LAB: Geometry and Visualization

Course: Computational Photography (DD2429)

KTH, Stockholm, Sweden

September 18, 2013

Introduction

The aim of this laboratory is to understand the relationship between images, i.e. the projection of the 3D world, and the 3D world itself. It consists of three parts:

1. Create a panoramic image from multiple images taken by a rotating camera.
2. Build a 3D model using two images from a calibrated camera.
3. Build a 3D model using two images from an uncalibrated camera.

All three exercises will be carried out with real images taken from a "normal" digital camera. This means that you can use the code to create your own panoramas or 3D models from your own digital images. All exercises will use MATLAB. When you have finished an exercise and answered all questions you should present your work to the lab assistant. *We encourage you to work and present your work in pairs!*

Some good advises in advance: check the web page of the course to make sure that you have the most up-to-date lab instructions. Read *all* the instructions of an exercise carefully before you start implementing that exercise. This laboratory assumes that you are familiar with the theoretical concepts of the chapters $1-4$ in the lecture notes. It is important that the code is written in a readable style. Run the test-scripts to check if your implementation produces correct output. You have to do the exercises in their order, i.e. 1 to 3, since the code will be reused. The first and last exercise are "fairly" short. You might expect to spend most time on the second exercise. Please contact the lab assistant during lab hours if you happen to find any error in these instructions.

GOOD LUCK!

Basic Data Structures

Image points, 3D points and cameras

All the 2D and 3D data will be stored in homogeneous coordinates, instead of the usual Cartesian coordinates. This means that a 2D image point p which has the Cartesian coordinates specified by the column vector $(x, y)^T$ is represented in homogeneous coordinates by a column vector with 3 elements, i.e. $p \sim (x, y, 1)^T$. Similarly, the homogeneous representation of a 3D point P is a column vector with 4 elements, i.e. $P \sim (x, y, z, 1)^T$. Note, if 2D or 3D vectors are written in homogeneous coordinates then "$\sim$" means equality with an unknown scale λ, i.e. p and $p\lambda\, p$ represent the same point. We write this $p \sim \lambda\, p$.

You can use the function $homogeneous\_to\_cartesian$ to convert a point represented using homogeneous coordinates to its representation in Cartesian coordinates. This function normalized the homogeneous coordinates such that the last coordinate is 1 before it is removed. The function can be used to convert a single point or many points, represented as columns in a matrix.

We use homogeneous coordinates since the projection of a camera can then be described as a linear transformation, i.e. represented by a matrix. The projection matrix of a camera is denoted as M and represents a 3×4 matrix. This means that a homogeneous 3D point P_i, which is visible in camera M_c, is projected to the homogeneous image point p_i^c by:

$$p_n^c \sim M_c\, P_n \tag{1}$$

In the code the cameras, 3D points and image points are stored in the variables:

cameras is a $3 \times 4 \times C$ array, where C is the number of cameras. The camera matrix of camera c is given by $M_c = cameras(:, :, c)$.

model is a $4 \times N$ array, where N is the number of 3D points. The n-th 3D point is given by $P_n = model(:, n)$.

data is a $3 \times N \times C$ array storing all image points. $p_n^c = data(:, n, c)$. Furthermore, $data(:, n, :)$ represents the projection of point n to all cameras and $data(:, :, c)$ represents to projection of all points to camera c. If a 3D point P_n is not visible in camera M_c then the image point p_n^c is the column vector $(NaN, NaN, NaN)^T$, where NaN is the MATLAB value representing "not a number".

Images

In this lab you will only work with grey-scale images. All images are stored in a cell array, i.e. *images = cell(C,1).* Pixel (x, y) in image c can be accessed with $images\{c\}(y, x)$. The grey-value in an image ranges from 0 to 255.

Get started

1. Create your own course directory, e.g. *kurs_datorgeometri*
2. Download *labfiles.zip* from the course page.
3. Extract this zip-file to your new course directory. You should then find the following subdirectories in your course directory: *data, debug, functions, images.*
4. Go into the subdirectory *functions* and start MATLAB

5. In MATLAB, choose *Set Path...* from the *File* menu
6. Click *Add with subfolders...* and select your new course directory
7. Click *Save*. If you get a dialog box asking if you want to create a new pathdef.m file, go ahead and create it in the *functions* subdirectory. It will be automatically loaded whenever you start MATLAB from that directory

Content of the Directories

All the necessary data for this lab is in the four subdirectories that you just downloaded. They contain the following data:

functions/ All the given and incomplete functions in MATLAB

images/ All grey-level images in JPG format. The files *.txt list all the respective images for loading them into a file.

data/ Some simulated data.

debug/ Data used to test that the functions you implement give the right output.

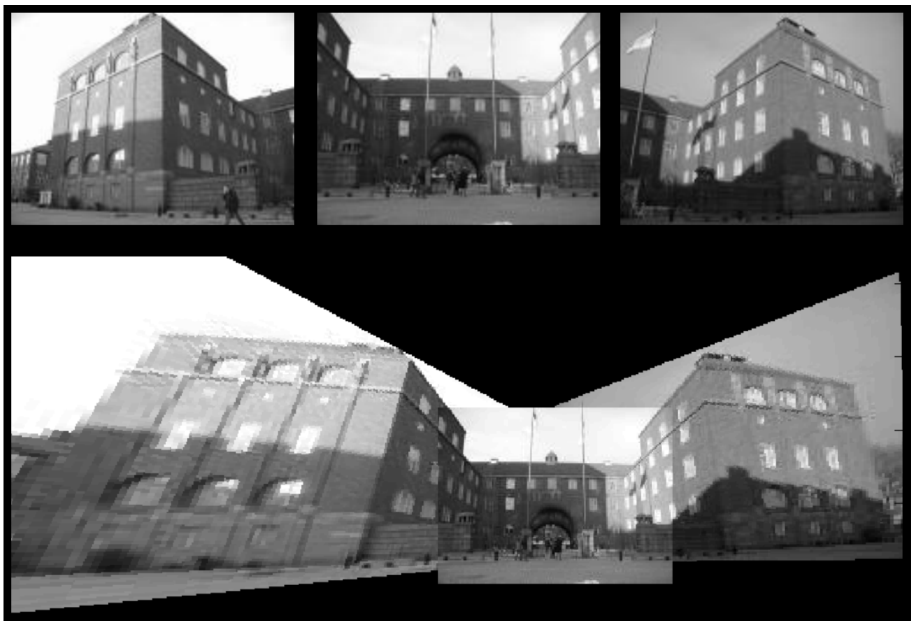

Figure 1: A panorama image created from three separate images.

1 First Exercise: Generating Panoramic Images

In this exercise you will stick together three images as in figure 1. This means that two images are mapped by a linear mapping to a third image (the reference view). Consider two views a and b of a rotating camera. You know from the lectures that the image points p^a and p^b, which are the projections of a 3D point P into the images a and b, are related by a linear mapping H (3×3 matrix (see eqn. (33) in the lecture notes). This gives:

$$p^b = H_{a,b}\, p^a \tag{2}$$

where $H_{a,b}$ relates the views a and b. This linear mapping H is called a *homography*. Let us specify one of the images, e.g. 3, as the reference image (ref). The task of this exercise is to determine the two homographies $H_{1,ref}$ and $H_{2,ref}$ for the two views $1, 2$ to the reference view. If you have computed these homographies, you can map the images 1 and 2 to the reference view. The result is one bigger image, which is built from these three images.

