I. Calibration

A. Projection matrix computation

The projection matrix was computed using the the DLT method:

1. The A matrix was built, which concentrated all the corresponding points. We have 9 points in total, which is more than required (6 pairs). Each pair can retrieve 2 unknown variables, and the camera matrix P is of size [3X4], meaning we are looking for 12 variables. The equation which has to hold is:

$$\underline{AP} = 0$$

   thus the problem turns into the minimization problem:

$$\min_{\underline{P}} ||AP||_2^2; s.t. ||P||_2^2 = 1$$

2. Using the SVD, the A matrix was converted into [U, D, V]. P is the minimal right eigenvector. Reordering the P we obtain the original projection matrix P:

$$\begin{bmatrix} 0.0057 & 0.0823 & -0.0010 & -0.8644 \\ 0.0100 & -0.0001 & 0.0820 & -0.4891 \\ -0.0000 & 0.0000 & 0.0000 & -0.0006 \end{bmatrix} \quad (1)$$

B. Points reprojection, error measurement

All the points from the 3D world coordinated were reprojected using the reprojection matrix found. To estimate the error it was decided to use the MSE method. The following table shows the reprojected coordinates for each point, and the MSE. the units of the MSE is pixels. By being around 3.5 pixels, we can assume that the error comes from the inacuracy (and inconsistency of the accuracy) of picking the points from the image manually.

Original coordinates		Reprojected coordinates		
X	Y	X	Y	ERROR
1661.000	981.800	1663.902	988.118	6.953
1436.600	763.700	1427.416	765.563	9.371
1432.300	427.400	1432.452	427.193	0.257
1669.800	612.200	1669.631	612.426	0.282
1876.500	1196.100	1879.132	1190.668	6.036
966.000	1214.600	967.713	1213.446	2.065
574.500	634.300	573.188	635.183	1.581
1278.300	627.000	1278.563	625.479	1.544
NaN	NaN	0.000	0.000	0.000
NaN	NaN	0.000	0.000	0.000
NaN	NaN	0.000	0.000	0.000
1032.000	397.800	1035.437	396.676	3.616
			MEAN:	3.523

Figure 1: Errors of the 3D points reprojection

C. R,K matrix reconstruction

The matrixes R, K were reconstructed using the given function.
In this case we use the RQ decomposition, and not the QR decomposition, because we require the opposite order of the matrixes on the output. QR decomposes matrix into an orthogonal matrix and upper triangular matrix. We require the reverse order, since the K matrix, which stands first, is upper triangular, and R is an orthogonal rotation matrix:

$$P = K[R|t] = [[KR]_{3X3}|[Kt]_{3X1}]$$

Why this works:
...

D. Camera characteristics

To answer this question, we normalize the camera characteristics matrix K. Knowing that the item [3,3] =1, we divide the whole matrix my this value. K obtained:

$$K = \begin{bmatrix} f_x & s & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} = 1e^{+4} \begin{bmatrix} 1.0005 & 0.0070 & 0.1224 \\ 0 & 1.0032 & -0.1053 \\ 0 & 0 & 0.0001 \end{bmatrix}$$

We can see that the matrix is indeed reasonable:

1. The focal length values here are in the units of $[pixels]$. We can see that they are almost similar, although they differ a bit, which is okay due to inperfect camera (and other factors).

2. To obtain the focal length values in $[mm]$ as usually in cameras, we need to know the pixel size of a camera. Lets take something found over internet, some real values: "Industrial cameras usually use 1/3" sensors in case of resolutions of 640 x 480 pixels" [1]. Using the image provided in this source, we obtain the $[\frac{pixels}{mm}]$ value:

$$m_x = \frac{640}{4.8} = \frac{480}{3.6} = 133.3[\frac{pixels}{mm}]$$

. Using this on our case, finding the focal length in [mm] gives:

$$f_{x_{mm}} = \frac{f_x}{m_x} = \frac{10005}{133.3} = 75.1[mm]$$

$$f_{y_{mm}} = \frac{f_y}{m_x} = \frac{10032}{133.3} = 75.2[mm]$$

Which is quiet typical.

