Determining depth in scene

COMS30121 - Image Processing and Computer Vision

Jaroslav Borovička – pinus@centrum.cz, jb2383@bris.ac.uk
University of Bristol / Czech Technical University in Prague

Contents

1 Problem statement **1**

1.1 Convolution and correlation 1

1.2 Smoothing the image 2

1.3 Edge detection 2

1.4 Calculating the direction map 3

1.5 Finding corresponding points 3

1.6 Triangulation 4

2 Key issues affecting the performance **4**

3 Experimental results **6**

3.1 Table scene 6

3.2 Office scene 6

3.3 Other scenes 10

4 Critical analysis and possible improvements **10**

5 Program listing **12**

1 Problem statement

The objective of this application is to determine the depth in a scene from two views obtained from a calibrated stereo system. This is done by reconstructing the 3-D position of detected edge points. The task is subdivided into following parts, which are then discussed in more detail in further sections.

First, edges have to be detected in both images. In order to reduce the noise in the image, I use a *Gaussian smoothing filter*. Then, edges are detected by using *horizontal and vertical Sobel filters*, and merging the obtained information about horizontal and vertical edges. The Sobel filters are also used to determine the edge directions. After some testing, I also added a *second-derivative* filter which, in connection with the first-derivative Sobel filters, offers a more robust way of edge recognition.

Secondly, the detected edge features in both images have to be matched. This is done by taking each edge pixel from the left image, and calculating the corresponding epipolar line in the right image. The corresponding edge pixel in the right image will lie on (or close to) the epipolar line. I tried to implement various measures for finding the matching edge point, and results are discussed in Section 3. Finding the matching points proved to be by far the most difficult part of the assignment.

After finding the matching points, 3-D position of the points in the actual scene is reconstructed using triangulation. Obviously incorrectly detected points (noise) are removed.

I also used an alternative method. I marked important points in one image manually, then ran the program in order to find corresponding points in the second image using the algorithm described above, and connected the calculated 3-D points with lines. Results are also shown in Section 3.

1.1 Convolution and correlation

Smoothing the image and using the Sobel filters requires the implementation of the discrete convolution functional. Since only filters of small sizes are used, I programmed the convolution functional in spatial coordinates, rather than using the Fourier transform (i.e. I do $h = f * g$ directly instead of $h = \mathcal{F}^{-1}(\mathcal{F}f \cdot \mathcal{F}g)$. Although the multiplication of the Fourier images is much faster than the convolution in the spatial coordinates, the Fourier transform, implemented as Fast Fourier Transform with its complexity $O(n \log n)$, would be inefficient for convolution of the image and a small filter. The convolution is implemented in the `convolution` function, and it generally uses the (discrete) formula

$$h[k,l] = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f[i,j]\, g[k-i, l-j]$$

but in practice, i and j only run through the support of $f[i,j]\, g[k-i, l-j]$. We should also not forget that convolution distorts the image along the edges (since it assumes a periodic function, which is not the case here, and we rather pad the image with zeros). Therefore, we should rather neglect the edges when working with the image after convolution.

As similarity measure, I used the correlation coefficient calculated on a selected neighbourhood of the two points being compared. Theoretically, I could also use the Fourier images but again, since the area is quite small, I calculate the correlation in spatial coordinates. I use the formula

$$\rho(a,b) = \frac{Cov(\Omega_a, \Omega_b)}{\sqrt{Var\Omega_a Var\Omega_b}}) = \frac{\sum_{i,j}(v_a - \overline{v}_a)(v_b - \overline{v}_b)}{\sqrt{\sum_{i,j}(v_a - \overline{v}_a)^2 \sum_{i,j}(v_b - \overline{v}_b)^2}}$$

where Ω_a, Ω_b are the neigbourhoods of the points a, b, and the sums are over these neighbourhoods. v is the compared quantity (i.e. magnitude or angle of the edge, or intensity). Adjustment has to be made for the angle, since ϵ and $2\pi - \epsilon$ are actually very similar angles. This is considered in the implementation.

1.2 Smoothing the image

Smoothing the image is accomplished using a 2-dimensional Gaussian smoothing filter. The size of the filter (standard deviation of the conjugate normal distribution) can be specified using the `-s` command-line parameter. Since approximating the values of the distribution function of normal distribution is rather complicated, I use the probability density function instead. So, since the centre of the filter is shifted to $(0, 0)$, we have

$$\text{smoothfilter}\,[i,j] = \frac{1}{(\sqrt{2\pi}\sigma)^2} \cdot e^{-\frac{i^2+j^2}{2\sigma^2}}$$

instead of (mathematically correct)

$$\text{smoothfilter}\,[i,j] = \int_{i-0.5}^{i+0.5} \int_{j-0.5}^{j+0.5} \frac{1}{(\sqrt{2\pi}\sigma)^2} \cdot e^{-\frac{x^2+y^2}{2\sigma^2}}\,\mathrm{d}y\,\mathrm{d}x$$

However, the error is less than 5% for a vast majority of cases, a number more than sufficient for our application. The filter values are then normalised so that they add up to one. The smoothing is then done by convoluting the above defined filter with the image (see function `smoothimage` for more detail on the application).

1.3 Edge detection

The edge detection is based on the implementation of Sobel filters. The horizontal and vertical Sobel filters are applied on the smoothed image. The filter masks are shown in Figure 1. These filters are

-1	0	1		-1	-2	-1
-2	0	2		0	0	0
-1	0	1		1	2	1

Figure 1: Vertical (left) and horizontal (right) Sobel filter masks

convoluted with the image, and then merged using the L_2 (Euclidean) norm

$$|\nabla f| = \text{sobel}_{merged}[i,j] = \sqrt{(\text{sobel}_{vert}[i,j])^2 + (\text{sobel}_{horiz}[i,j])^2} = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}.$$

sobel_{merged} then contains the edge map where each pixel contains the intensity of the edge. This edge map is then normalised and thresholded using the `-t1` parameter (values 0..255 – 0 corresponds to no thresholding, 255 to removing all edges). The suitable value of the threshold very much depends on the type of the image, as discussed in Section 3.