1.1 Incomplete Functions

In this exercise you have to complete the following functions/scripts:

- script: *exercise1_panorama*
- function: *compute_homography*
- function: *compute_normalization_matrices*

We highly recommend you to use the script *exercise1_panorama_test*, to test that each of the functions you write produce the right output. This should make debugging easier.

1.2 Obtain Correspondences between Images

The script which generates panoramic image is called: *exercise1_panorama*. Study this file to understand the main steps of generating a panoramic image. In order to compute homographies you have to get point correspondences. If you run the script *exercise1_panorama* three windows with images will open and you have to specify point correspondences between them. The functionality for doing this is:

Left and middle mouse button: click a point in each view (green)
Right mouse button: finish a correspondence of points (red)
U-key: change to next image (active)
D-key: change to previous image (active)
E-key: stop the clicking process

You should choose view 3 as reference view and click at least four points between view 1 and 3 and view 2 and 3. You should get correspondences for the images specified in *names_images_kthsmall.txt*. These images are not of high quality (size 160×120). Because of the high execution time of the process you should use these small images first. If your algorithm runs correctly you can use the images *names_images_kth.txt*.

1.3 Computing Homographies

Using the point correspondences you should be able to calculate the homographies between a reference view ref and all other views c. For each view c this homography will be stored in *homographies(:,:,c)*. Note, the homography $homographies(:,:,ref)$ has to be set as well. In order to find a specific homography you have to write the function: *H = compute_homography(points1, points2)*. This function should take all the clicked points of two images. This means that $points1$ and $points2$ are matrices of size $3 \times N$, i.e. the n-th point in view 1 is $points1(:,n)$. The homography connects these point sets by the matrix equation:

$$points2 \sim H\ points1. \tag{3}$$

You can determine the homography H as explained in section 1.8 in the lecture notes, i.e. by writing H as a vector:

$$h = (h_{11}, h_{12}, h_{13}, h_{21}, h_{22}, h_{23}, h_{31}, h_{32}, h_{33})^T \tag{4}$$

and formulate a homogeneous system of linear equations:

$$Qh = 0 \tag{5}$$

A numerical sound way to determine the solution of a homogeneous linear system: $Q\ h = 0$ is to first apply a singular value decomposition (SVD) on this matrix:

$$Q = USV^T \tag{6}$$

as discussed in section 5.1 of the lecture notes. In MATLAB a singular value decomposition is achieved by $[USV] = svd(Q)$. S is a diagonal-matrix with the singular values on the diagonal, sorted in decreasing order. The columns of V represent the singular vectors of Q. The solution is then given by the singular vector corresponding to the smallest singular value, i.e. the last column of V. In MATLAB this is given by:

$$h = V(:, end) \tag{7}$$

A more detailed discussion including a proof of this is given in the appendix at the end of these lab instructions.

The solution h is represented by the the nullspace of Q. The nullspace of Q is all linear combinations of singular vectors which have a singular value 0. Therefore, a linear system Q has a unique solution h, with the constraint $||h||^2 = 1$, if the nullspace of Q is one-dimensional. In this case the singular vector corresponding to the singular value of zero, gives the non-trivial solution for the homogeneous linear system (5). If we have measurement the smallest singular value might be larger than zero though.

If you have successfully estimated the homographies, the average error in the pixels should be less than 5.0 for the image set: *names_images_kthsmall.txt*. After correct implementation you should get a panoramic image. The final panoramic image is stored in the directory *images/* with the name *panorama_image.jpg*. Finally some useful tips:

- You can check if the data is NaN with the command $isnan$.
- You might have trouble to calculate the final panoramic image with the function: *generate_warped_image*, e.g. memory exhausted. In this case you can use the alternative but slower function: *generate_warped_image_alt* (see line 58 in script: *generate_panorama*).

1.4 Normalizing the Data

The process of determining the homographies can be further improved by preconditioning the data. In the following discussion we will assume that the image points are written as $p = (x, y, 1)^T$ instead of just $p \sim (x, y, 1)^T$, i.e. that the homogeneous coordinates have been scaled so that the third coordinate is exactly 1, not just up to a scale.

In the previous version, the x- and y-values of the image points have been in the range of [0,640] and [0,480]. This means that elements in the linear system of equation might have very different magnitudes. Since this is a drawback for numerical calculations we will now discuss how to normalize the data. Consider the centroid of the selected image points in image c:

$$\bar{p}^c = \begin{pmatrix} \bar{x}^c \\ \bar{y}^c \\ 1 \end{pmatrix} = \frac{1}{N}\sum_{n=1}^{N} p_n^c \tag{8}$$

We would like the points to be normalized such that the centroid of the selected image points in each image are at the origin, i.e. $(0, 0, 1)^T$. Furthermore, consider the average distance of the selected image points to the centroid for an image c:

$$d^c = \frac{1}{N}\sum_{n=1}^{N} ||\, p_n^c - \bar{p}^c \,|| \tag{9}$$

We would like the image points to be normalized such that this distance is $\sqrt{2}$. This normalization of p_n^c to $p_{n(norm)}^c$ can be expressed using a matrix:

$$N^c = \frac{\sqrt{2}}{d^c} \begin{pmatrix} 1 & 0 & -\bar{x}^c \\ 0 & 1 & -\bar{y}^c \\ 0 & 0 & \frac{d^c}{\sqrt{2}} \end{pmatrix} \tag{10}$$

$$p_{n(norm)}^c = N^c \, p_n^c \tag{11}$$

Complete the function:

$$norm\_mat = compute\_normalization\_matrices(points2d) \tag{12}$$

which takes the data, including NaN for points not visible, and gives back an array $norm\_mat$ of the size $3 \times 3 \times C$, where the matrix $N^c = norm\_mat(:,:,c)$ should represent the normalization matrix for image c. So the normalized points for camera c can be determined by:

$$points2d\_norm(:,:,c) = norm\_mat(:,:,c) * points2d(:,:,c) \tag{13}$$

Be aware that you don't use the NaN data for the normalization. *Tip:* You can check if the normalization is correct by applying *compute_normalization_matrices* on the normalized image data *points2d_norm*. The matrix norm_mat(:,:,c) should then be the identity matrix, for each camera c. A deviation of less then e^{-10} is acceptable.