3. We can see that the skew parameter s is small relatively to f_x and f_y, which is reasonable.

4. x_0 and y_0 indicate where the Principal Point Offset is located.

E. Orientation of the camera

We perform the sanity check. We know that in the Camera coordinates, the Y direction approximately lies on the Z direction of the World coordinates. R matrix rotates from World to Camera. Thus, we obtain:

$$\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} 0.1897 & 0.9816 & -0.0216 \\ 0.0189 & 0.0183 & 0.9997 \\ -0.9817 & 0.1900 & 0.0151 \end{bmatrix} = \begin{bmatrix} -0.0216 \\ 0.9997 \\ 0.0151 \end{bmatrix}$$

Thus, we indeed see that the rotation matrix is reasonable and that Z direction in World changed to Y direction in Camera coordinate system.

F. Translation vector

In the calculated P matrix, the derived translation vector is indeed in the Camera coordinates. We translate it into the the World coordinates using the given equation:

$$t_C = -Rc$$

, where c is the translation in the World space. The derived c is:

[1] https://www.vision-doctor.com/en/camera-technology-basics/sensor-and-pixel-sizes.html

$$c = -R^T t = \begin{bmatrix} -66.66 \\ 15.3 \\ 14.13 \end{bmatrix} [m]$$

Everything is in meters. And indeed, we need apply this transformation to move from Camera Origin to the World Origin in the world coordinates.

G. Camera plotting

The camera location is being plotted onto the existing Figure. We have used the same translation vector that was found in the previous subsection, this is actually the camera location.

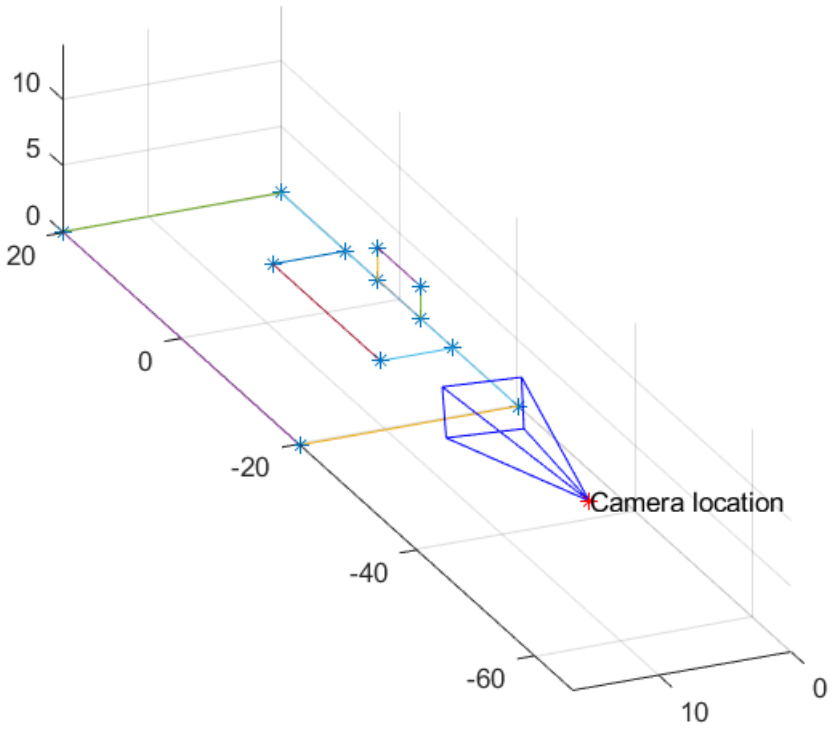


Figure 2: Camera location and camera frame

H. Goal detection

Using **only** the above information, it is impossible to determine whether the ball has crossed the line. There is ambiguity the distance of the ball from the camera. We can only project the vector, on which the ball can be placed in the real world:

