COMPUTER VISION COURSE (2021/22)
Master in Electrical and Computer Engineering
Laboratory Assignment 4

3D Reconstruction

Due Date : Wednesday, June 1st, 2022, 23h59m

1. Learning Goals and Description

The capacity of retrieving an object from images taken from different positions, known as 3D reconstruction, is being used nowadays in fields like medicine and robotics. It's widely known that, with a camera, one can project points from space to a plane. But, it is possible the way around? Is there any way to recover the scene from images?

In assignment#3 you learned how a camera can be modeled using projective geometry. In this assignment you will learn how, with only the images, the original scene can be recovered. In our final assignment we will try to answer the question : given several 2D images of an environment, can we recover the 3D structure of the environment, as well as the position of the camera?

In this assignment there are two programming parts:

1. **Sparse** reconstruction
2. **Dense** reconstruction.

Sparse reconstructions generally contain a number of points, but still manage to describe the objects in question. Dense reconstructions are detailed and fine grained.

To help you along the assignment, we have provided a few helpful `.mat` files and two test images : `im1.png` and `im2.png`. The test images are two renderings of a temple from two different angles. Take a close look on subfolder 'data'. On that subfolder you will find `correspondences.mat` which contains good point correspondences. You can use the content of this file to compute the Fundamental matrix. `intrinsics.mat` contains the intrinsic camera matrices, which you will need to compute the full camera projection matrices. Finally, the file `coords.mat` contains some points on the first image (`im1.png`) that should be easy to localize in the second image (`im2.png`).

Your submission for this assignment should be a zip file, `<ClassIDNamesID.zip>`, composed of your report, your Matlab implementations (including any helper functions), and, if the case, your implementations and results for extra credits (optional).

Your final upload should have the files arranged in this layout:

`<ClassIDNamesID>.zip`

- `<NamesID>`

- `<NamesId.pdf> - PDF Assignment report`
- `Matlab`
  - `camera2.m`
  - `computeDepth.m`
  - `computeDisparity.m`
  - `computeE.m`
  - `displayEpipolar.m`
  - `epipolarMatchGUI.m`
  - `findEpipolarMatches.m`
  - `pt2.m`
  - `refineMatrices.m`
  - `rectifyF.m`
  - `testCoords.m`
  - `testDepth.m`
  - `testRectify.m`
  - `triangulation.m`
  - `warp_stereo.m`
  - `Other functions needed`

Please make sure that any file paths that you use are relative and not absolute. Not `imread('/name/Documents/hw4/data/xyz.jpg')` but `imread('../data/xyz.jpg')`.

Every script and function you write in this assignment should be included in the `matlab/` folder. Please include resulting images in your report.

2. Introduction

We know that in the perspective projection process depth information is lost. This missing information may give raise to ambiguities in scene interpretation, as is the case shown in figure 1 that may you question if it represents the real Pisa tower or a smaller model.

One of the the solutions to solve these ambiguities is to recover the lost depth information. Simple geometric intuition shows us that 2 views of the same point are enough to recover its 3D coordinates. These 2 views can be generated by moving a single camera, or, equivalently, by using two cameras as in figure 2. It is clear that the original point can be recovered by intersecting the two lines (projection rays) that pass through the projection centres and respective image points. This problem is only difficult because it is hard to automatically identify which point in the right image corresponds to a given one of the left image (remember assignment#2). As can be seen in figure 2b) multiple points may generate the same left image and have

Figure 1: When we loose depth we do impossible things.

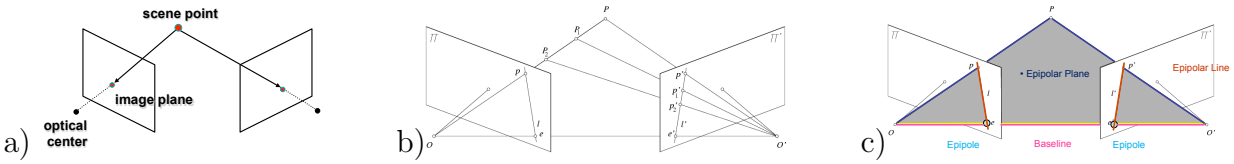

Figure 2: Two cameras to recover 3D scene information.

distinct projections on the right side. Geometry comes again to help us, and figure 2c) shows us that this ambiguity is in fact reduced to a line. As such, the search for the true correspondence needs to be done along this line and not all over the entire right image.

