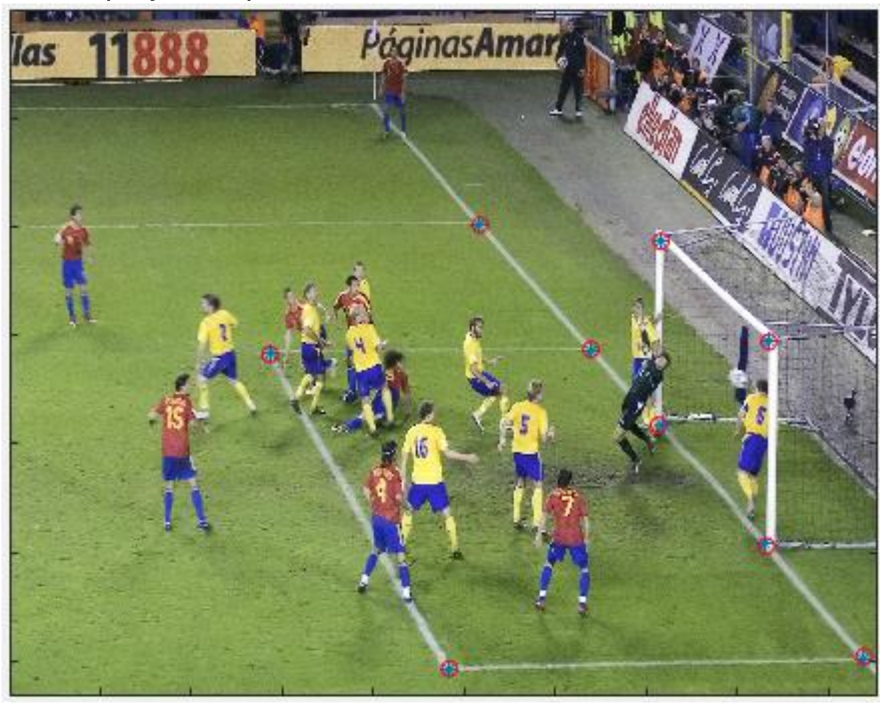
a. The camera matrix P we got using the DLT method is:



3x4 double

	1	2	3	4
1	0.0057	0.0823	-0.0010	-0.8644
2	0.0100	-1.3219e-04	0.0820	-0.4891
3	-8.0398e-06	1.5561e-06	1.2369e-07	-5.6149e-04

b. The re-projected points are the red circles:



As we can see they correspond with the image points given so we can be sure that the camera matrix P we got is good.
The error measure we define is:

$\hat{x} = Px,\ error = \frac{1}{n}\sum_{i=1}^{n}\|x_i - \hat{x}_i\|_2$, it measures the mean distance between the image points to the re-projected points we calculated in pixels.

$$error = 3.5229\ [pixels]$$

c.

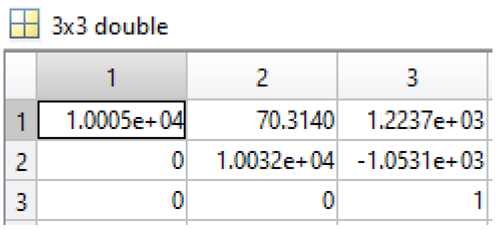
1) The goal of the rq() function is to deconstruct P to KR where K is an upper triangle matrix and R is a unitary matrix.
The difference with the matlab qr() function is that the qr() deconstructs to QR where Q is unitary matrix and R is an upper triangle, and this is the reason we didn't use qr().

2)
```
[Q,R] = qr(flipud(M)');
% QR decomposition of a flipped up down M'
R = flipud(R');
R = fliplr(R);

Q = Q';
Q = flipud(Q);

% force the diagonal to be positive
T = diag(sign(diag(R)));
R = R * T;
Q = T * Q;
```
Any full rank matrix can be decomposed into the product of an upper triangular matrix and an orthogonal matrix by using RQ-decomposition, but matlab doesn't have this function in its libraries so we need to implement it.
The first part implements an RQ-decomposition but it's not unique, so we force the diagonal to be positive.

d. The K matrix we got needs to be normalized because the solution of the optimization problem is correct until multiplication by a factor.