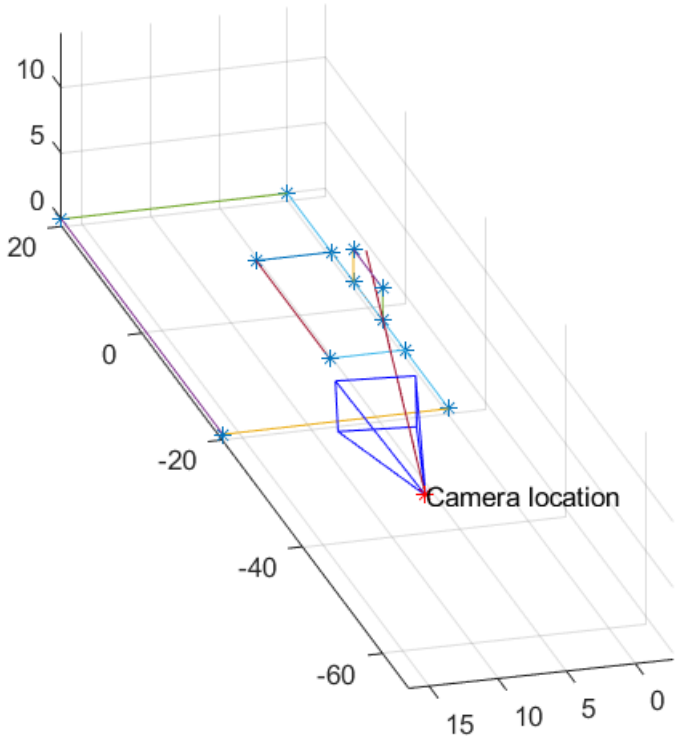


Figure 3: Projection of the vector where the ball is placed

We need aditional information to determine the distance of the ball. This information can be the size of the ball, to determine at which distance the ball will take indeed as many pixels as it takes in the picture. This is out of the scope of this question, thus will not be done here.

II. Depth with Stereo

A. Disparity map explanation

Disparity (x-x') map shows the depth of the image. The depth of a point is inversely proportional to the difference in the distance of 2 corresponding image pixels. The farther the object - the less movement it undergoes if the camera moves. Using the dislocation of each pixel relatively to its previous location, we build the disparity map.

B. Window matching

The function was implemented - it can be found in 'calc_SSD.m'. At first, the blockproc function was used. But the calculation times were too high, so it was replaced by another algorithm.

C. Finding the corresponding pixels

Rectification of the images reduces the number of the unknown parameters in the Excentric (E) matrix. We currently have no more rotation, only the translation of the camera. This implies that the corresponding points have the same y coordinate (vertical / row number).
The function was written called 'find_matching_pixels', which does all of the mentioned in the question. It receives 2 images, an array of points in the first image, and the template size. It returns the array of points in the second image.

D. Constructing the disparity map

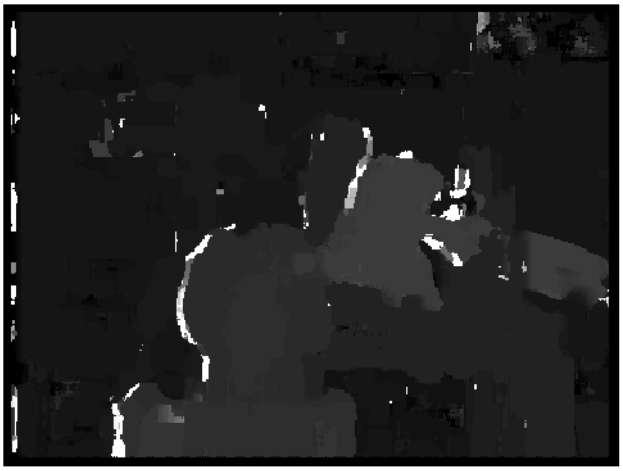
The function was written due to requirements. Called 'find_disparity'.