Calculate now the homography H_{norm} for the normalized points, i.e.

$$p_{norm}^{ref} \sim H_{norm} \, p_{norm}^c \, . \tag{14}$$

Be aware, the homography you have to store in $homographies(:,:,c)$ is still the one between the non-normalized points. If we insert $p_{norm}^c = N^c \, p^c$ into eqn. (14) we obtain:

$$p^{ref} \sim inv(N^{ref}) \, H_{norm} \, N^c \, p^c. \tag{15}$$

This means that the homographies H and H_{norm} are related by:

$$H = inv(N^{ref}) \, H_{norm} \, N^c \tag{16}$$

1.5 Use Larger Images (optional)

If your algorithm runs correctly you can produce a nicer panoramic image from the image set: *names_images_kth.txt*. In oder to do this you have to load these images (see line 16 in the script: *generate_panorama*). These images are of size 640×480. You can click the corresponding points once more or use the data from *names_images_kthsmall.txt*. In the latter case you have to multiply the x- and y-values of the data by 4. Since you algorithm might take a while to produce a nice result, this part of the exercise is optional. *Tip:* Use the function *generate_warped_image_alt* instead of *generate_warped_image* if necessary.

1.6 Questions:

- What is the minimum number of point-correspondances needed to determine the homography between 2 images?
- How much improvement did the normalization yield in terms of average and maximum error?
- Given that you have the minimum number of point-correspondances to determine a homography, the homography might still not be uniquely determined. How can this happen and how can you check if it has happened?
- Look at the function $generate\_warped\_image$. Why does the function use the inverse of the homographies (line 79)?
- Can you extend the method used in this excercise to generate a panorama for images from a camera rotating about 360^o?

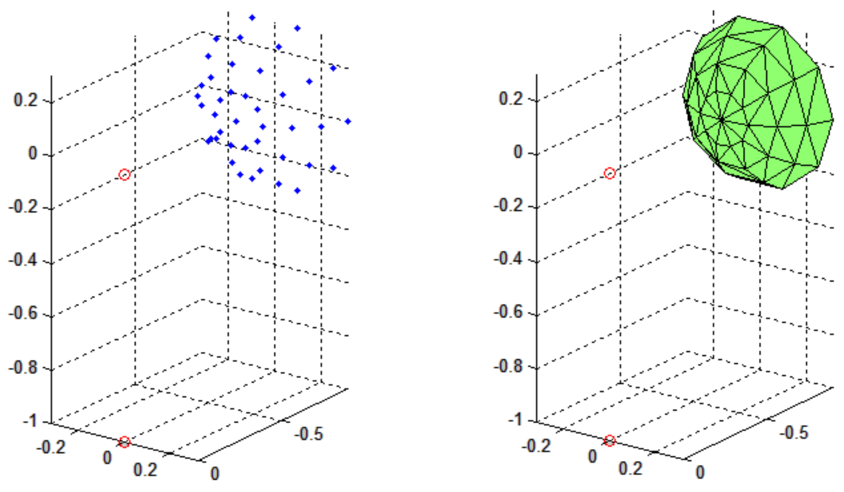

Figure 2: Visualization of the 3D reconstruction with the synthetic data representing a half sphere. On the left hand side we have the reconstructed 3D point cloud. On the right hand side we have the reconstructed triangular surfaces. The red circles indicate the two camera centers.

2 Second Exercise: Calibrated Reconstruction

In this exercise you will reconstruct a 3D object from a pair of images. This is called stereo reconstruction. We assume that the internal calibration of the camera is known. The 3D model and external camera parameters are unknown. The reconstruction will be done on the basis of point correspondences between the images. The main steps of this procedure are explained in section 2.7 in the lecture notes. The final step of this procedure is to convert the reconstructed 3D point cloud to a 3D model with a textured surface as seen in figure 5.

2.1 Incomplete Functions

In this exercise you have to complete the following functions/scripts:

- exercise2_calibrated_reconstruction
- compute_E_matrix
- reconstruct_point_cloud
- reconstruct_stereo_cameras
- check_reprojection_error
- visualize_reconstruction

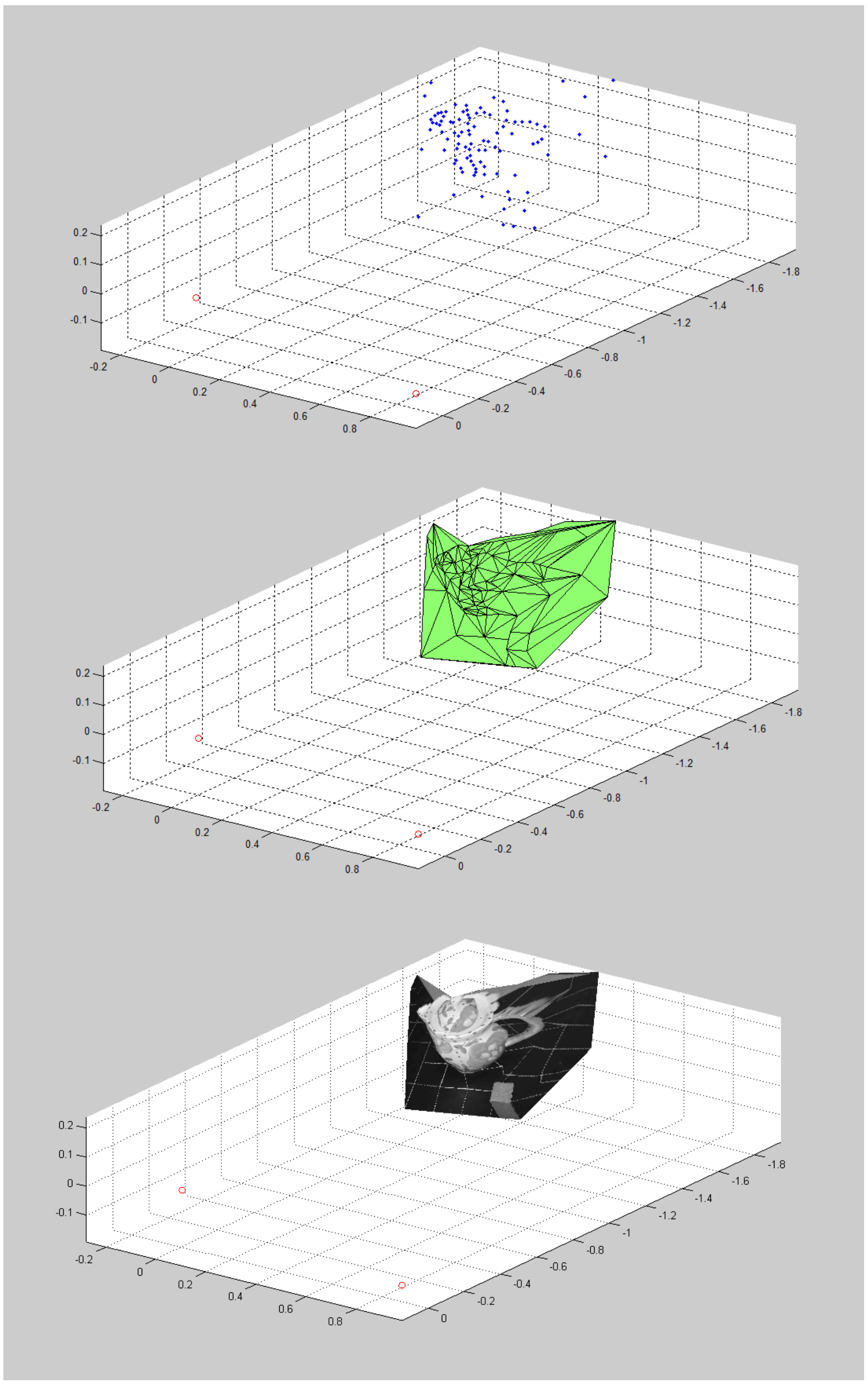

Figure 3: Visualization of the 3D reconstruction with the real data coming from two images of a teapot. The red circles indicate the two camera centers. The top figure shows the reconstructed 3D point cloud. The middle figure shows the reconstructed triangular surfaces. The bottom figure shows the triangular surfaces with a texture, i.e. an image pasted onto each triangle.