The Sobel filters themselves are however quite often mislead by noise which they falsely recognise as edges. This can be to a large extent solved by smoothing the image but then the edges tend to be thicker. A good solution is to support the Sobel filter by a 2^{nd}-derivative filter. Whereas Sobel filters are based on looking-up the maxima of the first derivative (middle graph of Figure 2, the 2^{nd}-derivative filter is looks-up zero crossings of the 2^{nd}-derivative (right image of Figure 2). The 2^{nd}-derivative filter is implemented in the following way. To reduce the occurrence of high frequencies (noise), we smooth the image, and then apply the Laplace operator on the smooth image, i.e.

$$\nabla^2(G * f) = \frac{\partial^2(G * f)}{\partial x^2} + \frac{\partial^2(G * f)}{\partial y^2}$$

where f is our image, and G is the Gaussian smoothing filter. From the linearity of convolution, it follows that

$$\nabla^2(G * f) = (\nabla^2 G) * f.$$

$(\nabla^2 G)$ (Laplacian of Gaussian – LoG) can be precomputed, and has the "Mexican hat" form (two "Mexican hat" filters of different sizes are defined in the beginning of the source code, page 12).

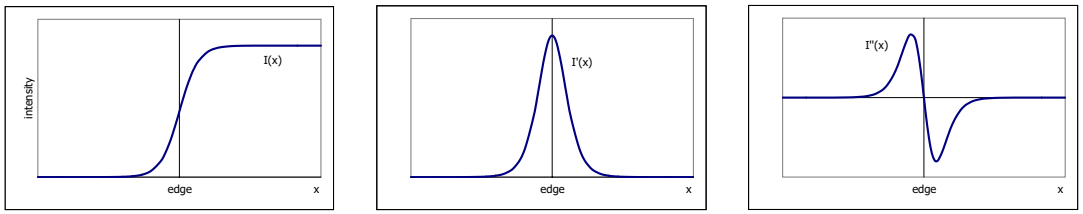

Figure 2: Behaviour of the intensity function (left), its derivative (middle), and second derivative (right) around an edge. The first derivative reaches its maximum, the second derivative crosses zero at the edge.

After applying the "Mexican hat" filter, the zero-crossings have to be found. This is done by running a 2x2 window over the image — if both positive and negative values appear in the window, a zero crossing occurs. A point is then considered as an edge point when the Sobel filter signalises an edge and the "Mexican hat" filter signalises a zero-crossing. See method `EdgesUsing2ndDer` in the `TPGMImage` class for implementation.

After edge maps are calculated, they are thresholded in order to keep only significant edges. Suitable values for the thresholding parameters are discussed in Section 3.

1.4 Calculating the direction map

In order to calculate the direction map (see function `thresholdandfinddirectionmap`), we make use of the already calculated convolutions of the image with the horizontal and vertical Sobel filter. For every detected edge point $[i, j]$ we calculate the angle θ as

$$\theta[i,j] = \tan^{-1}\left(\frac{\partial f}{\partial y}[i,j] \Big/ \frac{\partial f}{\partial x}[i,j]\right) = \tan^{-1}\frac{\text{sobel}_{vert}[i,j]}{\text{sobel}_{horiz}[i,j]}$$

Using the function `atan2` in `C` gives angles in the interval $(-\pi, \pi)$. However, since edges with angles θ and $\theta + \pi$ have got the same direction, we can shift all $\theta[i, j]$'s so that they lie in $(-\pi/2, \pi/2)$.

1.5 Finding corresponding points

In order to achieve this task, we need to establish the dependence between corresponding points. It follows from epipolar geometry that, for a point in the left image plane, the corresponding point in the right image plane will lie on the epipolar line. The epipolar line corresponds to the straight line which passes through the left camera and the point in the left image plane, projected onto the right image plane. Since (keeping the notation in lecture notes) P_l, T, and $P_l - T$ all lie in one plane, we have $(P_l - T)^{\mathrm{T}}(T \times P_l) = 0$. Now, each vector product can be expressed as a product of a matrix with one of the vectors ($T \times P_l = SP_l$). We obtain

$$\begin{aligned}
(P_l - T)^{\mathrm{T}}(SP_l) &= 0 \qquad \text{and since we have} \quad P_l - T = R^{\mathrm{T}}P_r \\
P_r^{\mathrm{T}}RSP_l &= 0 \qquad \text{and we define} \quad E \triangleq RS \quad \text{as the } \textit{essential matrix} \\
P_r^{\mathrm{T}}EP_l &= 0 \qquad \text{we divide by} \quad Z_rZ_l/f_rf_l \\
p_r^{\mathrm{T}}Ep_l &= 0
\end{aligned}$$

In pixel coordinates we have $\overline{p}_{l,r} = M_{l,r}p_{l,r}$, where $M_{l,r} = \begin{pmatrix} -1/s_x & 0 & o_x \\ 0 & -1/s_y & o_y \\ 0 & 0 & 1 \end{pmatrix}$.

$$\begin{aligned}
\overline{p}_r^{\mathrm{T}}(M_r^{-1})^{\mathrm{T}}EM_l^{-1}\overline{p}_l &= 0 \qquad \text{and we define} \quad F \triangleq (M_r^{-1})^{\mathrm{T}}EM_l^{-1} \quad \text{as the } \textit{fundamental matrix} \\
\overline{p}_r^{\mathrm{T}}F\overline{p}_l &= 0 \qquad \text{and since} \quad \overline{p}_r = (\overline{x}_r, \overline{y}_r, f)^{\mathrm{T}} \quad \text{and} \quad \overline{u}_r = F\overline{p}_l \\
\overline{x}_r\overline{u}_{rx} + \overline{y}_r\overline{u}_{ry} + f\overline{u}_{rz} &= 0
\end{aligned}$$

This is the equation of the epipolar line. Now we examine all edge pixels that lie on (or very close to) the epipolar line, and calculate the similarity measure between the pixel in the left image and these pixels. We consider the pixel with the highest similarity as the matching pixel (provided that the value of the similarity measure, i.e. correlation is sufficiently high).

We can significantly reduce the number of incorrectly matched points (noise) by running reversed matching. Once a match for a point from the left image is found in the right image, we run the matching procedure on the point from the right image. If the matching point in the left image is identical (or very close, say 1 pixel difference) to the original point, the match is confirmed. Otherwise, we remove the point from the list of matches. Of course the formulas for the right→left matching will slightly differ from the above stated formulas but their derivation is straightforward.