3x3 double

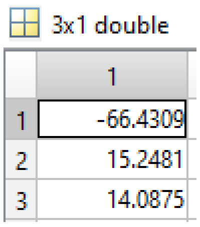
	1	2	3
1	1.0005e+04	70.3140	1.2237e+03
2	0	1.0032e+04	-1.0531e+03
3	0	0	1

The skew is 70.
The px and py are positive and negative respectively, we expected it to be the opposite. The reason it is like this is because the x-axis of the image and the camera are opposing each other, the same about the y-axis.
The focal lengths fx and fy are fairly the same but we expected the to be negated from the same reason.

- A link that explains the reasons for the results and what happened. http://ksimek.github.io/2012/08/14/decompose/?fbclid=IwAR26woRzfsramAlJpHlhS1AS7SNlkyBA96i15LLIpB6otD-h7DnuLcLRVZo

e. The orientation of the camera that we get from rq() isn't good (also the determinate is -1) it makes the camera face the other direction, the reason for this is the enforcement of the positive diagonal made in rq(). If we negate the R we got from rq() we get the camera orientation we expected, x-axis of the image is the same as x-axis of the camera, the same for y-axis and the camera faces the positive z coordinate.

f. The camera location c is:

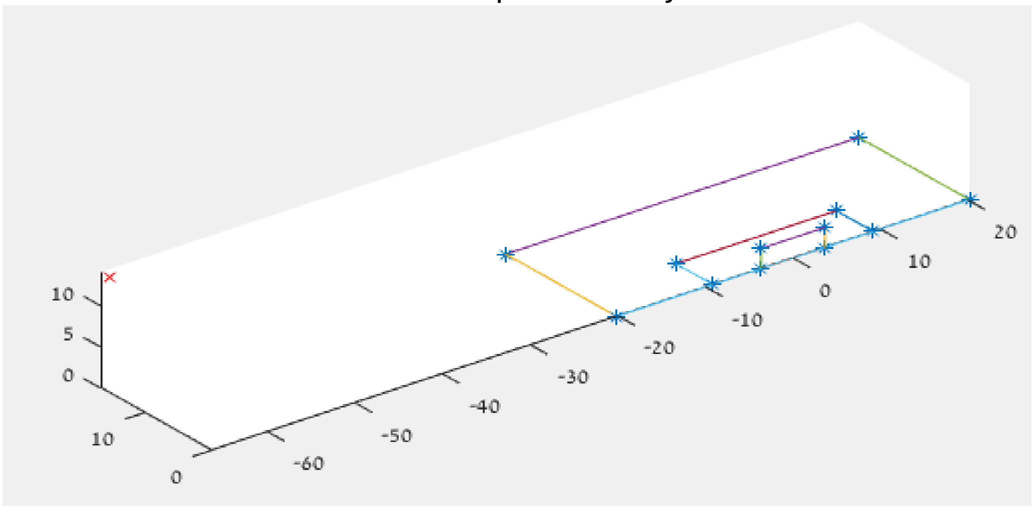


3x1 double

	1
1	-66.4309
2	15.2481
3	14.0875

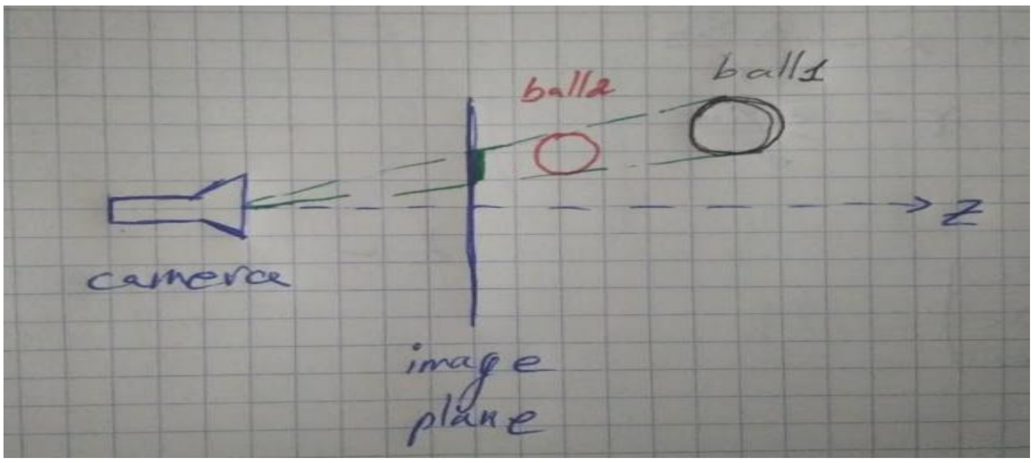
The location is reasonable because we expect it to be left of the goal frame (negative x and positive y), and above the ground (positive z).