2.2 Main Script

The main script is called:

$$exercise2\_calibrated\_reconstruction \tag{17}$$

You should complete this such that it performs all the steps required for a calibrated reconstruction. You can use the script:

$$exercise2\_calibrated\_reconstruction\_test \tag{18}$$

to test parts of your implementation. Before your algorithm works correctly you should use the synthetic (simulated) data by setting the variable VERSION to SYNTHETIC_DATA. When everything seems to work you can change VERSION to REAL_DATA_CLICK to start measuring points in the images of the teapot. After you have done this once you can set VERSION to REAL_DATA_LOAD to just load your saved measurements. The synthetic data is a half-sphere of 55 points seen by two cameras (figure 2). This synthetic data is free from noise, i.e. the 3D points project exactly to the 2D points:

$$points2d(:,:,c) \sim cameras(:,:,c) * points3d \tag{19}$$

The internal calibrations of the cameras are given. For the synthetic data, the calibration of the first and the second camera is:

$$K1 = \begin{pmatrix} 5 & 0 & 3 \\ 0 & 4 & 2 \\ 0 & 0 & 1 \end{pmatrix}, \quad K2 = \begin{pmatrix} 7 & 0 & 1 \\ 0 & 10 & 5 \\ 0 & 0 & 1 \end{pmatrix}. \tag{20}$$

In case of real images, the two calibration matrices are as follows:

$$K1 = K2 = \begin{pmatrix} 2250 & 0 & 400 \\ 0 & 2250 & 300 \\ 0 & 0 & 1 \end{pmatrix}. \tag{21}$$

Note, the principle point $(400, 300, 1)^T$ of the camera correspond to the middle of the image, which has size 800 $\times$ 600. The focal length is 2250 pixels.

2.3 Essential matrix

The first step of a calibrated reconstruction is to compute the Essential matrix from a set of corresponding points. Assume that a point p^a in image a and a point p^b in the other image b corresponds to the same point in 3D. The essential matrix E connects these corresponding points by the epipolar constraint (see equations (94) and (95) in the lecture notes):

$$(x^b \; y^b \; 1) \;\; E \;\; \begin{pmatrix} x^a \\ y^a \\ 1 \end{pmatrix} = 0 \;\; with \tag{22}$$

$$p^a = \begin{pmatrix} x^a \\ y^a \\ 1 \end{pmatrix}, \quad p^b = \begin{pmatrix} x^b \\ y^b \\ 1 \end{pmatrix}, \quad E = \begin{pmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \end{pmatrix}.$$

This can be written as:

$$e_{11}x^bx^a + e_{12}x^by^a + e_{13}x^b + e_{21}y^bx^a + e_{22}y^by^a + e_{23}y^b + e_{31}x^a + e_{32}y^a + e_{33} = 0. \tag{23}$$

The points p^a and p^b are points in a normalized camera coordinate system, i.e. they have the identity matrix $I_{3\times3}$ as the internal calibration matrix. This means that the original image points p^a_{orig}, p^b_{orig} and the normalized points p^a, p^b are related by:

$$p^a = K_a^{-1}\, p^a_{orig}, \quad p^b = K_b^{-1}\, p^b_{orig}\ , \tag{24}$$

where K_a, K_b are the internal calibration matrices of camera a and b respectively. We obtain E by an algorithm which is called the "normalized 8-point algorithm". Each pair of corresponding points p^a, p^b give one linear constraint (23). The nine, unknown elements of the E-matrix can be put into a vector:

$$h = (e_{11}, e_{12}, e_{13}, e_{21}, e_{22}, e_{23}, e_{31}, e_{32}, e_{33})^T. \tag{25}$$

By using the linear constraint in (23), we obtain a linear system $W\ h = 0$, which can be solved as in the previous exercise by using SVD. Since the E matrix has only 8 degrees of freedom (but 9 entries), 8 corresponding points are sufficient. Complete the function:

$$compute\_E\_matrix(points1, points2, K1, K2) \tag{26}$$

$K1$ and $K2$ represent the internal camera matrices K_a and K_b, which you need for transforming the original points to the normalized camera coordinate systems. In order to evaluate if the E matrix is correct you should check the epipolar constraint (22) for each point. For the synthetic data you should get: $p^{b\ T} E\ p^a < e^{-10}$.

2.3.1 Improving Numerical Accuracy

We will now discuss two methods that improves the numerical accuracy when computing the E-matrix from noisy data. As you probably discovered in the first exercise, the normalization of the image data can be important for the numerical stability. You can use the function of the previous exercise: $compute\_normalization\_matrices$ to normalize the points p^a, p^b in the images a and b:

$$p^j_{norm} = N_j\ p^j \tag{27}$$

Note that this is an additional kind of normalization compared to the normalization described in equation 24. The normalization of equation 24 is necessary for us to be able to later reconstruct the camera rotation and translation from the E-matrix. The normalization of equation 27, on the other hand, is not necessary in theory but used in practice for improving the numerical accuracy. If you first compute a matrix F fulfilling:

$$p^{b\ T}_{norm}\ F\ p^a_{norm} = 0. \tag{28}$$

you then get the E matrix as:

$$E = N_b^T\ F\ N_a \tag{29}$$

Note, the function $compute\_E\_matrix$ should always return the matrix E and not F. A property of the E-matrix, but not F, is that two singular values are equal and one singular value is exactly 0. If the image data is noisy this property might not be fulfilled. However, we would like to determine an E matrix which fulfills this property exactly. If E has the singular value decomposition $E = USV^T$, than a correct Essential matrix $E_{correct}$ can be defined as:

$$\begin{aligned} S_{correct} &= \frac{S(1,1) + S(2,2)}{2} && (30) \\ E_{correct} &= U \begin{pmatrix} S_{correct} & 0 & 0 \\ 0 & S_{correct} & 0 \\ 0 & 0 & 0 \end{pmatrix} V^T. && (31) \end{aligned}$$

Include this property and the normalization of the points into your code. If you run the script: $exercise\_calibrated\_reconstruction$ you should obtain an E-matrix which fulfills the epipolar constraint (22) and has the above properties.

2.4 Reconstruct 3D Points

If the cameras are known the points can be reconstructed in 3D, as described in section 2.1 in the lecture notes. In the next subsection (2.5) we will discuss exactly how to reconstruct the cameras. However, before doing that we start with assuming that this has already been done. This is because the function reconstruction the cameras (2.5) will use the 3D point reconstruction function to test different cameras hypotheses.

