# SEMI-AUTOMATIC ORIENTATION OF IMAGES WITH RESPECT TO A POINT CLOUD SYSTEM

David Novák

Institute of Geodesy and Photogrammetry, ETH Zurich, 8093 Zurich, Switzerland – dnovak@student.ethz.ch

**KEY WORDS**: Adjustment, automation, calibration, detection, matching, orientation, programming

**ABSTRACT:**

This paper presents the results of a master thesis in which it was tried to orient a set of images of an object to a point cloud of the same object. As test object the "Semper Sternwarte" in Zürich was used. As data sets a dense point cloud from a laser scanner and a sparse point cloud obtained by photogrammetric means, as well as the orientation of the images were used. Precisely orientated images with respect to a point cloud can be used to incorporate an edge-constrained triangulation techniques, blunder and outlier detection, which leads to an overall better representation of a 3D model. This paper looks at a specific semi-automated work flow using self-programmed tools and tries to determine whether the work flow is suitable for this task or not.

## 1. INTRODUCTION

Registration of images taken by CCD array cameras with respect to point clouds obtained by active sensors like laser scanners is necessary prior to the integration of the both data sets (texturing, overlaying, gap filling, etc). This is mostly done by the use of some targets visible in both data sets or by preliminary laboratory calibration, while the camera is mounted on the active sensor (e.g laser scanner). Nevertheless, there are various cases in which there are no common targets between the two data sets or the pre-calibration is not possible due to either using not mounted cameras or different acquisition times. Additionally, in case common targets are used, the mapping function that maps the point cloud to the image coordinate system is obtained over targets, thus the accuracy of the function is very much dependent on the distribution of the targets. More over, in case of pre-calibration, the calibration parameters may not remain stable if the camera has been dismounted for some movement purposes. Therefore a kind of on the job registration would be useful. The goal of this project is to perform the registration of images taken by CCD array cameras and the point cloud of the same object (e.g. obtained by an active sensor like a laser scanner) to one coordinate system, while approximate registration parameters are available. By orienting images with respect to an existing point cloud system it should be possible to see what improvements can be done when combining point cloud data obtained with different measurement techniques.

We assume that the rough orientation parameters of the images are available in an arbitrary coordinate system. These parameters serve as approximations for an automatic process that incorporates a point cloud obtained by a laser scanner or a similar active sensor. By back projecting the points of the laser scanner point cloud into the images, the orientation parameters can be improved, by iteratively moving the two point clouds in a way that the distances between those two become minimal. As a result more precise orientation parameters are obtained, which in turn can be used to improve the existing point cloud with respect to edges and outliers that usually are present in point clouds from laser scanners or similar active sensor devices.

The whole process is semi-automatic because the initial approximations of the orientation parameters are obtained with manual measurements while the improvement is then done with a fully automated algorithm.

## 2. THE TASK

The task of this project is divided into two parts. First a manual pre-processing is done by checking the data and then registering the two data sets into one. Then a set of C++ tools is programmed to accomplish the second part as seen in figure 1.

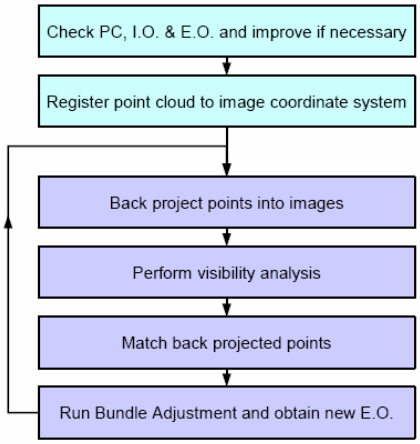


Figure 1: Suggested work flow. Pale green marks the manual part. Dark blue the automated one.

The data is divided into two data sets. One set is a dense point cloud that was obtained with a laser scanner. The other dataset consists of images taken of the object as well as measurements in the images. With the measurements an exterior orientation has been calculated with photomodeler and as of such a sparse point cloud has been obtained.

In a first step the point cloud has to be checked if gross outliers are present. If this is the case those have to be removed. Additionally the interior and exterior orientation parameters

have to be checked and improved if they are not good enough. Then the laser scan point cloud has to be registered to the image coordinate system. This can be done with a surface matching algorithm like ICP or LS3D – however if one of the point clouds is very sparse (usually the one obtained photogrammetrically), then the mentioned algorithms will fail because they cannot find a corresponding surface.

After the first two steps, which can be considered pre-processing steps, the main processing chain is started.