This restriction comes from observing that the two projection centres and the left projection (as well as the corresponding 3D points and respective right projections) lie on a plane. This plane intersects the right image plane along the line that contain all the possible projections of the points that may generate the left image point. This plane is known as the *epipolar plane*, the intersection with the left and right image planes generate the corresponding *epipolar lines*, and the intersections of the line that connects the projection centres (baseline) with the image planes are the *epipoles*.

As the baseline is contained in any epipolar plane, all the epipolar lines intersect at the epipoles. As a consequence knowing the epipolar lines give us some information about the camera pair configuration or single camera movement characteristics. In particular horizontal epipolar lines, tell us that the camera views are parallel. In the following we will develop the idea of estimating the epipolar lines.

3. Essential Matrix

Considering a pair of calibrated cameras (see figure 3), we know not only their projection matrices, but also the rigid transformation between them, which is expressed as a rotation $\mathbf{R}$ and a translation $\mathbf{t}$. So, a point $\mathbf{x}$ in the first camera referential can

be expressed in the second camera's one, by

$$\mathbf{x}' = \mathbf{R}\mathbf{x} + \mathbf{t}.$$

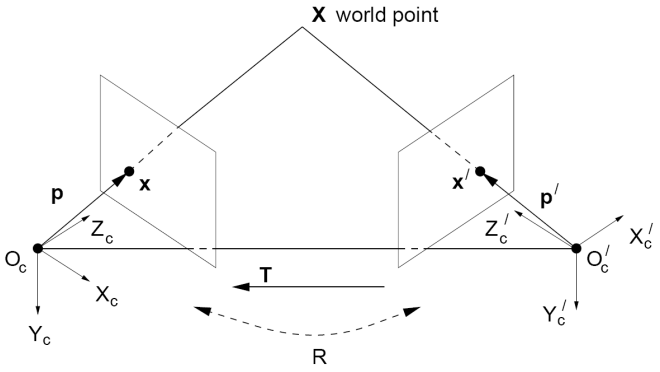

Figure 3: Two calibrated cameras

Seeing the translation as a vector, this vector together with point $\mathbf{x}'$ define an epipolar plane. Their vector product defines a normal to this plane given by

$$\begin{aligned} \mathbf{t} \times \mathbf{x}' &= \mathbf{t} \times (\mathbf{R}\mathbf{x} + \mathbf{t}) \\ &= \mathbf{t} \times \mathbf{R}\mathbf{x} + \mathbf{t} \times \mathbf{t} \\ &= \mathbf{t} \times \mathbf{R}\mathbf{x} + \mathbf{0} \\ &= \mathbf{t} \times \mathbf{R}\mathbf{x} \end{aligned}$$

As the dot product of between two orthogonal vectors is zero, we can write

$$\mathbf{x}'.(\mathbf{t} \times \mathbf{x}') = \mathbf{x}'.(\mathbf{t} \times \mathbf{R}\mathbf{x}) = 0.$$

This can be written as

$$\mathbf{x}'.[\mathbf{t}_\times]\mathbf{R}\mathbf{x} = 0 \tag{1}$$

where the cross product is replaced by a matrix multiplication by making

$$[\mathbf{t}_\times] = \begin{bmatrix} 0 & -t_3 & t_2 \\ t_3 & 0 & -t_1 \\ -t_2 & t_1 & 0 \end{bmatrix}.$$

By making

$$\mathbf{E} = [\mathbf{t}_\times]\mathbf{R}$$

we can write equation (1) as

$$\mathbf{x}'^T\mathbf{E}\mathbf{x} = 0. \tag{2}$$

where $\mathbf{E}$ is called the *essential matrix*, and it relates corresponding image points between cameras, given the rotation and translation.