1.6 Triangulation

Each matching pair of pixels defines a pair of rays which should, ideally, meet in the 3-D position of the point in the scene. However, because of inaccuracies, this will not happen. We therefore find the shortest transversal line between the two rays, and set the position of the 3-D point into the middle of this transversal line. The direction vectors of the two rays, as seen from the point of view of the left camera, are p_l and $R\mathrm{T}p_r$. The rays can be therefore parametrised as ap_l and $bR^{\mathrm{T}}p_r + T$ (T is the shift between the cameras). Shortest transversal line between the two rays is perpendicular to both rays, and therefore can be parametrised as $c(p_l \times R\mathrm{T}p_r)$. Finding the shortest transversal line then means solving the set of 3 linear equations

$$ap_l - bR^{\mathrm{T}}p_r - T + c(p_l \times R\mathrm{T}p_r) = 0$$

for the unknown parameters a, b, and c. The middle of the transversal line, from the point of view of the left camera, is then $ap_l + \frac{c}{2}(p_l \times R\mathrm{T}p_r)$. This is our estimate $\hat{P}_l$ of the 3-D point P_l. Ideally, $c = 0$, and then $a = Z_l/f$.

2 Key issues affecting the performance

The most severe problem is finding the correct matches for the features in the left and right image. Following issues, ordered in the succession how they occur when processing the images, influence the performance of the program. I also state measures that were implemented in the program in order to eliminate or reduce these problems.

- *Quality of the images:* This is an issue where the program can offer only secondary solutions. Once low-quality images are submitted on input, they can be adjusted in order to suit better the implemented algorithms but lost information cannot be won back. Since we based our algorithms on edge detection, the main problems are *noise in the images*, and *low contrast*. I handle the noise in the images (which is present mainly due to the low color quantisation) by *smoothing the images* before they are submitted to the edge detector. Low contrast could be handled by contrast stretching or histogram equalisation.

- *Suitable edge detector:* Finding a reasonable edge map is very important for two reasons. Firstly, the edge map gives us the features (edge points) which we are trying to match. Secondly, the edge map provides data for calculating the similarity measure when looking up the matching points. The implemented Sobel filters, since they work only on a 3×3 area, are not very resistent to noise. I therefore implemented a *2nd order filter* which supports the Sobel filters. A pixel is only considered as an edge if both the Sobel filter and the 2nd order filter confirm the pixel as an edge.

- *Position of the cameras:* If the cameras are *too close* to each other, we get bad stereo information (the images are nearly identical and we the extraction of the depth is very sensitive to even very small errors). On the other hand, if the cameras are *too far apart*, problems with too big differences between the too images occur. These include *distortions in shape and*

scale, different *illumination levels*, or *occlusions* (objects hidden by other objects, or objects visible from a different side). A partial solution to the problem with too distant cameras might be using feature based methods rather than correlation, since correlation methods are more sensitive to big differences in shape and shade.

- *Selection of features that are being matched:* I restricted my algorithm on working with *individual edge pixels*. Such a method produces a dense disparity map which is illustrative for a human observer since the whole surface is reconstructed (assuming that there is enough edge points that we can match). The main advantage of this method is that we do not have to make any assumptions about the image (type of occurring objects etc.). Textured images are of an advantage for this method.

  Methods working with more sophisticated objects are discussed in Section 4. I also tried processed some of the images using a simplified variant of such a method: I marked the features (in my case corners) in one image by hand, telling the program which corners should be connected in the end. Then I ran the algorithm to fing the correspondences in the second image, and reconstructed a 3-D scene by connecting the corners with lines. Results are shown in Section 3.

- *Issues concerning correlation:* Correlation based methods can be slow, especially on larger images, and with larger matching windows. Fortunately, we can restrict our search to the epipolar line, as described in Section 1.5.

  Another question is the size of the matching window. I experimented with the width of the window, $2W + 1$, and consider values around $W = 10$ as most suitable for a variety of cases. Smaller window sizes often result in spurious matches, larger windows contain already too big distortions caused by different positions and angles of the two cameras. Moreover, larger windows slow down the algorithm (quadratically with W). Naturally, the further apart the cameras are, the more difficult is the matching using correlation methods because the distortions increase.

  Finally, we have to ask which features should be matched. Using the options we have, we can use the original image (intensities), magnitudes of the edges, or angles of the edges. My experiments showed that a combination of these three correlations is suitable. Taking always the maximum value of these three correlations also gave good results.

- *Spurious matches:* None of the above described precautions could fully avoid spurious matches. These spurious matches result in an incorrect choice of the 3-D point on the ray which passes through the camera and the image point. Since these points still lie on the ray, they are not observable when looking on the 3-D reconstruction from the position of the camera. However, once we start rotating the 3-D reconstruction, they appear as incorrectly positioned points (noise).

  In order to solve this problem, I implemented the idea of *reversed matching* (described in Section 1.5). Reversed matching siginificantly reduced the amount of noise in the 3-D reconstructions (most, but not all, of the spurious matches were removed).

  Another measure, which however requires some prior knowledge about the scene, is to restrict the range for the Z coordinate. If we approximately know where the scene is positioned, we can remove all points with Z coordinate outside this range. A natural restriction is of course $Z > 0$, or, for some applications, $Z > f$.

- *Occlusions:* These particularly happen when cameras are far apart, or the objects are very close to the cameras. There is no way how to reconstruct the 3-D position of points that are occluded in one of the images. Even worse, these points often result in spurious matches. One way how to handle this problem is to set a threshold on the calculated correlation — if the correlation between the point and its best match is less than a certain value, we neglect such a match. Of course there is a tradeoff: the higher the threshold, the higher the chance that we also neglect some correct matches.

- *Rounding errors:* Since we match integer pixel positions to integer pixel positions, even the best match does not have to be the most accurate. This sometimes results in zigzag reconstructions

of straight lines. Strictly speaking, due to aliasing, the lines are already zigzag in the 2-D images, so the reconstruction is actually "correct". But a more desired result would rather be a straight 3-D representation of such a line. Similarly, we would like to suppress the variation in the representation of surfaces, which is also to a big extent caused by the discrete matching (this can particularly happen for distant objects where even a small inaccuracy in the match causes a big inaccuracy in the Z coordinate).

Again, due to discrete sampling (pixelisation) of the continuous reality, we loose information, so that this problem cannot be solved sufficiently. One approach is to calculate the correlation in subpixel accuracy, and estimate the matches as non-integer numbers. I did not implement this feature, and I discuss it in more detail in Section 4. Instead, I try to average out the Z coordinates for the pixels that lie next to each other. This indeed leads to improvements in the variation, and is much faster than the implementation of the subpixel accuracy. On the other hand, problems appear when pixels lie next to each other in the 2-D image but actually belong to different objects which are behind each other. These pixels should not be averaged out. But if such a situation occurs then the matching pixels for these two pixels will not lie close to each other in the second 2-D image. I therefore only average out pixels (their Z coordinates) that lie next to each other in one image, and very close to each other (max. 1 pixel between them) in the second image.