E. Experiments

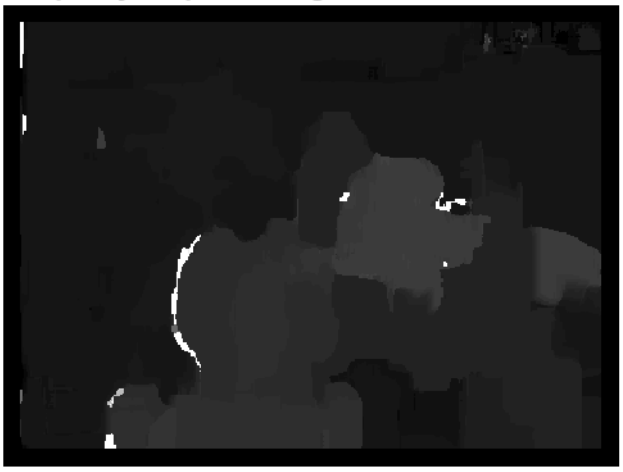
The function was executed on all of the examples, and all of the window sizes. Those are the results:

Disparity map for image03 window size =5

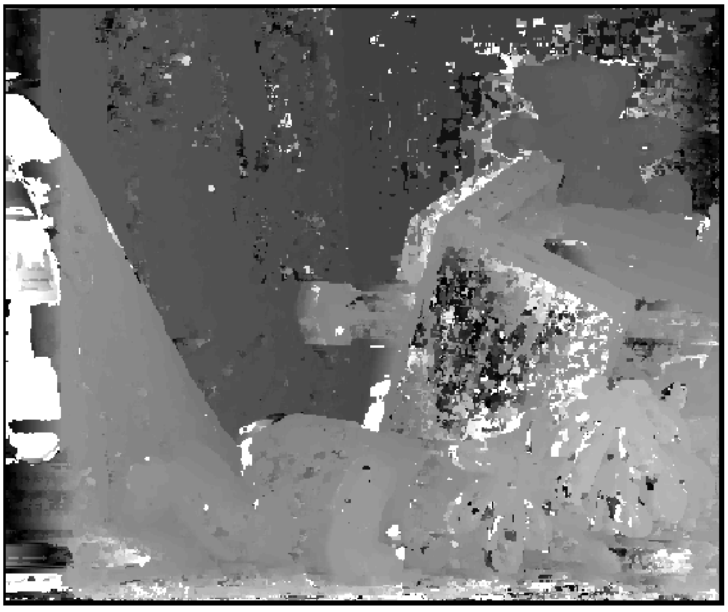
Disparity map for image03 window size =11



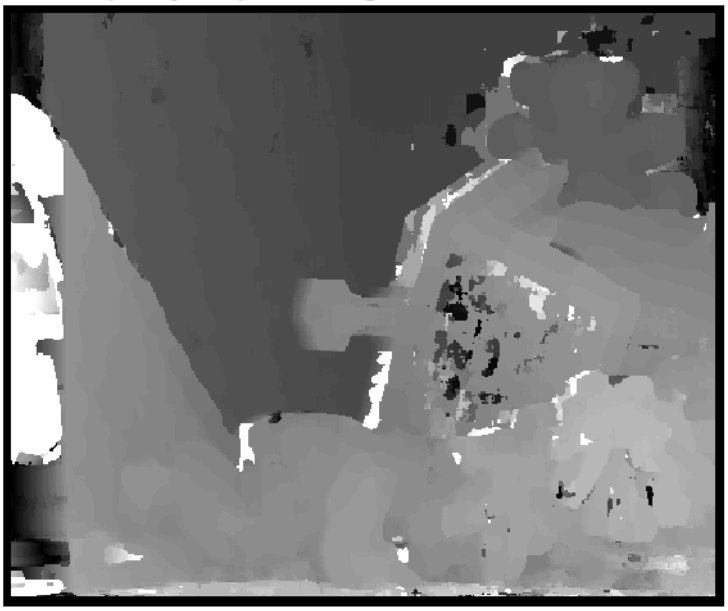
Disparity map for image03 window size =21