The automated part consists of three steps:
- Backprojection of the laser points into the images
- Visibility analysis to reduce probability of mismatching
  Matching of the points
-

Two C++ tools are written which incorporate these three steps – the first tool is performing back projection and visibility analysis while the second one is doing the matching part.

Once the matching is finished, the matched points can be used as measurements for the bundle adjustment, which is performed with an external tool. After the bundle adjustment is completed successfully a new interior and exterior orientation of the images can be obtained. If the new orientation is acceptable the process can be stopped. If it is not then the points have to be projected back into the newly orientated images and the same procedures have to be performed again until the results are satisfactory.

In the end the orientation of the images should be good enough to allow further processing like edge-constraint triangulation and blunder removal.

## 3. PRELIMINARY TASKS

### 3.1 Import procedure

In order to be independent from any software the input and output files for the interior and exterior orientation are described in XML. As mentioned in the introduction the exterior orientation has been obtained with photomodeler. A simple conversion tool converts the photomodeler file into an XML file, that incorporates for each image a separate tag with its interior and exterior orientation. This has the advantage, that the data structure does not need to be tailored towards a specific orientation/bundle adjustment program.

For the laser scanner point cloud a simple X,Y,Z-structure is chosen which is stored in a plain-text file. This is done because an XML-File would create a considerable data overhead and wouldn't really introduce a benefit in return. For each point a new line is defined with its X, Y, and Z coordinates separated by a space character.

### 3.2 Data check

The interior and exterior orientation as well as the point cloud have to be checked before any further processing is done. The orientation parameters have to be precise enough so that the matching algorithm can work correctly. It is however not easy to check if the orientation parameters are correct or not. Especially if the camera used is no longer usable it can be almost impossible.

The point cloud has to be checked for outliers. This mainly to reduce the probability of mismatching later on. This can be done easily by using a 3D modelling tool.

### 3.3 Registration

In order to be able to back project the laserscanning points into the images, it is important to have both the laser scan and the photogrammetric point cloud in the same coordinate system. This has to be done in a manner, that the exterior orientation of the images is still usable in order to be able to backproject the laser scan points correctly into the images in the first step. This is achieved by keeping the photomodeler point cloud fixed and rotating and scaling the laser scanner cloud into the photomodeler system.

Point clouds are registered by minimizing the sum of squares of the Euclidean distances between the surface scans. Different algorithms are available like ICP or LS3D. However all of them need dense point clouds to work properly.

In this case only the laser scan point cloud is dense. The point cloud obtained by photomodeler is very sparse. When trying to register the two point clouds with Geomagic's ICP algorithm the problem arises, that the photogrammetric point cloud is so sparse that it does not really offer any surfaces to be matched.

Because of this a 3D Helmert transformation is used as seen in equation 1. Eight points are selected in each point cloud and afterwards the six Helmert Transformation parameters are obtained. With these the laser scanner point cloud is registered into the camera system.

$$\begin{bmatrix} X_{fix} \\ Y_{fix} \\ Z_{fix} \end{bmatrix} = \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix} \begin{bmatrix} 1 & m & -r_z & r_y \\ r_z & 1 & m & -r_x \\ -r_y & r_x & 1 & m \end{bmatrix} \begin{bmatrix} X_{Free} \\ Y_{free} \\ Z_{free} \end{bmatrix} \quad (1)$$

This procedure is not perfect since there might be some systematic errors regarding the X,Y,Z-positions and the rotation angles. Especially selecting corresponding points between a dense and a sparse point cloud can prove to be very difficult. However in this case the only other possibility would be to densify the photogrammetric point cloud, which would take too much time. So the downside of the 3D Helmert transformation is accepted.

## 4. BACK PROJECTION AND VISIBILITY ANALYSIS

### 4.1 Back projection

Once the point clouds have been registered, the laser scan points can be back projected into the images, since the laser scanner points are now in the object coordinate system of the images. The back projection is performed with the collinearity equations:

$$x_i = x_0 - c\frac{d_{11}(X_i - X_0) + d_{21}(Y_i - Y_0) + d_{31}(Z_i - Z_0)}{d_{13}(X_i - X_0) + d_{23}(Y_i - Y_0) + d_{33}(Z_i - Z_0)} \quad (2)$$

$$y_i = y_0 - c\frac{d_{12}(X_i - X_0) + d_{22}(Y_i - Y_0) + d_{32}(Z_i - Z_0)}{d_{13}(X_i - X_0) + d_{23}(Y_i - Y_0) + d_{33}(Z_i - Z_0)} \quad (3)$$

Since all of the parameters are known the equation is easily performed and the pixel positions of the 3D points in the images are obtained.