3 Experimental results

In this section, I will discuss the two approaches I took: first, automatic recognition of features (edge pixels) being matched, and second, marking features (corners) by hand, and their automatic matching. I will also point out some peculiarities of the given test images and problems that occurred.

3.1 Table scene

This scene contains a lot of objects, and is in my view the most complicated of all. There are some occlusions (e.g. the bottle on the left occludes the border of the paper pads, shadow of the bottle is occluded to a big extent in one image), shifts which distort the position of the foreground and background (making the usage of intensity correlation more difficult), and scaling (especially the stapler). I consider a dense disparity map (at least regarding the edges) more suitable for this type of images, since it is more illustrative.

Figures 3 4 show the whole process of 3-D reconstruction beginning with the edge map, over the epipolar lines for some important pixels, and finally the 3-D scene viewed from various angles. No manual adjustments were made in the reconstructed data. Note the layers created by the objects in the view from above. These layers correspond to the human perception of the depth in the scene.

Figure 5 shows the significant noise reduction when using reversed matching.

Figure 6 compares various combinations of edge detectors and threshold levels. Higher threshold levels, or the usage of the 2^{nd} derivative filter result in thinner lines, which might appear more accurate but some less striking edges might get lost. Lower threshold levels, on the contrary, create denser disparity maps, showing better the shapes of the objects. Denser disparity maps might appear rough and overcrowded, but this is a false impression — once the view is zoomed, the details come up. It depends on the purpose of the reconstruction which of the two approaches should be used. For general purposes the middle value of the thresholding parameter as most suitable.

3.2 Office scene

I choose the office scene as the second example which I examine in more detail, because this is a scene very different from the previous one. It is computer generated, and contains only straight lines. I also used two approaches — first, I marked the corners by hand in one image, and then let the program recognise the matches and connect the corners (I gave the program information about

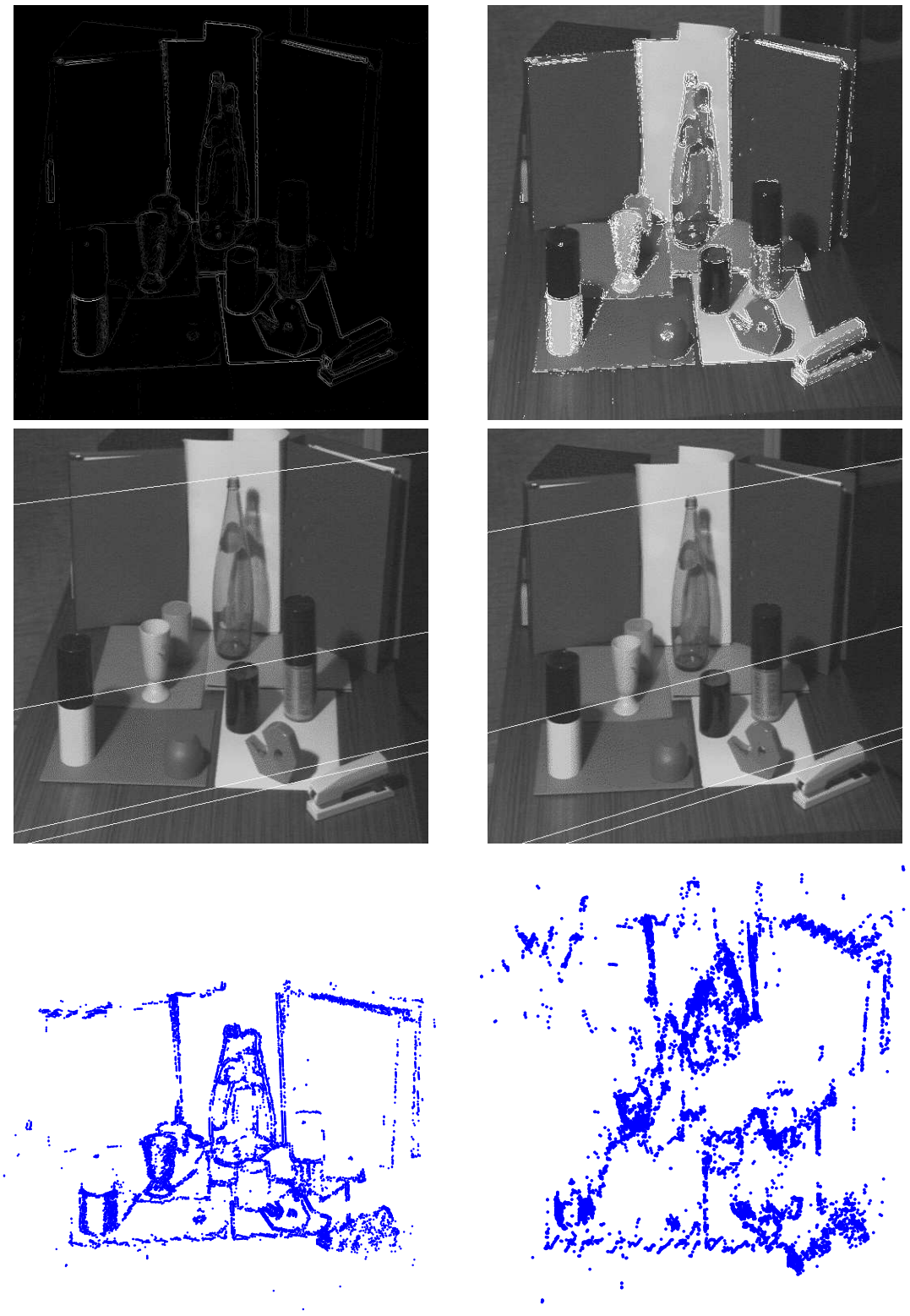

Figure 3: *Scene* ***Table****:* (parameters: edge threshold 20 (of max 255), 2nd derivative edge detector, smoothing matrix 3×3, correlation window 21×21, correlation coefficient threshold 0.5, reversed matching on) *First row left:* detected edgemap *First row right:* matched edge pixels *Second row left:* epipolar lines in the right image for the top of the bottle, top left corner of the white piece of the left bottle, top point of the stapler, bottom point of the perforator *Second row right:* epipolar lines for the reversed matching *Third row left:* reconstructed scene from the camera viewpoint *Third row right:* scene from the viewpoint elevated by 40°. *Continued in Figure 4*