4. Fundamental Matrix

Note that in equation (2) the points are expressed in the coordinate frames of the respective cameras, not in pixel coordinates. To obtain the corresponding points in pixel coordinates we make use of the projection matrices of each camera and then

$$\mathbf{x_{im}} = \mathbf{K}\mathbf{x}$$

and

$$\mathbf{x'_{im}} = \mathbf{K'}\mathbf{x'}.$$

Then we can get

$$\mathbf{x} = \mathbf{K^{-1}}\mathbf{x_{im}}$$

and

$$\mathbf{x'} = \mathbf{K'^{-1}}\mathbf{x'_{im}}$$

which, once substituted in equation (2) results in

$$(\mathbf{x'_{im}}^{T}\mathbf{K'^{-T}})\mathbf{E}(\mathbf{K^{-1}}\mathbf{x_{im}}) = 0.$$

Now by making

$$\mathbf{K'^{-T}}\mathbf{E}\mathbf{K^{-1}} = \mathbf{F}$$

we get

$$\mathbf{x'_{im}}^{T}\mathbf{F}\mathbf{x_{im}} = 0, \tag{3}$$

where $\mathbf{F}$ is called the *fundamental matrix*, that relates pixels from one camera to pixels of the other.

5. Estimating the Fundamental Matrix

To estimate the fundamental matrix we can use a set of corresponding points on left and right images. Each pair of corresponding points verifies equation 3). As $\mathbf{F}$ is a 3×3 matrix, we have 9 unknowns, but, as equation 3) is homogeneous, the matrix can be defined up to a scale factor. By consequence the number of degrees of freedom is 8 and so is the number of unknowns. To estimate the fundamental matrix, we need therefore at least 8 pairs of corresponding points in the two images. For each pair, we stack the corresponding equation in

$$\begin{bmatrix} x_1x'_1 & y_1x'_1 & x'_1 & x_1y'_1 & y_1y'_1 & y'_1 & x_1 & y_1 & 1 \\ x_2x'_2 & y_2x'_2 & x'_2 & x_2y'_2 & y_2y'_2 & y'_2 & x_2 & y_2 & 1 \\ & & & & \vdots & & & & \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{12} \\ \vdots \\ f_{33} \end{bmatrix} \tag{4}$$

The solution can be computed by performing the singular value decomposition of the above matrix $\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^{\mathbf{T}}$ and selecting the column of $\mathbf{V}$ that corresponds to the null singular value.

In fact, due to measurement noise, the $\mathbf{A}$ is normally of rank 9, instead of 8 as it should be, and so there is no null singular value. The solution is chosen as the column of $\mathbf{V}$ that corresponds to the smallest singular value.

As the $\mathbf{F}$ must be of rank 2, we need to force it. This is done by performing the SVD decomposition of the $\mathbf{F}$ matrix, and replace the smallest singular value in the diagonal of $\mathbf{D}$ matrix with zero. Then the fundamental matrix will be given by

$$\mathbf{F}' = \mathbf{U}\mathbf{D}'\mathbf{V}^{\mathbf{T}}.$$

5.1 Avoiding numerical problems

To ensure the good estimation of the fundamental matrix it is very important to perform a data normalization, to make the solution more stable from the numerical point of view.

For each point set (left and right), we will apply a translation so that origin of the coordinate system will be their centre of mass, and a scale factor so that their mean Euclidean distance to the origin will be 1. Thus, considering the set of points $\{p_i\}, i = 1, ..., n$, the mass centre will have coordinates

$$\bar{x} = \frac{\sum_i x_i}{n}$$

$$\bar{y} = \frac{\sum_i y_i}{n}.$$

The mean distance is given by

$$\bar{d} = \frac{\sum_i \sqrt{(x_i - \bar{x})^2 + (y_i - \bar{y})^2}}{n\sqrt{2}}.$$

Using the triplet $(\bar{x}, \bar{y}, \bar{d})$ for both each of the images we build 2 matrices $\mathbf{T}_l$ and $\mathbf{T}_r$ that will enable us to obtain the normalised coordinates of the points by

$$\hat{\mathbf{p}}_l = \mathbf{T}_l \mathbf{p}_l$$

and

$$\hat{\mathbf{p}}_r = \mathbf{T}_r \mathbf{p}_r.$$

These normalised points will then be used to estimate the fundamental matrix.

The question now is what is the relationship between the fundamental matrix that is obtained from these normalised points and the fundamental matrix that corresponds to the original ones?