### 4.2 Out-of-plane check

Since the collinearity equations are performed for all the points there will be cases where the back projected points will not lie on the image plane. Such cases are easily detected by checking the obtained x- and y coordinates from the collinearity equation. If they are negative or are larger than the image dimensions, then they are not on the image and thus can be flagged as not visible. By doing so memory space and calculation time is saved for later steps.

### 4.3 Pixel correction

Once the visible points are back projected their position needs to be corrected. This is because of systematic errors of the camera. These errors are modelled by the additional parameter set from Brown. They include the interior orientation, scale- and shear factor, symmetrical radial lens distortion and decentering lens distortion.

The collinearity equation is then extended with the Brown Parameter model which results in:

$$f_x = x - x_0 \quad _x \tag{4}$$

$$f_y = y - y_0 \quad _y \tag{5}$$

Since $f_x$ and $f_y$ are non-linear functions they have to be linearised and then adjusted to obtain the correct value. As first approximations the back projections from the collinearity equation are used for the adjustment.

The corrections of the solution vector are added to the pixel positions until convergence is reached.

### 4.4 Hidden Point Removal

Before the matching of the points is performed occluded points should be removed. This has two advantages:

– The calculation time and the amount of memory for the matching procedure is reduced greatly
The probability of mismatches later on is reduced
–

The main problem is that at this stage we're still facing points and as of such common techniques for Hidden Surface Removal like z-buffer, binary space partitioning or similar are not applicable since there are no surfaces to process.

Because of this a “Hidden Point Removal”* algorithm is used. This algorithm takes a back projected point and draws a buffer around it in image space. It then checks if any other points lie within that buffer. If this is the case the 3D distance is

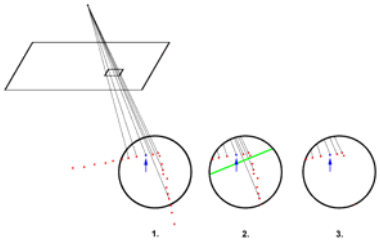


Figure 2: HPR visualized. 1. Shows how all points in the buffer around the blue point are selected. In 2 the mean distance is calculaed for the buffer. In 3. all the points that are farther away than the mean distance are thrown out

Figure 4: Close-up of a back projection with HPR

Figure 3: Close-up of a back projection without HPR

### 4.5 HPR in pseudo code

In a first step the algorithm has to be constructed in a way that it loops through all the points. Additionally for each point the point, cloud has to be looped through again so that for each point it can be checked, which points are inside the actual buffer and which aren't. This means that the point cloud is run through at least $n^2$ times which of course is very inefficient.

```
1               while (pointnr != amount of points)
2               {
3                  if (not flagged as not visible)
4                  {
5                     while (pointnr2 != amount of points)
6                     {
7                        if (not flagged as not visible)
8                        {
9                           if (projected point inside buffer)
10                          {
11                             Calculate obj distance + mean +
write to buffer
12                          }
13                       }
14                    pointnr2++;
15                    }
16                    while (pointnr3 != amount of points)
17                    {
17                       if (projected point inside buffer)
18                       {
19                          if (point distance > mean distance)
20                          {
21                             flag point as not visible
22                          }
23                       }
24                    pointnr3++;
25                    }
26                }
27             pointnr++;
28             }
```

In line 1 the first loop is initiated. It selects the first point and checks if it hasn't been flagged as not visible. This is done to

* From now on called HPR

reduce calculation time. If the point is visible then it is used for the distance check. At line 5 another loop is started. That one is used to determine which points are inside the buffer - the check is done at line 9. Prior to this another check is done whether the point has been already flagged as not visible or not – again to reduce calculation time.

Once a point lies inside the buffer its distance is calculated as well as the mean distance of the buffer as seen at line 11. This is done for all the points in the point cloud.

When the second loop finishes a third loop is started at line 16. This loop goes – again – through all the points. This actually wouldn't be necessary – instead a more elegant solution could be found.

In any case the points inside the buffer have their object space distance compared to the mean object distance of the buffer at line 20. If the point distance is bigger than the mean distance then the point is flagged as an outlier.

The first buffer is fully processed when line 27 is reached. Once there the point iterator is increased by 1 and thus the next point is selected for processing.