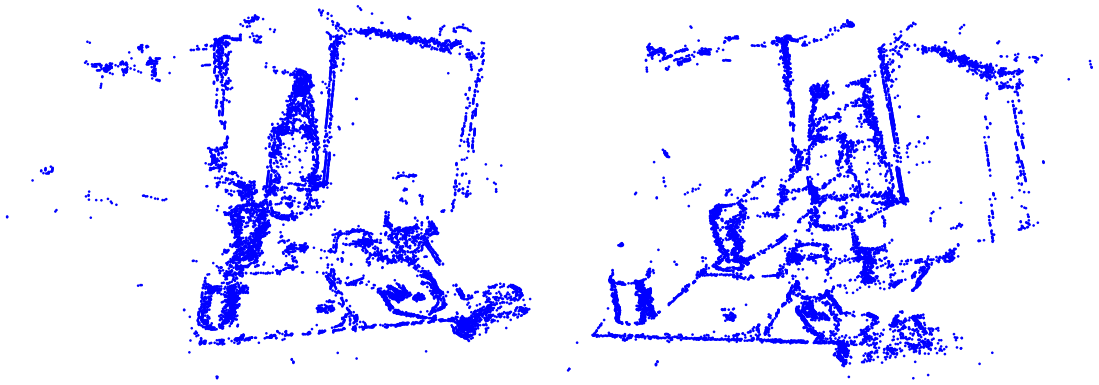

Figure 4: *Scene* ***Table****: Continued from Figure 3. Left:* scene rotated by 20° anticlockwise *Right:* scene rotated by 20° clockwise

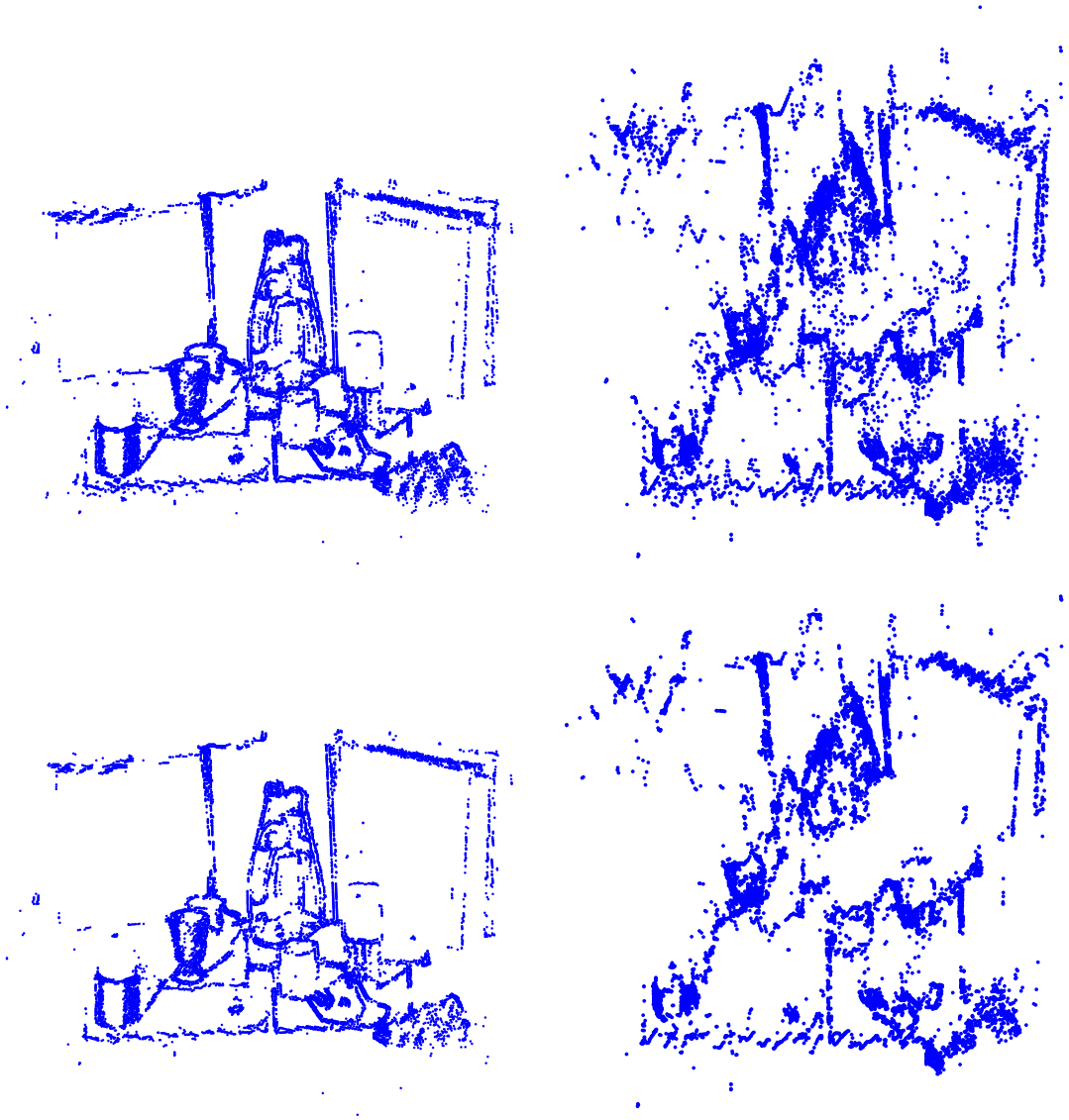

Figure 5: *Scene* ***Table****:* Influence of reversed matching. Left: scene from the camera viewpoint. Right: scene from the viewpoint elevated by 40°. Top row: reconstruction without using reversed matching. Bottom row: reconstruction using reversed matching. Observe the significant reduction in noise in the view from above when using reversed matching.