These transformed points will verify similar equations given that they still respect epipolar constraints, or

$$\hat{\mathbf{p}}_r^T \tilde{\mathbf{F}} \hat{\mathbf{p}}_l = 0.$$

But from given the relationships between these points and the original ones one can write

$$(\mathbf{T}_r\mathbf{p}_r)^T\tilde{\mathbf{F}}(\mathbf{T}_l\mathbf{p}_l) = 0$$

so we can write

$$\mathbf{F} = \mathbf{T}_r^T\tilde{\mathbf{F}}\mathbf{T}_l.$$

Thus we have a way to estimate a fundamental matrix avoiding numerical problems by centering and rescaling the original image points via the appropriate linear transformations. The obtained matrix can be transformed into the real fundamental matrix by multiplying it with the same transformation matrices.

5.2 Computing cameras from *E*

Once we have determined the essential matrix **E** we need extract cameras from it. From what you have learned in theoretical classes, there are four possible solutions for this task:.

$$P_2 = \begin{bmatrix} UWV^T & u_3 \end{bmatrix} \quad P_2 = \begin{bmatrix} UW^TV^T & u_3 \end{bmatrix}; \tag{5}$$

$$P_2 = \begin{bmatrix} UWV^T & -u_3 \end{bmatrix} \quad P_2 = \begin{bmatrix} UW^TV^T & -u_3 \end{bmatrix}. \tag{6}$$

where $E = Udiag\left(\begin{bmatrix} 1 & 1 & 0 \end{bmatrix}\right)V^T$ and $W = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$.

When we have computed these four solutions we compute the $3D$ points using triangulation for all the choices of P_2 and select the one with where points are in front of both P_1 and P_2. Check also the valid solutions for the $|R|$, i.e, $|R| = 1$. Only one of them have all the points in front of both the cameras. Figure 4 show the four possible camera choices given by the essential matrix.

6. Sparse reconstruction

For this part, you will use two images of a building to compute its sparse reconstruction. This involves the necessary steps which consist in estimating the fundamental matrix, computing point correspondences and then plotting the points in a 3D view. As test images, we provided you two renderings of a temple from two different view angles, as well as a mat file containing point correspondences between the two images. Supported on the conceptual explanation presented above you must:

1. Write a function that computes the fundamental matrix between the two images.
2. Write a function that uses the epipolar constraint to find more point matches between the two images.

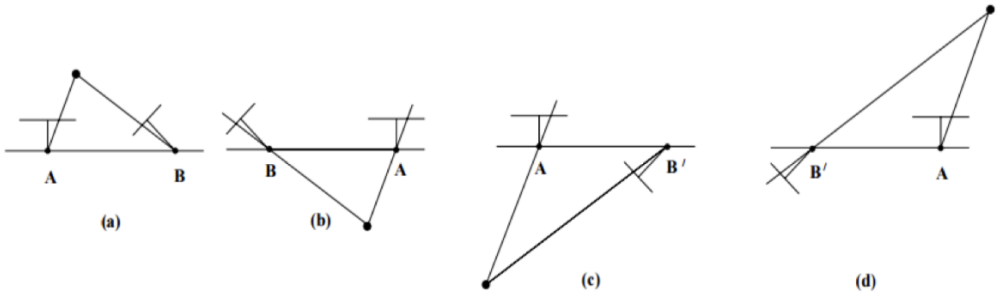

Figure 4: The four possible camera choices given by the essential matrix.

3. Write a function that will triangulate the 3D points for each pair of 2D point correspondences.

6.1 Compute the Fundamental Matrix

The first step is the computation of the Fundamental Matrix using the point correspondences provided in file `correspondences.mat` eight point algorithm. Write a function

```
function F = computeF(pts1, pts2)
```

where *pts1* and *pts2* are $N \times 2$ matrices containing the (x, y) coordinates of the N corresponding points in the first and second image.

For this you should

1. "Normalize" the data points using the matrices $\mathbf{T}_l$ and $\mathbf{T}_r$.

2. Verify that the rank of the $\mathbf{F}$ is 2. To enforce this, you may perform the SVD decomposition of the fundamental matrix $\mathbf{F} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\mathbf{T}$. Then force the smallest singular value in $\mathbf{\Sigma}$ to zero producing a new $\mathbf{\Sigma}'$ and compute the proper fundamental matrix as $\mathbf{F}' = \mathbf{U}\mathbf{\Sigma}'\mathbf{V}^\mathbf{T}$.

3. You may now try to refine the (scaled) matrix using local minimization. Note that this will not fix a completely wrong solution. For this you may call the `refineF.m` function that receives an estimate of the $\mathbf{F}$ matrix and the two point sets.