## 5. MATCHING

Matching is performed to obtain more measurements which then can be used to get new orientation parameters. In this case the points that have been back projected and processed are used as points to be matched. Manually one image is set as template and another one as search. The back projected points in the template image are the points that have to be matched in the search image.

Different matching techniques are available nowadays. In this case an area based matching algorithm – Least Squares Matching* – has been chosen. The LSM algorithm is used in its simplest form. No additional constraints like geometric constraints or multiphoto constraints have been used.

### 5.1 Least Squares Matching

As described in detail in (Gruen, 1996) matching between the two regions is established when $f(x,y) = g(x,y)$.

Because the images have not been taken from the same position, and noise can be present in one or both images the true error vector $e(x,y)$ is introduced. This results in the following equation:

$$f(x,y) - e(x,y) = g(x,y) \tag{6}$$

The location of the function values of $g(x,y)$ must be determined to acquire the matched position. This is achieved by minimizing a goal function which measures the distances between the grey values in the template and search window.

Equation 6 can be considered a nonlinear observation equation which models the vector of observations $f(x,y)$ with a function $g(x,y)$ whose location has to be estimated in the search photograph. The location is described by shift parameters $x$, $y$ which are counted with respect to an initial position of $g(x,y)$, the approximation of the conjugate picture region $g^0(x,y)$.

In order to account for a variety of systematic image deformations and to obtain a better match, image shaping parameters are introduced additionally to the shift parameters.

If the local terrain surface patch is a plane in sufficient approximation, then the affine transformation model is considered to be sufficient and used as a full parameter set. Radiometric parameters are left out.

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a & b \\ c & e \end{bmatrix} \begin{bmatrix} x_{sw} \\ y_{sw} \end{bmatrix} \begin{bmatrix} e \\ f \end{bmatrix} \tag{7}$$

In order to be able to apply the least squares approach the function $g(x,y)$ has to be linearised. Upon performing the adjustment a solution vector is obtained:

$$A^T P A^{-1} A^T P l = \begin{bmatrix} a & b & c & d & e & f \end{bmatrix}^T = x \tag{8}$$

The solution vector $x$ is used to alter the affine transformation parameters. With the new parameters a new pixel position of the search window can be determined. However after the affine transformation the pixel position will most likely not be an integer value. As of such the new grey value has to be interpolated. In this case this is done by means of a bilinear interpolation. This is done mainly because it is easier to implement than cubic convolution interpolation, while offering superior image quality over nearest neighbour interpolation. Once the new grey values have been obtained for the new pixel positions the whole adjustment process can be repeated again.

With the newly acquired grey value the adjustment can be run again until each of the elements in the solution vector fall below a certain limit.

$$|a| < limit_a \quad |b| < limit_b \quad |c| < limit_c \quad |d| < limit_d$$
$$|e| < limit_e \quad |f| < limit_f$$

It is possible that the solution vector converges only very slowly, oscillates, converges to a false solution or that it doesn't even converge to a solution. These cases indicate severe matching problems and require further attention.

The precision of the estimated parameters can be obtained with the covariance matrix:

$$K_{xx} = \sigma_0^2 Q_{xx} = \sigma_0^2 A^T P A^{-1} \tag{9}$$

The precision of the matching location in the image is described with the standard deviation of $e$ and $f$.

$$\sigma_x = \sigma_e = \sigma_0 q_e^{\frac{1}{2}} \tag{10}$$

$$\sigma_y = \sigma_f = \sigma_0 q_f^{\frac{1}{2}} \tag{11}$$

By obtaining the Covariance matrix it is possible to get the standard deviations of the shift parameters as described in equations 9, 10 and 11. The minimum, maximum and mean

* From now on called LSM

values of the shift parameters after matching two images can be seen in the following table:

| | Min | Max | Mean |
|---|---|---|---|
| $\sigma_{\Delta x}$ Img_30-31 | 0.0019 | 0.1006 | 0.0106 |
| $\sigma_{\Delta y}$ Img_30-31 | 0.0020 | 0.2152 | 0.0254 |

Table 1: Standard deviations of the shift parameters in pixel

While the values itself seem very good they should be taken with a grain of salt. For once the reliability is not included which means that mismatches are not visible in this case. Another problem is that certain patterns require the deletion of one or more affine transformation parameters – if this is not done then the match can be unreliable.

In this project only two images could be matched, because the others had poorer approximations of the point positions and as of such the matching failed there.

Problems can also arise in areas that do not have enough contrast / texture to produce reliable matches. The matching algorithm might converge on surfaces with no texture but the result will not be reliable. In order that the algorithm does not match these areas, the template window has to be checked if the signal produces enough contrast for matching.