We have n unknown 3D points, which means that the image data is of size $6 \times n$. Each unknown 3D point $(X, Y, Z, 1)^T$ has 3 degrees of freedom (but 4 entries). Each camera provides two linear equations of the form (73) in the lecture notes. Therefore 4 linear equations (from the two cameras) give a linear system (see middle of page 24):

$$W \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = 0, \tag{32}$$

where W is of size 4×4. This homogeneous system of equations can again be solved by using SVD. Implement this in the function $reconstruct\_point\_cloud$.

2.5 Reconstruct Cameras

The next step is to determine the two camera matrices from the Essential matrix. The two cameras M_a, M_b, which are each 3×4 matrices, are defined as:

$$M_a = K_a(I \mid \mathbf{0}), \;\; M_b = K_b\, R\, (I \mid \mathbf{t}). \tag{33}$$

Note, M_a and M_b are only unique up to scale, i.e. $M_a \sim \lambda M_a$. The remaining unknowns in equation (33) are R which describes the rotation of the second camera and t which is related to its translation. The camera center of the second camera is $-t$. As you know from the lecture these unknowns can be determined from the E-matrix.

We will determine the translation t first. Let us assume that E has the SVD: $E = USV^T$. In section 2.5.1 of the lecture notes, we saw that t is the null vector of E, i.e. $E\ t = 0$. The solution to this homogeneous system of linear equations can be computed as $t = V(:, end)$. Note that the scale of t can not be determined.

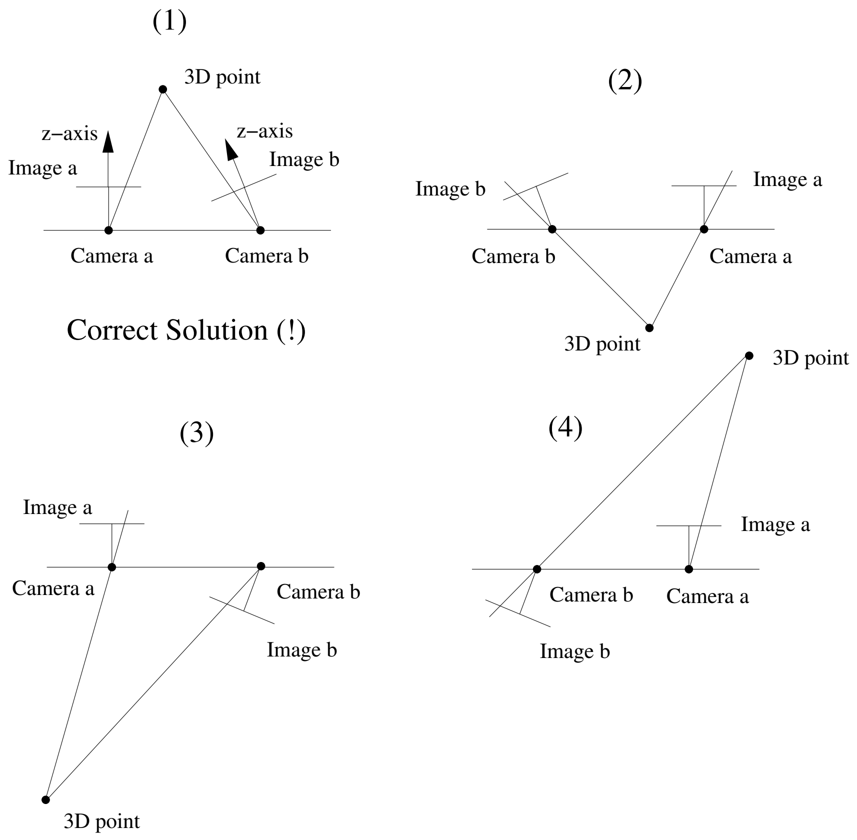

Figure 4: The 4 different solutions for the cameras M_a and M_b which reconstruct one 3D point. However, only one solution satisfies the property that the reconstructed 3D point is in front of both cameras. Between the left and right sides camera b is translated to the other side of camera a. Between the top and bottom rows camera b is roted $180°$ about the baseline, i.e. it is upside down.

Let us now compute the rotation R. We will here present an alternative way to the one discussed in section 2.5.2 in the lecture notes. The rotation matrix R has 2 possible solutions:

$$R_1 = U\,W\,V^T;\;\; R_2 = U\,W^T\,V^T \quad with \quad W = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{34}$$

The matrix W is a rotation matrix. Note, the ambiguity in R is briefly mentioned in the lecture notes. A rotation matrix must have determinant 1. If the determinant is -1 it does not only describe a rotation but also a mirroring. Therefore, you have to multiply R_1 or R_2 with -1 if the respective determinant is -1. The second camera matrix M_b has 4 different solutions:

$$M_b = K_b\,R_1\,(I \mid \mathbf{t}) \quad or \quad K_b\,R_1\,(I \mid -\mathbf{t}) \quad or \quad K_b\,R_2\,(I \mid \mathbf{t}) \quad or \quad K_b\,R_2\,(I \mid -\mathbf{t})$$

The 4 different solutions are displayed in figure 4. The camera M_a is in all 4 cases the same. The difference between the 1 and 2 case is that the translation of the camera

M_b is in the opposite direction, i.e. t and $-t$. The same applies to the third and fourth case. The difference between the first and third case, i.e. R_1 and R_2, is that the camera M_b is rotated about 180^o around the axis which connects the two camera centers. The figure shows that only one solution fulfills an essential property that a reconstructed 3D point is in front of both cameras. In the figure this is case 1. In order to check this property for all 4 solutions of pairs of cameras, you have to reconstruct one point for all 4 possible pairs of cameras using $reconstruct\_point\_cloud$. This is why you should implement 3D point reconstruction first. After that you should complete the function:

$$[cams, cam\_centers] = reconstruct\_stereo\_cameras(E, K, points2d) \tag{35}$$

With the function $reconstruct\_point\_cloud$ you can reconstruct a single 3D point, for all 4 possible pairs of cameras. Determine which of the 4 pairs of cameras is the right one, by checking the property that the reconstructed 3D point is in front of both cameras. To realize how to do this consider the composition of a camera matrix:

$$M = KR(I|t) \tag{36}$$

The transformation without the internal calibration K transforms a 3D point (X, Y, Z) in the global coordinate system to a 3D point (x, y, z) in a system centered at the camera and having a z-axis pointing in the viewing direction of the camera:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = R(I|t) \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{37}$$

By looking at the z-coordinate of the point in the coordinate system aligned with the camera it is possible do determine if the point is in front or behind the camera.

Verify in the end that the following equation holds (see lecture notes eqn. (95)):

$$E = R \begin{pmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{pmatrix} \quad with\ t = \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix}. \tag{38}$$

Remember that E is only determined up to some scale factor.

2.6 Check Re-projection Error

If you have reconstructed the 3D points and the cameras you should check how well the projection constraints are fulfilled:

$$points2d(:, :, c) \sim cams(:, :, c) * points3d \tag{39}$$

You should complete the function:

$$[error\_average, error\_max] = check\_reprojection\_error(points2d, cam, points3d) \tag{40}$$

The average and maximum error represents the average and maximum deviation between a re-projected point $cams(:, :, c) * points3d(:, i)$ and the point in the image $points2d(:, n, c$. Write the function $check\_reprojection\_error$. If you run the script $calibrated\_reconstruction$ with the synthetic data, you should get $error\_average = error\_max = 0.0$. The function $homogeneous\_to\_cartesian$ can be used to normalize the homogeneous coordinates such that the last coordinate is 1, and then remove it, thereby getting the Cartesian coordinates. You should compute the error in Cartesian coordinates.