4. Now you should "renormalize" the matrix.

5. Verify the estimated $\mathbf{F}$ matrix using the `displayEpipolar.m` enclosed function. It receives the $\mathbf{F}$ matrix and two images. Using it you may select a pixel in one of the images and visualize the corresponding epipolar line on the other. Include some of the these images in your report.

6.2 Finding correspondences using epipolar constraints

In previous assignments you have worked on feature detectors and feature descriptors and their use for finding matching pairs between two images. The similarity between descriptors forces the use of statistical sampling methods for selecting the possible true correspondences. The use of the Fundamental matrix provides a great help to this task, as it defines, for one point on one image, the line on the other image where the corresponding one is located. In other words, using it the search for the match is then constrained to be along a known line.

Write a function

```
function pts2 = findEpipolarMatches(im1, im2, F, pts1)
```

where, $im1$ and $im2$ are the two images, $\mathbf{F}$ the fundamental matrix, and $pts1$ contain coordinates of N points in the first image. The function returns the corresponding points in the second image.

1. given one point $\mathbf{x}$ in the first image, compute the corresponding epipolar line and generate a set of candidate points in the second image.

2. For each of these candidate points $\mathbf{x}'$, a similarity score with $\mathbf{x}$ must be computed and the highest value assumed to be the good match.

3. The similarity score may be computed using many different ways. Defining a window around point $\mathbf{x}$ and compare it with ones defined around its corresponding candidates may be enough. Then computing the score as a simple L2- or L1-norm (Manhattan distance) should be enough. If you don't know about the L1-norm search on the web.

4. You may alternatively use some descriptor previously implemented to find the point matches.

5. Use `epipolarMatchGUI.m` to test your implementation. Do not forget to include in your report a screen capture of it using your implementation.

6.3 Compute the essential matrix

Write a function

```
function E = computeE(F, K1, K2)
```

where $\mathbf{F}$ is the fundamental matrix, $K1$ and $K2$ are the intrinsic camera matrices for the first and second images respectively, and which are included in `intrinsics.mat` file. The function returns the computed essential matrix $\mathbf{E}$. Include the computed essential matrix in your report.

6.4 Compute the 3D sparse structure

Now you will compute the 3D points from the pairs of 2D images points. Write the function

```
function pts3d = triangulation(P1, pts1, P2, pts2)
```

where $\mathbf{P_1}$ and $\mathbf{P_2}$ are the projection matrices for left and right cameras, and $pts1$ and $pts2$ the image points' coordinates ordered by correspondences.

Remember that the above projection matrices are obtained by multiplying the intrinsic camera matrices by the corresponding extrinsic camera matrices, and if considering that the stereo pair is defined with respect to camera 1, then its extrinsic matrix is just $\mathbf{P_1} = [\mathbf{I}|\mathbf{0}]$. For $\mathbf{P_2}$ you will use the function given by `camera2.m` and chose among the four solutions it returns the one that is the correct one to use. Using the function given by `triangulation.m`, verify the sparse reconstruction performance by computing the 3D-reconstructed points.

Do not forget to describe the details in your report.

7. Test script for testing the sparse reconstruction

Putting all the above together, you should produce a script `testCoords.m` that:

1. Loads the two images and correspondences from the respective files.
2. Computes the fundamental matrix.
3. Loads the points contained in image 1 from the file `coords.mat` and using your function findEpipolarMatches find the corresponding points in image 2.
4. Computes the essential matrix using the given intrinsics camera models.
5. Computes the two camera projection matrices, (where the first is the trivial one), and for the second you will obtain the four candidates.
6. Uses your triangulation function for each of the four camera models to compute corresponding 3D point sets.
7. Chooses the correct P2 matrix that produce the 3D point set in front of both cameras.
8. Displays the 3D points using matlab plot3.
9. Saves the computed rotation matrices and translation vectors to the file `extrinsics.dat` to be used in the next section.

Be sure that your script runs smoothly. Be careful with path dependences and remember that Linux and MacOS versions filesystems are case sensitive.

8. Building dense point clouds

In most situations, in particular those that imply visualisation, sparse models are not sufficient and dense ones are clearly preferred. In this part you will be working to obtain dense reconstructions of the examples.

This starts with the use of the intrinsic and extrinsic camera parameters to compute the rectified versions of the two input images. With this transformation, the epipolar lines become horizontal and the search for correspondences becomes much simpler. It will then be applied to every pixel and it is possible to compute the disparity and depth maps.