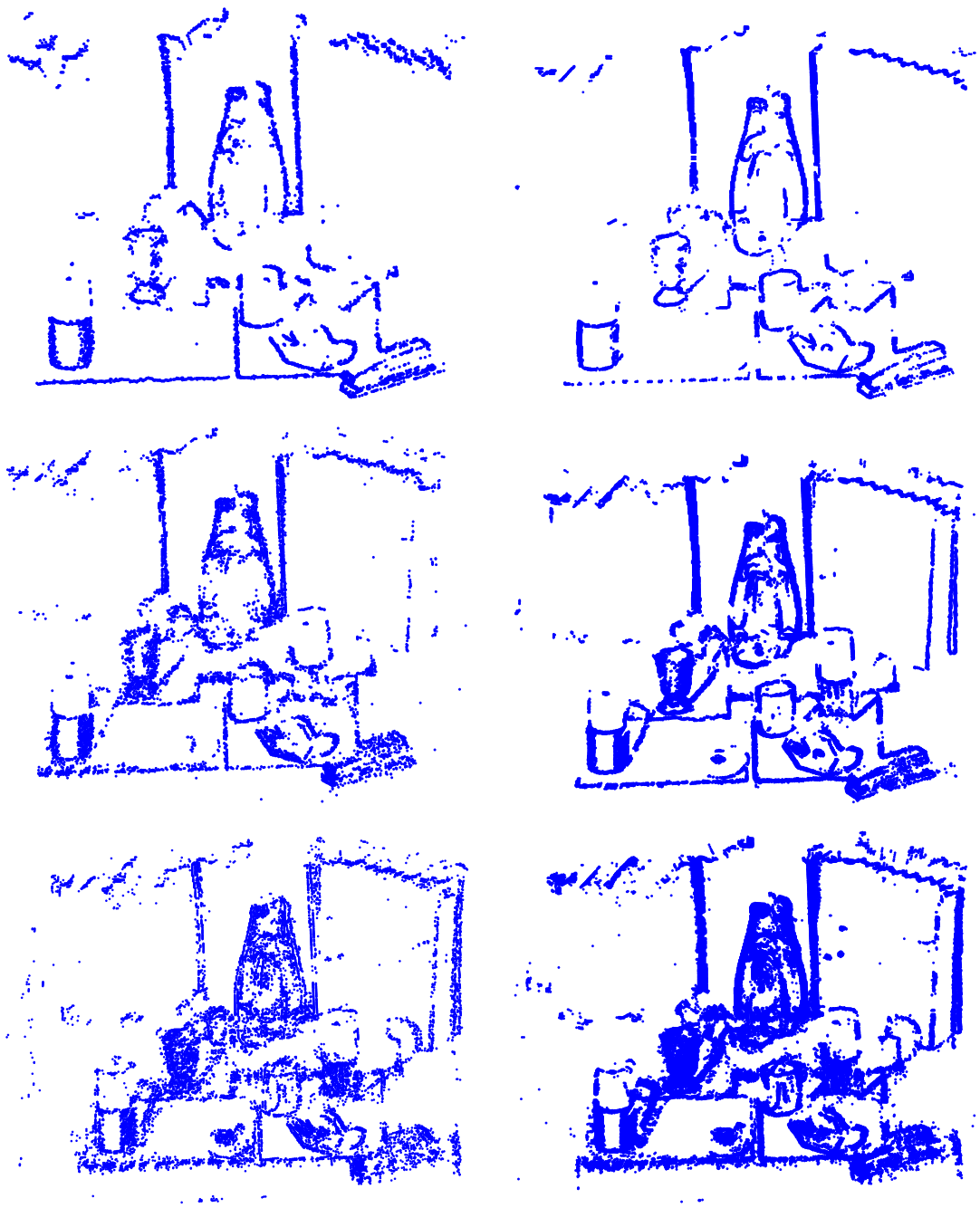

Figure 6: *Scene* ***Table****:* Various edge thresholding levels. *First row left:* Threshold 40 (of 255) + 2^{nd} derivative edge detector. *First row right:* Threshold 40. *Second row left:* Threshold 20 + 2^{nd} derivative edge detector. *Second row right:* Threshold 20. *Third row left:* Threshold 10 + 2^{nd} derivative edge detector. *Third row right:* Threshold 10.

which corners should be connected). Secondly, I used the approach which I used for the previous scene — using all edge pixels for matching.

The first approach creates straight lines, but is quite dependent on a suitable choice of the pixels which should be matched. If we are unfortunate and choose a point which is not matched correctly, we loose one corner of the object together with connecting edges. This cannot happen in the second approach when loosing one point is not a severe loss since there are thousands.

This scene also suffers of the fact that many edges are nearly parallel with the direction of the

epipolar lines. This makes the correlation method very susceptible to errors, since a very similar window occur many times. This is apparent from the image containing epipolar lines in Figure 7.

Also, many edges are not very distinct (chair, right side of the table, upper right corner of the window) — therefore it is adviceable to use a low threshold. I use smoothing and then thresholding at intensity 5 (of max. 255). Furthermore, since the rotation angle between the two cameras is quite significant, I use a smaller correlation window (11×11).

In Figure 7, I offer different views on the scene reconstructed using the approach with manually marked corners. The views are chosen in the way that it is apparent that points that should lie in one plane indeed are coplanar (especially the window, legs of the table, upper surface of the table).

There are also two points that have not been matched — one of the corners of the chair, and one of the legs of the chair. In the first case, the corner is difficult to detect in the second image since the edge is very blurred. In the second case, the leg of the chair is occluded, and therefore cannot be reconstructed.

Figure 8 shows the reconstruction using all edge points in the thresholded edge map. It is apparent that the nearly colinear epipolar lines and the upper edge of the room cause the algorithm to fail to recognise a consistent edge. The zigzag shape is due to the integer jumps in pixel matching.

3.3 Other scenes

The remaining scenes are less dificult than the *Table* scene, and the approach remains the same as in the previous cases. I therefore show here only the reconstructed scenes, viewed always from the left camera view, and then from a view where the reconstructed 3-D information becomes apparent. See Figure 9.

Also here, some problems occur. First problem is a repeating pattern alongside the epipolar lines (the epipolar line crosses several similarly looking edges and it is difficult to pick the right one). This is the case in the *Testpatt* scene, or in the *Room* scene. Especially in the *Room* scene, where epipolar lines crosses similar edges, and the scene is otherwise not very textures, the algorithm gets into significant trouble, and I the results show that the *Room* scene is the worst reconstructed scene of all.

Secondly, the rounding errors appear again, and this is most apparent in the *Testpatt* scene.