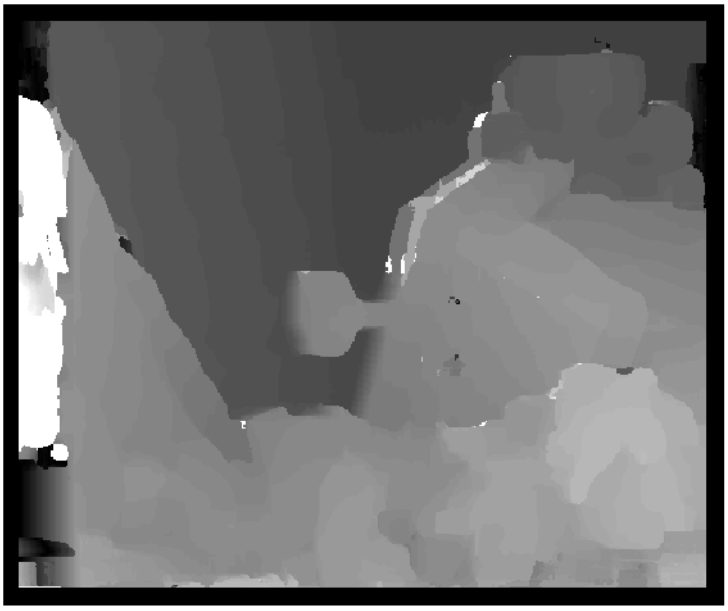
Disparity map for image04 window size =5



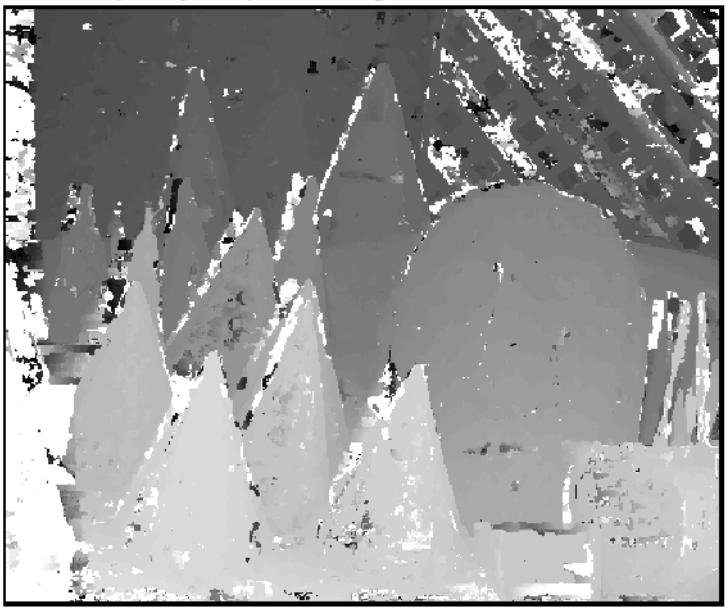
Disparity map for image04 window size =11