8.1 Image rectification

Write a function that computed the rectification matrices with the following declaration:

```
function [M1,M2,K1n,J2n,R1n,R2n,t1n,t2n] = rectifyMatrices(K1,K2,R1,R2,
    t1,t2)
```

This function uses left and right camera parameters $(\mathbf{K}, \mathbf{R}, \mathbf{t})$ and returns rectification matrices $\mathbf{M1}, \mathbf{M2}$ and the updated camera parameters. You may test your function using `testRectify.m`.

Your function should follow the next steps:

1. Compute the optical centres of the cameras as $\mathbf{c} = -\left(\mathbf{KR}\right)^{-1}\mathbf{Kt}$.
2. Compute the new rotation matrix $\tilde{\mathbf{R}} = \begin{bmatrix}\mathbf{r}_1 & \mathbf{r}_2 & \mathbf{r}_3\end{bmatrix}^T$, where the vectors $\mathbf{r}_i$ correspond to the unitary vectors representing the axes of the camera reference frame.
   (a) The new x-axis ($\mathbf{r}_1$) is parallel to the baseline: $\mathbf{r}_1 = (\mathbf{c_2} - \mathbf{c_1})/||\mathbf{c_2} - \mathbf{c_1}||$.
   (b) The new y-axis ($\mathbf{r}_2$) is orthogonal to x and to the (old) left camera z-axis, which is the third line of $\mathbf{R} = \begin{bmatrix}\mathbf{r'}_1 & \mathbf{r'}_2 & \mathbf{r'}_3\end{bmatrix}^T$: $\mathbf{r}_2 = \mathbf{r'}_3 \times \mathbf{r}_1$
   (c) The new z-axis ($\mathbf{r}_3$) is orthogonal to the previous 2: $\mathbf{r}_3 = \mathbf{r}_1 \times \mathbf{r}_2$.
3. Compute the new intrinsic parameters matrix $\tilde{\mathbf{K}}$. As this can be an arbitrary one, we choose $\tilde{\mathbf{K}} = \mathbf{K}_2$.
4. Compute the new translation vectors: $\mathbf{t}_1 = -\tilde{\mathbf{R}}\mathbf{c}_1$, $\mathbf{t}_2 = -\tilde{\mathbf{R}}\mathbf{c}_2$
5. Compute the rectification matrices: $\mathbf{M}_1 = \tilde{\mathbf{K}}_\mathbf{1}\tilde{\mathbf{R}}_\mathbf{1}\left(\mathbf{K_1R_1}\right)^{-1}$. What about $\mathbf{M_2}$?

You should now test it by running `testRectify.m`. But first analyse this script.

8.2 Dense matching and disparity computation

Write a function that creates a disparity map from a pair of rectified images.

```
function dispMap = computeDisparity(im1, im2, maxDisp, windowSize)
```

where $maxDisp$ is the maximum value allowed for the disparity, and the remaining parameters are self explainable. Note that the output map has the same dimensions as the input images. Since both images are rectified the correspondences computation is reduced to an horizontal search (1D).

$$dispMap(y,x) = \underset{0 \leq d \leq maxDisp}{\arg\min} \; dist\left(im1(y,x), im2(y,x-d)\right)$$

where $dist\left(im1(y,x), im2(y,x-d)\right)$ is any similarity criteria (difference measure) you decide to adopt to measure the difference between windows in comparison. You may use the SSD, as used in assignment#2, or the Normalized Cross-Correlation (NCC) between both windows. Use your preferred way to compute this.

8.3 Depth map

Write a function that creates the depth map from a given disparity map.

```
function depthMap = computeDepth(dispm,K1,K2,R1,R2,t1,t2,bits)
```

You should compute it as

$$depthMap(y,x) = \frac{b \times f}{dispMap(y,x)}$$

b is the baseline, i.e. the distance between optical centres, $f = K_1(1,1)$. For the cases where the disparity value is 0 (zero) set the depth to 0 as well.

You may test your disparity and depth map functions using `testDepth.m`.

Evaluate the depth-resolution obtained for the stereo pair configuration in use. Start by considering a 16-bit representation for the disparity values and gradualy decrease the number of bits in used, i.e., consider disparity representations with 8-bits, 6-bits and finally 4-bits.