In this case this is done by calculating the standard deviation of the grey values for the template window and defining a threshold value. If the standard deviation of the grey values is bigger than the threshold, then it will be used for matching – else it will be marked as "non-matchable" due to poor contrast.

By marking the matched points in the images it is possible to visually check whether the matching actually works or not. In this case it seems that the algorithm has been implemented correctly since in most cases the matches seem reasonable.

Figure 5: Correctly matched points on a random target

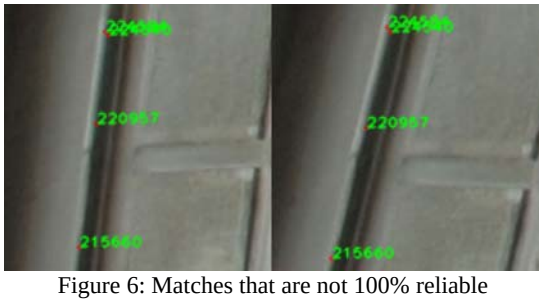

Figure 6: Matches that are not 100% reliable

### 5.2 LSM in pseudo code

```
1               while (pointiterator != amount of points)
2               {
3                  if (point number in template == point number
in search)
4                  {
5                     read template gray values into array
6
7                     if (stdDev of template GV-array > limit)
8                     {
9                        read search gray values into array
10                       read gradients and fill A matrix
11                       Perform adjustment and get new
parameter values
12                    }
13                    while (convergence criterion not reached)
14                    {
15                       perform affine transformation
16                       perform bilinear interpolation
17                       store new gray values into search array
18                       fill A matrix and perform adjustment
again
19                    }
20                 if (convergence reached)
21                 {
22                    write final values into XML file
23                 }
24                 if (maximum amount of iterations reached)
25                 {
26                    No convergence reached – nothing is
written
27                 }
28                 }
29              }
```

In a first step the template and search images have to be selected. This is done manually by asking the operator to input the desired image numbers on the command line. Once this is done the whole point cloud has to be iterated through as seen in line 1.

The first point in the laser scan point cloud is selected and its number is compared to the numbers stored in the template and search image. Each back projected point has a point number associated from the point cloud file and as of such if the point number in the template and search image are identical, they will be used for matching. This check is performed at line 3.

When the point numbers match, the grey values in the template image are read out and stored in a vector. The values are read out column-wise in x-direction.

Afterwards the standard deviation of the vector is checked – if it is below a certain value then this point will not be used for matching because the contrast is too low to allow for reasonable matching. The check is done at line 5.

If the standard deviation is accepted the grey values in the search image are read out in the same manner like for the template image. The check is performed at line 7 and the grey values are read out at line 9.

Additionally the gradients and the partial derivatives of the affine transformation are obtained and the values filled into the design matrix. With these it is possible to perform the first adjustment and obtain the solution vector as seen in lines 10 and 11.

Afterwards the convergence loop is started at line 13. While this loop is not that much different than the loop starting at line 7, the two were separated. The loop only differs from the one at line 7, in that an affine transformation is performed and afterwards a bilinear transformation (lines 15 and 16). These two calculations are not done in the first loop which helps reducing the calculation time a bit.

Once the bilinear interpolation has been performed the same procedure is performed: New grey values for the search image are obtained and the adjustment is run again (lines 17 and 18).
When convergence is reached the final pixel position of the point is written into a XML file. If the maximum amount of iterations has been performed then nothing is written. Both checks are performed at line 20 and 24 respectively.
After that, the point iterator is increased by 1 and it is checked if the new point is visible in both images. If so the whole procedure is done again until all the points in the point cloud have been processed.

## 6. ADJUSTMENT

Once the points have been matched, a visual check can be done to determine whether the algorithm works correctly, or if there are some serious errors with the implementation. If the matching is satisfying then the bundle adjustment can be performed.
The bundle adjustment is a procedure to adjust an arbitrary number of images, while considering measured image points and if available geodetic observations. With homologous points the images are linked into a model where the object can be reconstructed in full 3D. It is possible to set a relation to a higher-ranking coordinate system by introducing a minimal amount of control points.
The most important condition is that all the homologous image rays have to intersect in their respective object point.
Bundle Adjustment itself has been performed in Australis. However any Bundle Adjustment program could be used, or the bundle adjustment could be newly programmed. The Outputfile just has to be written into a file structure that the Bundle Adjustment programm can read, which can be done with a simple export filter.
One problem that becomes apparent is that the huge amount of points poses a problem. Australis has the rather interesting behaviour to increase the import time drastically the more points are imported. It also has the tendency to crash when using large amounts of points.