4 Critical analysis and possible improvements

In my opinion, the algorithm, taken into account all its limitations described above, works quite well, and the reconstructed scenes are well visible. The algorithm definitely works better on textured scenes with a variety of objects, shapes, and edges. The problems that occurred are already described in Section 3. To summarise, the three most important are:

- Lack of textures or the presence of only sparse and similarly oriented lines makes the correlation approach inefficient. This is especially apparent in the *Room* scene.
- If the epipolar lines are approximately parallel with some edges, it is difficult to choose which edge point is the correct match. This is most visible in the *Office* scene.
- Finally, discrete matching produces some zigzag lines, which is most apparent in the *Testpatt* scene.

Of course, one way of improving the performance of the program is to improve the currently implemented algorithms, e.g. improving the correlation measure so that more relevant matches are chosen, or altering the edge detector. Here I want to suggest and explain some of possible improvements that should increase the accuracy and reliability of the algorithm, and are different from those implemented in my program.

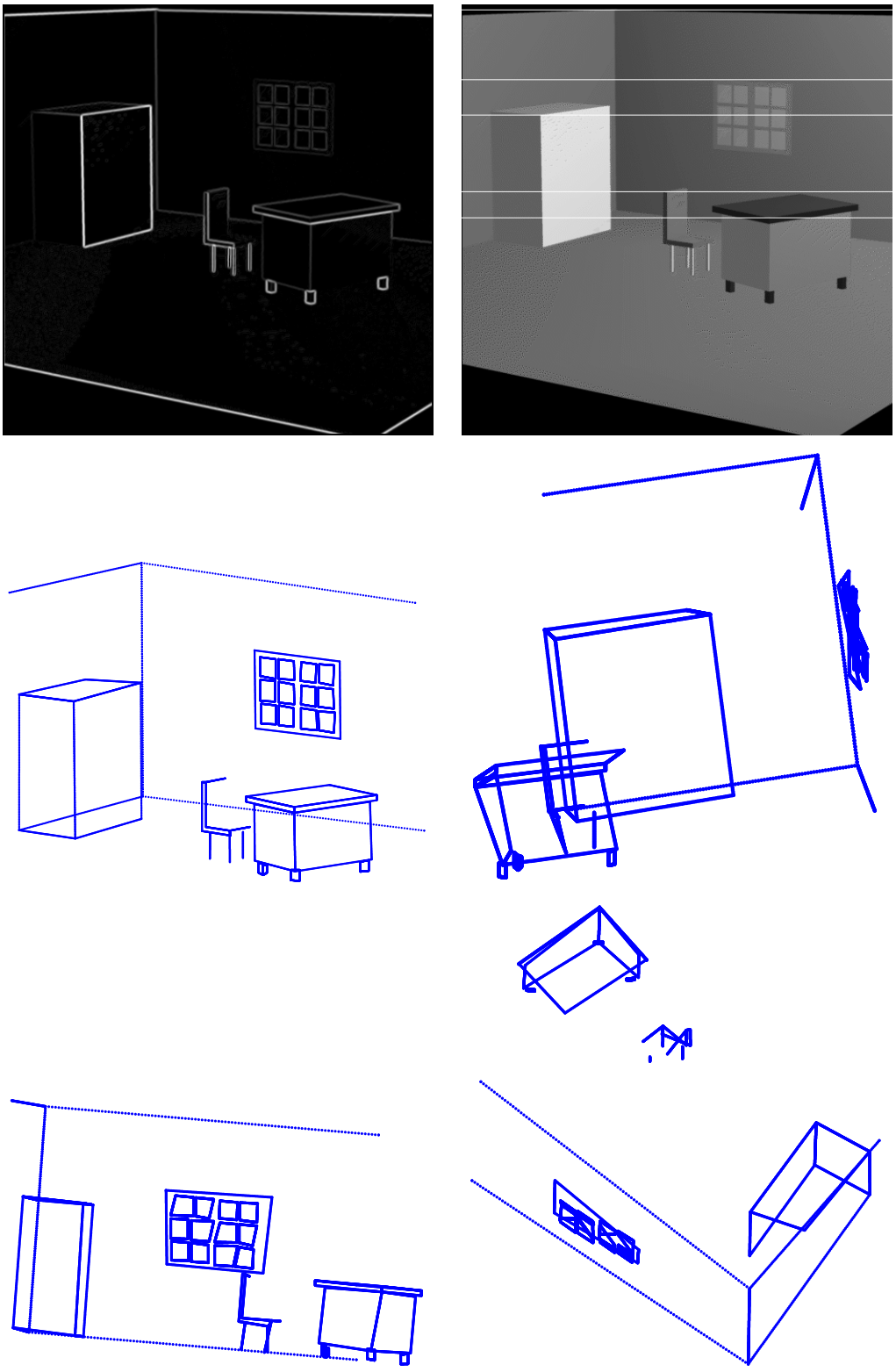

Figure 7: *Scene* ***Office****:* (parameters: edge threshold 5 (of max 255), 2nd derivative edge detector, smoothing matrix 3×3, correlation window 21×21, correlation coefficient threshold 0.5, reversed matching on) *First row left:* detected edgemap *First row right:* epipolar lines for the left top corner of the window, front top corner of the wardrobe, top left corner of the chair, and top front corner of the table *Second and third row:* various views on the scene: camera view, rotated clockwise 53°, rotated anticlockwise 30° and lowered 10°, rotated 180° and elevated 80°. *Note:* the slope in the second and third view is given by camera view and the impossibility to rotate the scene in the appropriate direction in Matlab, it is not a flaw of my algorithm.

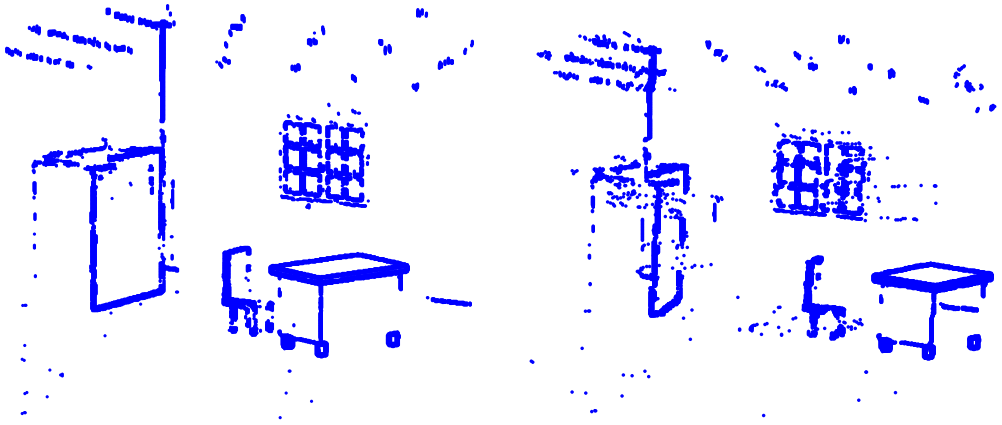

Figure 8: *Scene **Office**: Reconstruction using all edge points* (parameters: edge threshold 20 (of max 255), 2nd derivative edge detector, smoothing matrix 3×3, correlation window 21×21, correlation coefficient threshold 0.5, reversed matching on) *Left:* camera viewpoint. *Right:* rotated 20° anticlockwise.