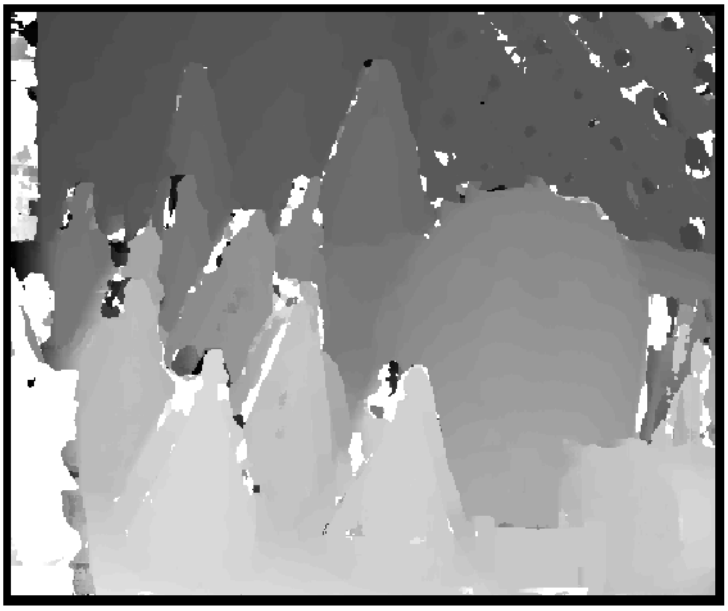
Disparity map for image04 window size =21

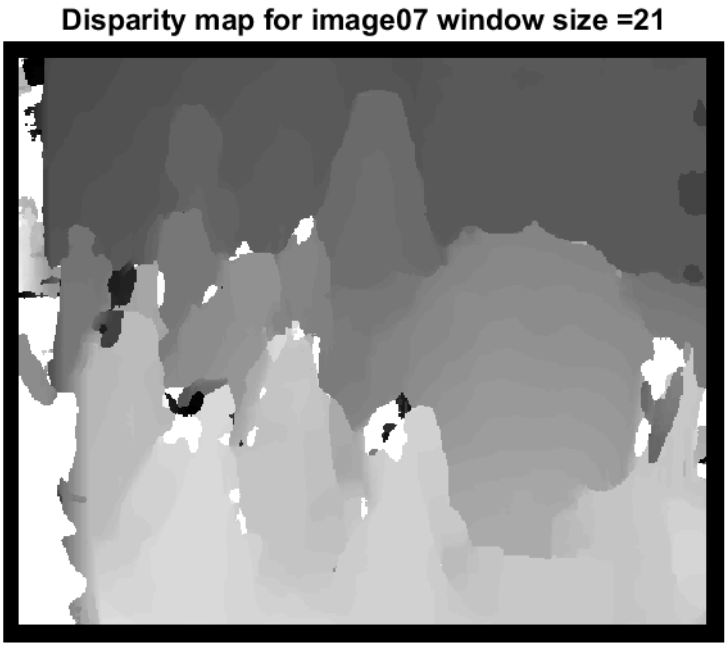


Disparity map for image07 window size =5



Disparity map for image07 window size =11





We can observe that for window size of 11 the performance is the best for visual experience. The window 5 is affected by noise too much. While the window size of 21 blurs the image too much. The problematic regions are the regions of uniform background, or those who change less in the X direction. The algorithm performs best on the edges, which are easier to detect using the SSD algorithm.

Finding the RMSE for all 3 images and all 3 window sizes gives:

Image number	Filter size	RMSE
03	5	9.79
	11	6.89
	21	4.57
04	5	12.19
	11	12.34
	21	13.7
07	5	12.6
	11	12.73
	21	14.37

From the RMSE calculations we can see that:

1. For the first image the smallest RMSE amount accounts for the largest filter size. But we have seen that the result is blurry. This happened because the maximum disparity value was set too high. The majority of disparities was calculated to be around 15 pixels in the histogram. So with the small filter size we have large presense of 'white' regions - those with disparity ¿ 60.

2. In both other images the RMSE value does not change much with the template size difference, although visually the results are different.

Out of pure interest, The Max Disparity value was decreased to 30 (which is 60/2) and the test has been redone on the Image 03. The results found are the following:

Image number	Filter size	RMSE
03	5	5.99
	11	4.33
	21	3.1

We can observe that we could substantially decrease the error, but still the errors proportion remained the same, and the error is smallest for the largest filter size, because the amount of pixels who got the maximum value is still the same (we just decreased their value in half).