g. The 3d location of the camera is represented by the red x:



And we can see now the result from the previous section is reasonable.

h. We cannot determine the depth of the ball because we don't know it's real world size, this is the missing piece we need to be able to calculate.



As depicted in the image ball1 and ball2 have different distance from the camera and are in different sizes but the size on the image plane is the same so we cannot determine the ball distance from camera using only information from the image.

<u>**Q6 – Morphologic Operations**</u>

<u>Note:</u>

We are solving the question as if initially only the train set is known. So, we are doing an “offline” process to extract each image descriptors (by using basic and morphologic operations).

Then, when the test image is given (which is unknown initially), we are processing a similar “online” process to extract it descriptors and compare them with those of each image resulted from the “offline” step.

<u>Assumption:</u>

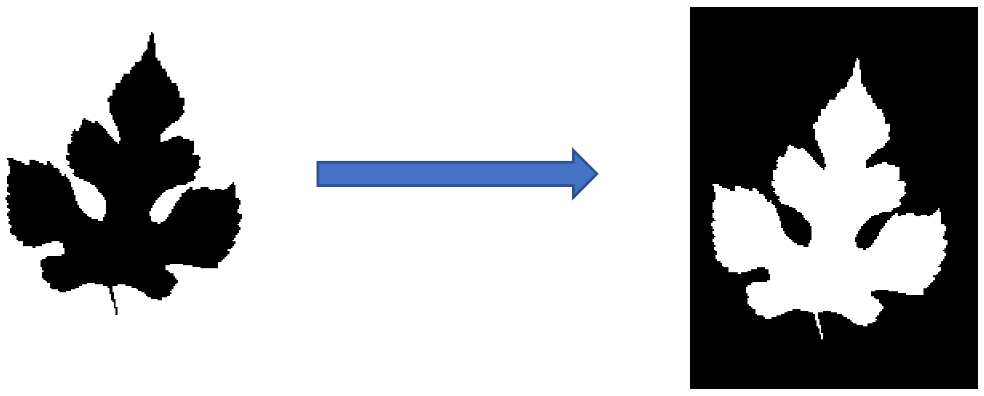
The image of the leaf in the images is in ratio to the leaf real size and not dependent on the image size (we do this because the images are in different sizes).

Algorithm:

1) Load train images and find the widest and highest dimensions.

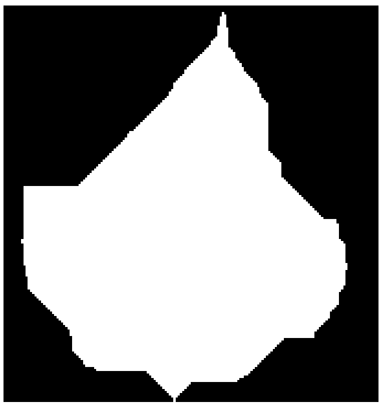
2) Get gray images for all RGB images.
   For example, we are demonstrating steps 2-5 with the image of leaf 1:

3) Binarize all images with a threshold of T = 200/255 and then do logical NOT on the image.