2.7 Reconstruct Surface

You should now reconstruct the surface of the reconstructed 3D point cloud. We will represent the whole surface by many small triangular surfaces. MATLAB offers a command:

$$triang = delaunay(points\_x, points\_y)$$

You can read more about it with $help\ delaunay$. It basically triangulates a set of n points, where $points\_x$ and $points\_y$ represent the x and y coordinate of a point. $triang$ is of size $k \times 3$, where k is the number of triangles. Each entry in $triang$ is an index (between 1 and n) to the corresponding 2D point. You should add functionality to the function:

$$visualize\_reconstruction \tag{41}$$

It currently visualized the reconstructed cameras and 3D point cloud in a figure. You should extend it such that it also reconstructs the surface of the point cloud using $delaunay$ and then draw the resulting triangles using $trisurf$ in a second figure. If you run the script: $exercise2\_calibrated\_reconstruction$ you should get an extra window which shows the triangulated 3D point cloud in green.

2.8 Texture Mapping

Make sure that you got everything working on both the synthetic and real data before proceeding with this final step. This step only matters for the real data. We would like to add texture to its reconstructed surface. We would like to paste the image from one of the cameras on top of the surface. This is called texture mapping. We have written a function to make this possible in MATLAB:

$$draw\_textured\_triangles \tag{42}$$

Your final task is to extend:

$$visualize\_reconstruction \tag{43}$$

such that it also draws a third figure with the textured surface by calling this function. The argument $textureSize$ controls the resolution of the texture of each triangle. A larger value gives a clearer image, but a slower visualization. A reasonable value is 32, but feel free to experiment with different values.

For the synthetic data the image sent to $visualize\_reconstruction$ as the argument $texture$ is empty, since the synthetic data does not come with an image. Thus, we cannot draw the figure with the textured surface for the synthetic data. You can check if this argument is empty using:

$$isempty(texture) \tag{44}$$

You should only draw the third figure with the textured surface if this argument is not empty.

2.9 Questions

- What is the rank of the essential matrix E and $E_{correct}$ if (a) the selected image points correspond perfectly and (b) if the selected image points do not correspond perfectly, i.e. they do not represent a unique 3D point in space?
- It might happen that all the 3D points lie on a certain surface, called a *critical configuration*, so that the Essential matrix is not unique. How could you detect such a critical configuration?
- Assume that you have m cameras looking at a scene which consists of 3D points. The camera matrices are known and the 3D points unknown. Some of the 3D points are visible in all m views, other points just in a subset of views. How would you obtain the unknown structure in an optimal way?
- The final VRML-model might not look perfect, e.g. the object looks “flat”. What would you suggest to get a nice reconstruction with texture of the whole(!) teapot?

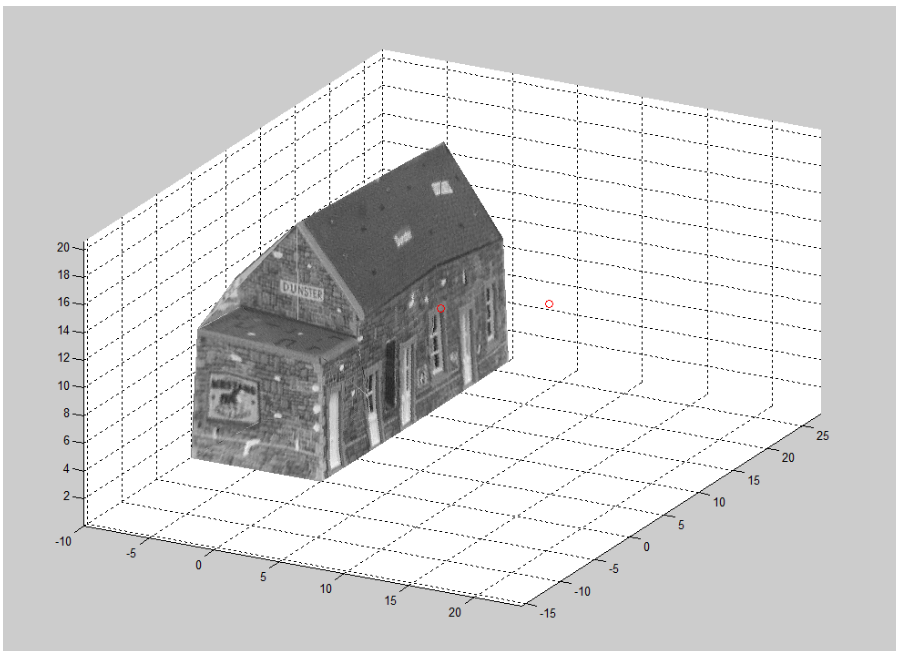

Figure 5: Visualization of the 3D reconstruction with the real data coming from two images of a toy house. The red circles indicate the two camera centers.

3 Third Exercise: Uncalibrated Reconstruction

In the previous exercise you have seen how to obtain a metric reconstruction from two calibrated cameras. However, in practice the calibration of a camera might not be known. In this case it is still possible to reconstruct an object. In this exercise you will reconstruct an object from two views of an uncalibrated camera. The reconstruction is called a *projective reconstruction*.

Since a projective reconstruction does not look very "nice", you have to rectify it. This rectification process transforms the projective reconstruction into a *metric reconstruction*. In this exercise the rectification process uses metric properties about the world. These metric properties are distances between 3D points. Alternatively, the rectification process could exploit the fact that the internal calibration of the camera is constant for several views. This process is then called *self-calibration* and will not be considered here.

3.1 Incomplete Functions

In this exercise you have to complete the following functions/scripts:

- exercise3_uncalibrated_reconstruction
- compute_F_matrix
- reconstruct_uncalibrated_stereo_cameras
- compute_rectification_matrix

3.2 Main Script & Data

The main script for this uncalibrated reconstruction is called:

$$exercise3\_uncalibrated\_reconstruction \tag{45}$$

As in the previous exercise you should first test your algorithms with the synthetic data, i.e. having VERSION=SYNTHETIC_DATA. When everything seems to work you can change VERSION to REAL_DATA_CLICK to start measuring points in the images of the house. After you have done this once you can set VERSION to REAL_DATA_LOAD to just load your saved measurements.