- *Sub pixel accuracy:* When reconstructing straight lines, inaccuracies occurred due to the fact that the pixels are matched discretely, i.e. integer pixels on integer pixels. It would be an improvement to implement sub pixel accuracy, so that the pixels could be matched on non-integer pixel positions in the second image. Non-integer results are of course of no problem for the triangulation. But a bigger problem is a severe increase in computatitonal demandingness. Another problem is the model we use for the interpolation — a simple linear interpolation of the intensities might not be sufficient, and we are of course only guessing the model.

- *Handling problems with finding the correct match on the epipolar line:* A possible improvement would be if we could pre-estimate the position of the matching point on the epipolar line. If we could for instance assume that the modelled surface is continuous, then the left-to-right order of the matching pixels has to be fulfilled in both images. But this assumes making additional assumptions about the scene — it can be useful in the case of *Room* or *Testpatt* scenes, but definitely in the case of more complicated scenes like *Table*. However, for instance the *Testpatt* scene is very suitable for such an assumption, since here, the correlation method suffers by the fact that the epipolar lines cross a very similar pattern several times. Here, the left-to-right order assumption would be of big help.

- *Object detection identification* With the approach I am using, a visually easy understandable disparity map is produced. But if we want to recognise and reconstruct more complex objects, it would be more suitable to work with more sophisticated features like lines, corners, or uniform regions. These more sophisticated methods are however rather suitable when we have some prior information about what type of objects we might expect (like indoor scenes with long straight lines). They might prove faster than correlation based methods. They produce sparse disparity maps, which are visually not so easy to understand for a human, but might be very useful for automatic object reconstruction.

  An implementation of such a feature based solution would require significant changes in the program (choice of significant and sparse matching points, matching features like lines etc.), and would be definitely more difficult than the correlation based approach.

5 Program listing

I submit the program listing in a separate file.

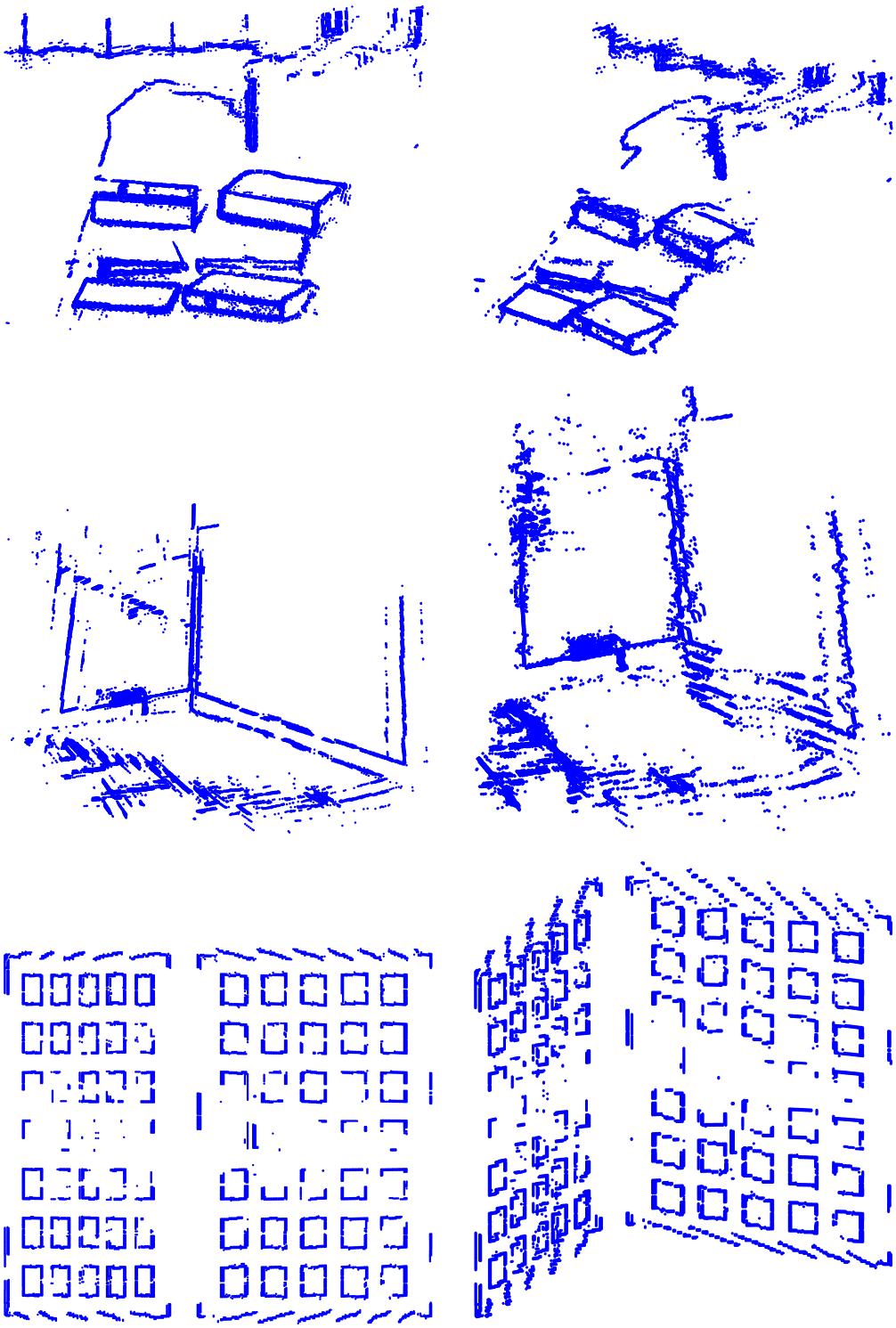

Figure 9: *Remaining scenes: Left:* original camera view. *Right:* rotated view so that 3-D information is apparent.