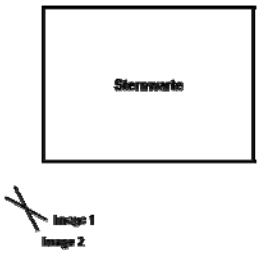


Figure 7: Top view of the poor image distribution that are used in the Bundle

However since only two images could be matched the amount of points can be reduced to reasonable levels. It has to be noted though that with more images, the point cloud would've been larger and as of such the problem that Australis simply fails can surface.

Because of the poor image distribution as seen in figure 7, it is not expected that the Bundle Adjustment actually can achieve any improvements regarding the orientation.

After the bundle has been completed successfully, the new exterior orientation parameters for the images can be obtained. However due to the poor image distribution those values are not really meaningful.
If however the matching was successful between multiple images the new orientation can be used further – either as the final result if it is good enough, or as next approximation to repeat the automated work flow until the result is satisfying.

## 7. CONCLUSION

Registering images and a point cloud into a common coordinate system is feasible in a semi-automated approach. The suggested workflow as seen in figure 1 seems to be a viable solution to orient images precisely to a point cloud. However it is strongly suggested to use cross correlation matching in a first iteration in order to obtain better approximations, then in a second iteration use LSM. If this is not done, then the orientation process can fail, because the approximations for LSM are not good enough to allow for correct matching, as seen in this project.

While implementing cross correlation would not take a lot of time, it sadly could not be done in this project since the time frame of 16 weeks was too short to allow for an implementation of both LSM and cross correlation.

Of course improvements could be done for each of the modules. The HPR algorithm for example could be sped up tremendously by using a slightly different approach.

The matching could be improved by implementing the more sophisticated methods of LSM like multiphoto geometrically constrained matching to reduce the probability of mismatches. Unreliable matches could be filtered out by applying certain test methods that check if the pattern in the patch is actually distinct or if it allows for multiple solutions in the neighbourhood.

Additionally computational aspects could be considered to improve the speed of the algorithms. Optimizing for multicore / multi processor systems would be one aspect – another to let the code run on the Graphics Processing Unit which would be a lot faster than to run it on the Central Processing Unit.

All in all it can be said that the suggested workflow can produce precisely orientated images with respect to the point cloud if a little more time is invested into it. The work flow is an iterative process – it however is difficult to say how many iterations would be needed in general. It is entirely possible that only two iterations would be enough to orient images to a point cloud system – with the first iteration using cross correlation matching and the second LSM.

## ACKNOWLEDGEMENTS

I'd like to thank everyone.

## REFERENCES

American Society for Photogrammetry and Remote Sensing, 2004, *Manual of Photogrammetry – fifth edition*, ASPRS, Bethesda.

Baltsavias, E., P., 1991, Multiphoto Geometrically Constrained Matching, *IGP ETH Zürich Mitteilungen Nr. 49.*

Carosio, A., 2003, *Fehlertheorie und Ausgleichung – Band 1 + 2*, ETH Zürich.

Cattin, P., C., 2006, *Lecture notes "Introduction into Signal and Image Processing"*.

Gruen, A., 1996. Least squares matching: a fundamental measurement algorithm. *In: K. Atkinson (ed.), Close Range Photogrammetry & Machine Vision*, Whittles, pp. 217-255.

Grün, A., 2006/07, *Lecture notes "Digital Photogrammetry I + II"*.

Grün, A., Remondino, F., 2007, *Lecture notes "Close-Range Photogrammetry"*.

Grün, A., Remondino, F., Urban, C., 2007, *Lecture notes "Photogrammetry and Machine Vision"*.

Kraus, K., 1996, *Photogrammetrie Band 2 – verfeinerte Methoden und Anwendungen*, Dümmler, Bonn.

Nipp, K., Stoffer, D., *2002, Lineare Algebra*, VDF, ETH Zürich.

Luhmann, T., *2000, Nahbereichsphotogrammetrie*, Wichmann, Heidelberg.

Remondino, F., 2006, Image-Based Modeling for Object and Human Reconstruction, *IGP ETH Zürich Mitteilungen Nr. 91*.

Simon, K., 1996, *Programmieren für Mathematiker und Naturwissenschaftler*, VDF, ETH Zürich.

Sperb. R, 2004, *Analysis I*, VDF, ETH Zürich.