3.3 Fundamental Matrix

The Fundamental matrix plays the same role for uncalibrated cameras as the Essential matrix does in the calibrated case. Two image points p^a, p^b are related with the F-matrix (size 3×3) as (see lecture notes eqn. (127)):

$$(x^b\ y^b\ 1)\ \ F\ \begin{pmatrix} x^a \\ y^a \\ 1 \end{pmatrix} = 0 \quad with\ p^a_{orig} = \begin{pmatrix} x^a \\ y^a \\ 1 \end{pmatrix},\quad p^b_{orig} = \begin{pmatrix} x^b \\ y^b \\ 1 \end{pmatrix}. \tag{46}$$

Note, the image points p^a_{orig}, p^b_{orig} are *not* normalized with respect to the camera coordinate system, i.e. we do not assume that the internal calibration matrix K is the identity matrix $I_{3\times3}$ as we did in the previous exercise. Complete the function:

$$compute\_F\_matrix \tag{47}$$

You can use most of the code that you previously wrote for:

$$compute\_E\_matrix \tag{48}$$

To increase numerical accuracy you should base the computation of the F-matrix on image points normalized by the transformation defined by *compute_normalization_matrices*:

$$p^j_{norm} = N_j\ p^j_{orig} \tag{49}$$

The F-matrix has rank 2. Adapt the F-matrix to $F_{correct}$ so that this property is fulfilled. Remember that the two non-zero singular values of the F-matrix are in general not equal, unlike the E-matrix.

3.4 Projective Reconstruction of Cameras & Point Cloud

It is much easier to reconstruct the two cameras M_a and M_b in the uncalibrated case compared to the calibrated case. The two cameras may be defined as (see lecture notes eqn. (143)):

$$M_a = (I \mid \mathbf{0}),\ \ M_b = (SF \mid \mathbf{h}). \tag{50}$$

The vector h is the epipol in the second camera, i.e. $F^T\ h = 0$. The matrix S is an arbitrary 3×3 antisymmetric matrix. You may choose:

$$S = [v]_\times \quad \text{with } v = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}. \tag{51}$$

Write the function:

$$[cams, cam\_centers] = reconstruct\_uncalibrated\_stereo\_cameras(F)$$

which should compute the two camera matrices and the two camera centers. The camera centers t_a and t_b (4×1 vectors) of cameras M_a and M_b are projected onto the center of projection, i.e. $(0,0,0)^T$. This can be written as:

$$M_a\, t_a = (0,0,0)^T \qquad M_b\, t_b = (0,0,0)^T \tag{52}$$

This means that the camera centers t_a and t_b can be obtained from the SVD of the matrix M_a and M_b respectively. Note that if you want to test your implementation by running the script $exercise3\_uncalibrated\_recosntruction\_test$ you need to define the cameras as in equation 50 and 51.

The 3D point cloud is reconstructed in the same way as in the previous exercise. You have already implemented the necessary function $reconstruct\_point\_cloud$. If you run the script *exercise3_uncalibrated_reconstruction* with the synthetic data, you should obtain *error_average = error_max = 0.0.*

3.5 Rectify the Projective Reconstruction

The reconstruction of 3D points and cameras is correct in the sense that a 3D point P_i is projected via camera M_j onto the image point p_i^j, i.e. $p_i^j \sim M_j\ P_i$. However, we can apply an arbitrary linear transformation H (4×4 matrix) to both the 3D points and the cameras without changing this relationship. If we do this:

$$P_i' = H\ P_i \qquad M_j' = M_j\ H^{-1} \tag{53}$$

we will still get the same projected points in the image:

$$p_i^j \sim M_j'\ P_i' = M_j\ H^{-1}\ H\ P_i = M_j\ P_i. \tag{54}$$

Such a reconstruction is called $projective\ reconstruction$ since H is a general projective transformation. A projective reconstruction does not fulfill metric properties. Some metric properties are: the angle between two lines is correct, the ratio of two lengths is correct, parallel lines are parallel (intersect at infinity). Since we are familiar with these properties, i.e. we live in a Euclidean space, we would like to find a transformation H which rectifies the structure P to P'. The new structure P' should have these metric properties. In order to determine H, we need some metric information about the scene or some information about the camera. The process which exploits only information about the camera is called *self-calibration* but will not be covered in this lab.

One simple way to determine H is to specify the 3D coordinates of some of the points, i.e. exploit metric information about the scene. This might almost be seen as cheating since it is these 3D coordinates that we would like to reconstruct. However, we only need to specify the 3D coordinates for 5 of these points to rectify all of them. A point $P = (x,y,z,w)^T$ and the rectified point $P' = (x',y',z',1)^T$ are related by H as:

$$P' \sim H\ P \quad \text{which is} \quad \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} \sim \begin{pmatrix} h_{11} & h_{12} & h_{13} & h_{14} \\ h_{21} & h_{22} & h_{23} & h_{24} \\ h_{31} & h_{32} & h_{33} & h_{34} \\ h_{41} & h_{42} & h_{43} & h_{44} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}. \tag{55}$$

Note, in a projective reconstruction the last coordinate w of a point P might be 0. Therefore, we can not normalize w to 1. You have seen such a mapping H already in the first exercise (see eqn. (2)). In that case H connected two 2D homogeneous points. You can think of H in eqn. (55) as a 4D homography which connects two 3D homogeneous points. The matrix H in eqn. (55) has 16 entries and 15 degrees of freedom (H is unique up to scale). Two corresponding points P, P' give three linear equations of the form:

$$\begin{aligned} x' &= \frac{h_{11}x + h_{12}y + h_{13}z + h_{14}w}{h_{41}x + h_{42}y + h_{43}z + h_{44}w} \\ y' &= \frac{h_{21}x + h_{22}y + h_{23}z + h_{24}w}{h_{41}x + h_{42}y + h_{43}z + h_{44}w} \\ z' &= \frac{h_{31}x + h_{32}y + h_{33}z + h_{34}w}{h_{41}x + h_{42}y + h_{43}z + h_{44}w} \end{aligned} \tag{56}$$

This can be written as:

$$\begin{aligned} h_{11}x + h_{12}y + h_{13}z + h_{14}w - h_{41}xx' - h_{42}yx' - h_{43}zx' - h_{44}wx' &= 0 \\ h_{21}x + h_{22}y + h_{23}z + h_{24}w - h_{41}xy' - h_{42}yy' - h_{43}zy' - h_{44}wy' &= 0 \\ h_{31}x + h_{32}y + h_{33}z + h_{34}w - h_{41}xz' - h_{42}yz' - h_{43}zz' - h_{44}wz' &= 0 \end{aligned} \tag{57}$$

Therefore, 5 corresponding 3D points give 15 equations and define the matrix H uniquely up to scale. These 15 equations can be put into one linear system W of equations:

$$W\,h = 0 \tag{58}$$
$$h = (h_{11}, h_{12}, h_{13}, h_{14}, h_{21}, h_{22}, h_{23}, h_{24}, h_{31}, h_{32}, h_{33}, h_{34}, h_{41}, h_{42}, h_{43}, h_{44})^T$$

This linear system can be solved by applying a SVD on the matrix W. Write the function:

$$H = compute\_rectification\_matrix(points1, points2)$$

which returns the mapping H. The mapping H should satisfy: $points2 \sim H\ points1$. The arguments $points1$ and $points2$ are of size $4 \times n$ (n is in this case 5). You do *not* have to normalize the 3D points. Furthermore, you have to rectify the 3D point cloud, cameras and camera centers with the use of H. Complete the part of the script:

$$exercise3\_uncalibrated\_reconstruction \tag{59}$$

which starts with the comment: *% rectify the cameras and the model.* Tip: The camera centers are rectified in the same way as a 3D point.

