BCRF - Binary-Coded Illumination Range Finder Reimplementation

Tomáš Pajdla
Katholieke Universiteit Leuven
ESAT, Kardinaal Mercierlaan 94, B-3001 Leuven
Tomas.Pajdla@esat.kuleuven.ac.be

April 20, 1995

Abstract

This reports describes the implementation of "one shot" active range finder based on binary coded illumination. First, a concise description of the princples is given and a few examples of its performance are presented. Second, the reimplementation of the system at Arrows MI2 ESAT KU Leuven is documented.

1 Introduction

One shot range finders attract attention since 3-D scene analysis has become widely popular. Besides passive techniques relying on stereo algorithms various kinds of active 3-D measurement devices were developed. See [Jar93, Vuy87] for overview of the whole gamut of methods.

Vuylsteke and Oosterlinck [VO90] have invented attractive idea of local redundant ray index encoding. Based on the idea working experimental setup was developed and tested with promising results. In short time some successful applications of the system appeared.

This text documents current implementaion of Binary-Coded Illumination Range Finder (BCRF) at Arrows group, MI2, ESAT, K.U.Leuven. It is organized as follows. In the section 2 the principles of the method are presented and a few examples of system's performance are given. The section 3 comments on previous applications and implementations of the original Vuylsteke's system. The rest of the text documents the reimplementation of the system.

2 Principles

The original method has been invented by Vuylsteke and Oosterlinck [Vuy87] [VO90] in 1987. At the same time, the first experimental setup was developed and tested with promising results. Before going to details I shall emphasize the important features of the system:

- fast acquisition suitable for scanning dynamic scenes based on a single image,
- high robustness and the locality of a measurement,
- simple, robust and cheap experimental setup, and
- the ability to provide both 3-D surface and its gray level image.

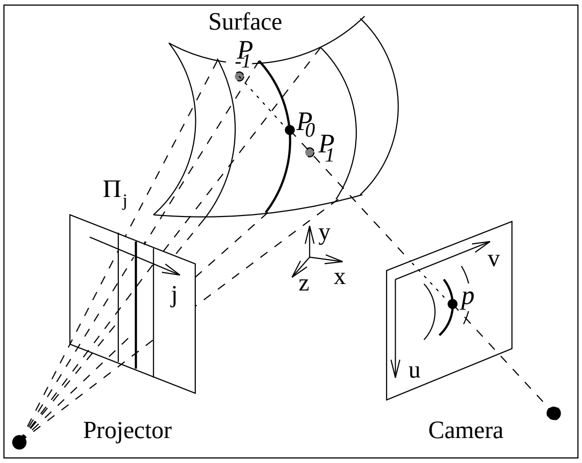


Figure 1: Ray Identification problem. The image point p can rise from a number of 3-D surface-plane-ray intersections. The index j of the plane intersecting the surface at the point P_0 has to be found.

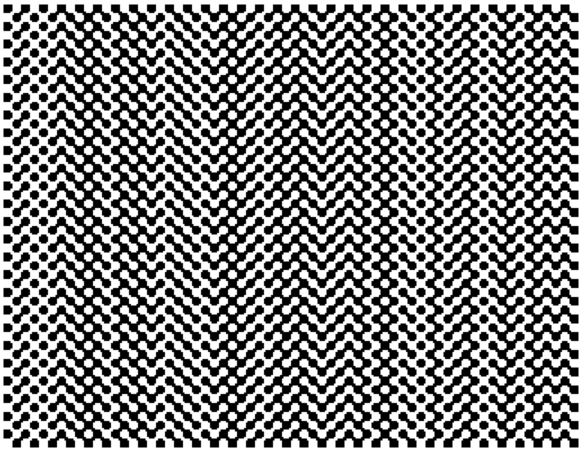


Figure 2: The binary coded pattern. Ray column indices are encoded locally in every window of minimal size 2×3 bits.

2.1 Ray identification problem

All optics based range finders utilize a variation of the same idea. Rays resp. planes are projected onto a scene and a triangulation is used to obtain 3-D coordinates of the corresponding dots resp. curves observed by a camera. Since the projector-camera rig is calibrated, 3-D coordinates are readily computed from the intersection of rays emanating from the camera (every pixel corresponds to a ray) with rays resp. planes emanating from the projector.

In particular, two kinds of controlled illumination have attracted most attention, namely single laser light plane for high precision and a fan of light planes for reasons of simplicity and fast acquisition. If a single plane is projected only one curve resulting from the intersection of the plane with a surface is measured for one image frame. To measure the whole surface a motion of the plane is needed. It is generally implemented either as a rotation of the scene or as a translation and/or rotation of the projector-camera rig. A large number of images has to be taken and processed to cover the visible part of a surface.