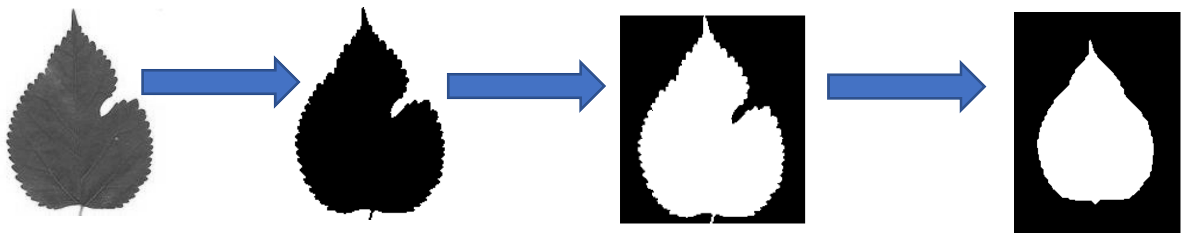


4) Pad the image with zeros (black pixels) from the sides and up and down to fit the maximal dimensions found in step 1.

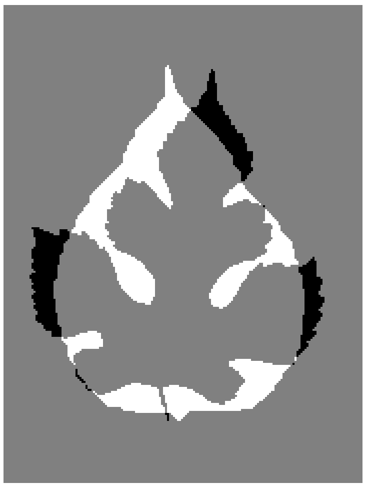
5) Perform morphological **closing** on the target image with a disk structure element in order to remove small holes and fill gaps that are created by imperfections.
   We choosed the closing morphological operator since it is doing dilation and then erosion, meaning it removes the holes without changing the size of the element. This is the reason why we didn't use dilation or erosion instead of closing.



6) **When the test image is arriving**, do the same steps for it.
   For example, for the provided test image the visual progress steps are:



7) Perform subtraction of a training image from the test image.
   The image values are: white: 1, grey: 0, black: (-1).
   - for every grey pixel inside the test image contour we add 1 to the score.
   - For every black pixel outside the test image contour we subside 1 from the score.
   - For every white pixel inside the test image contour we subside 1 from the score.
   - We normalize the score by the number of white pixels in the image from section 6.
   - If the score is negative assign 0 for the score.
   - Do this for every training image.



8) The most matching training image to the test image is the one with the highest score.

Note: in order to make the algorithm have a "softer" decision we can change the scoring options so that for every white pixel inside the test image contour we don't change the score.

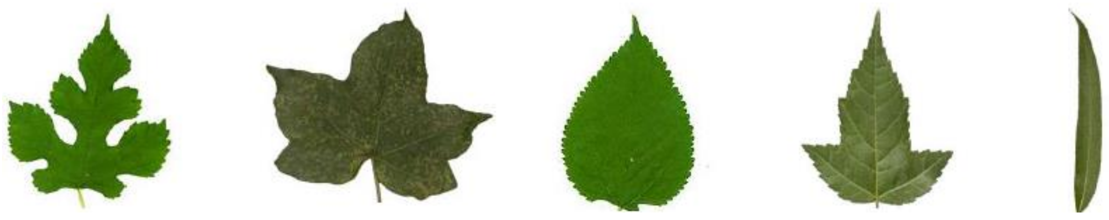
The reason behind the scoring algorithm is that the most matching image will fill most of the contour of the test image and by, so it will get the highest score, and the rest of the image will not. We account for the size of the leaf by giving a lower score to leaf's that are bigger or smaller from the test leaf.

The results:

	Leaf1	Leaf2	Leaf3	Leaf4	Leaf5
Score	0.3723	0.3457	0.7661	0.3229	0

If we use the softer approach:

	Leaf1	Leaf2	Leaf3	Leaf4	Leaf5
Score	0.6325	0.5765	0.8690	0.5516	0.2370



We can see that there is significant difference between the results, meaning that the score of leaf3 is much higher than the other scores.

The second highest score is of leaf1 (the difference in the scores is more significant when using the softer approach).

These are indeed the results we accepted for the given test image (leaf3 is almost identical to the test image, except of the hole).

It works well because after using closing morphological operator, the test image, leaf3 and leaf1 look similar to each other – after the removal of the holes they all have similar external frames.