If you run the script *exercise3_uncalibrated_reconstruction* with the synthetic data, you should obtain the half-sphere and the two camera centers as in the previous exercise (see figure 2). The first and second camera center should be: $t_a = (0,0,0,1)^T, t_b = (0,3,0,1)^T$.

After everything works with the synthetic data you should try the real data. For the real data you have to manually provide the ground truth 3D coordinates for 5 of the reconstructed points in order to rectify the projective reconstruction. Figure 6 is helpful for this.

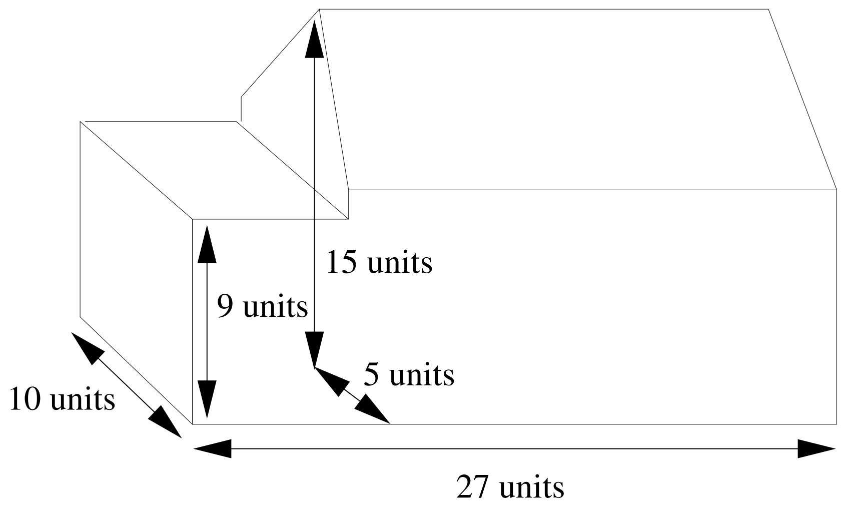

Figure 6: The dimensions of the toyhouse.

3.6 Questions

- How do you determine the rectification matrix H in an optimal way if you have more than 5 3D points given?
- What is the condition that two set of points define a unique rectification matrix H?
- Assume that you do not rectify the model, i.e. H = *eye(4)*. What kind of error can occur if you convert the 3D point cloud from homogeneous coordinates to Cartesian coordinates using the function *homogeneous_to_cartesian*?
- How many degrees of freedom has the rectification matrix H? And how many degrees of freedom of H do you have to fix in order to rectify the 3D point cloud to Euclidean space? Note, a model in the Euclidean space can only be rotated and translated.
- Assume that you have 3 cameras and $n \geq 8$ 3D points which are visible in all views. The cameras and the 3D points are unknown. How do you obtain 3 correct camera matrices? If you have achieved this, how would you obtain all the 3D points?
- Assume that you have extended this method to more than two views. What are the main differences between this method and the factorization method for multiple parallell projection cameras described in section 4 of the lecture notes?

Appendix - Linear Homogeneous Systems of Equations

We want to solve systems of linear homogeneous equations:

$$Ax = 0 \tag{60}$$

where A is a known matrix and x is an unknown column vector. We always have the trivial solution $x = 0$ but we are interested in finding other solutions. The elements of A typically comes from noisy measurements and therefore the equation does not have an exact non-trivial solution in general. We therefore instead solve a similar problem which always has an exact solution:

$$\min_{\|x\|^2=1} \|Ax\|^2 \tag{61}$$

This optimization problem can be solved using the singular value decomposition (SVD). This decomposition of an arbitrary matrix A is defined as:

$$A = USV^T \tag{62}$$

where U and V are orthonormal matrices and S is a matrix with only diagonal elements, the singular values, which are non-negative and given in descending order. It can be computed in MATLAB by: [U,S,V] = svd(A). The solution to (61) is then given by the last column of V:

$$x = V(:, end) \tag{63}$$

Proof To solve our optimization problem we will first simplify the objective function and the constraint using the SVD of A. We will also use the fact that multiplication with an orthonormal matrix does not change the norm of a vector:

$$\|Ax\| = \|USV^T x\| = \|SV^T x\| \tag{64}$$

and

$$\|x\| = \|V^T x\| \tag{65}$$

If we also do the variable substitution $y = V^T x$ we get:

$$\|Ax\| = \|USV^T x\| = \|SV^T x\| = \|Sy\| \tag{66}$$

and

$$\|x\| = \|V^T x\| = \|y\| \tag{67}$$

We can therefore first solve the simpler problem:

$$\min_{\|y\|^2=1} \|Sy\|^2 \tag{68}$$

where S only has diagonal elements and then substitute back from y to x.

We solve this constrained optimization problem using the method of Lagrange multipliers. It converts the constrained optimization to the unconstrained optimization by adding an extra variable λ:

$$\min_{y,\lambda} f(y,\lambda) \tag{69}$$

where the new objective function is:

$$f(y,\lambda) = \|Sy\|^2 + \lambda(1 - \|y\|^2) = \sum_n (S_n y_n)^2 + \lambda\left(1 - \sum_n y_n^2\right) \tag{70}$$

We look for extreme values where all the partial derivatives are zero:

$$\frac{\partial}{\partial y_i} f(y,\lambda) = 2S_i^2 y_i - 2\lambda y_i = 0 \tag{71}$$

$$\frac{\partial}{\partial \lambda} f(y,\lambda) = 1 - \sum_n y_n^2 = 0 \tag{72}$$

The first constraint can be rewritten to:

$$y_i(S_i^2 - \lambda) = 0 \tag{73}$$

To solve this system of non-linear equations we will first assume that all singular values are different:

$$S_i = S_j \leftrightarrow i = j \tag{74}$$

Equation (73) is fulfilled if $y_i = 0$ or $\lambda = S_i^2$. If we assume $\lambda = S_i^2$ then $\lambda \neq S_j^2$ and therefore $y_j = 0$ for all $j \neq i$. Equation (72) then gives $y_i = 1$. The value of the objective function at this extreme value is:

$$\|Sy\|^2 = \sum_n (S_n y_n)^2 = S_i^2 \tag{75}$$

The global optimum is therefore given by the index i corresponding to the lowest singular value, i.e. the last index. The corresponding value of x is:

$$x = Vy = \begin{pmatrix} v_1 & \dots & v_N \end{pmatrix} \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix} = v_N \tag{76}$$

i.e. the last column of V. In MATLAB this is given by V(:,end). Note that we have assumed that all singular values are different. If the second smallest singular value is very close to the smallest, we have two minima that are almost equally good. We can therefore check if we have a unique solution by comparing the two smallest singular values.

Assignment for the Course: Datorgeometri i bildanalys och visualisering (2D1424)

Students Namn: ....

Personnummer: ....

Exercise 1: ....

Datum Kursledare/kursassistent

Exercise 2: ....

Datum Kursledare/kursassistent

Exercise 3: ....

Datum Kursledare/kursassistent