To shorten the acquisition time and to render complicated and vulnerable mechanics obsolete several rays resp. planes can be cast onto a scene simultaneously. Provided that the *ray or plane identification problem* can be solved, a large portion of a visible surface can be retrieved from a single image frame. The problem of ray identification comes from the fact that, unlike in the one plane setup, there is no longer a unique plane-ray intersection, see Fig. 1. A number of plane-ray intersection points, say $P_{-1}, P_0, P_1, \ldots$, project into the single point p in the image. To solve the problem at a point p the index j of the plane passing through the point P_0 has to be found. If such index is known for all points in the image, triangulation gives the same data set as if a sequence of planes was used. Hence, indices of planes have to be somehow "encoded" in the illumination to allow their extraction from the image. Vuylsteke and Oosterlinck have elaborated a method for local ray index encoding [Vuy87]. A pencil of rays arranged to lie in a fan of 63 vertical planes with a common line of intersection is casted onto a scene. Every ray has assigned the j (column) index of its plane. To encode column indices a special binary pattern has been developed. The basic structure of the pattern consists of a regular black and white chessboard pattern so that rays can be identified with corners of touching squares. The chessboard is modulated by an overlay pattern of small black and white squares that can be viewed as a code bit map if black is taken as 0 and white is taken as 1. By this, every ray carries one bit of additional information which together with the bits of its neighbours is used to identify its column index. A sample of 8 rows of the complete binary pattern is shown in Fig. 2.

The code bit map is designed in order to assure that every 2×3 window of neighboring bits, wherever found in the image, provides a unique column indices of its rays. Moreover, if some larger $2 \times n$ window of contiguous bits is recovered, the bits can be rearranged to form a word of a redundant binary code. In fact, every two consecutive rows of the bit map can be rearranged to form a pair of code words of 63 bit long cyclic code with characteristic polynomial $h(x) = 1 + x + x^6$. The minimal Hamming distance between every two different windows is growing as a function of n.

The redundancy is the key to the successful decoding of column indices in the presence of occlusion. Imagine the situation when the window is overlaps a depth discontinuity. In such a case, quite often, the bit map recovered comes from two pieces of distant bitmaps "glued" together. Then, the index recovered from a 2×3 window would certainly be wrong. If larger windows are used, the redundancy allows to find inconsistencies in code words coming from glued maps. Inconsistent code words can be marked as unreliable and their column index can be assigned in the post processing step using indices of their consistent neighbors.

2.2 Algorithm

In the previous section I explained how the projector pattern and its underlying bit map have been designed to provide an efficient solution to the ray identification problem. Now, I shall concisely describe how encoded images are processed in order to recover image coordinates, ray column indices, and 3-D coordinates of ray-surface intersections.

1. **Image coordinates detection**. In the first step of the algorithm the image of projected pattern is convolved with a 2×2 chessboard pattern. Local maxima and minima found by a nonmaxima suppression are subjected to a series of tests. Only significant and sufficiently symmetric extrema are marked as *feature points* and passed to the second step of the algorithm. Image coordinates (u, v) of feature points are found with subpixel accuracy.

2. **Mesh linking**. In order to form a local code word at a feature point its neighboring feature points have to be found. Hence, feature points are linked into a quadrilateral mesh. The known epipolar geometry of the calibrated rig and the four connectedness of the bit map are used to constrain the search space for north, west, east, and south neighbors of every feature point. Links are established to minimize the Euclidean distance between neighboring points in north, west, east, and south quadrant respectively.

3. **Column index decoding**. A region growing algorithm is applied to every node in the mesh. The region is grown until not enough bits are available to provide the information for decoding or an inconsistency is found. The number of bits, necessary for decoding, is a function of the Hamming distance required. The procedure is repeated starting with

a large Hamming distance and finishing with the smallest Hamming distance allowed. This process can be viewed as a greedy algorithm which never undoes what it has already done. So, we start with a large distance to find the most consistent indices first and less consistent later. A one step algorithm could result in either many unresolved indices for a large Hamming distance or many erroneous indices for a low distance.

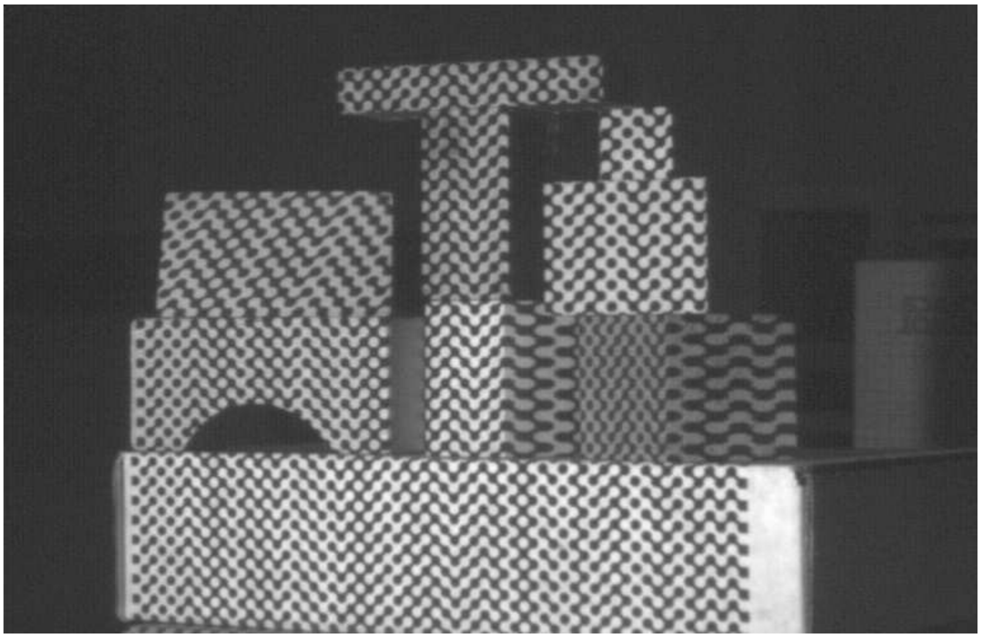
4. **Iterative index propagation**. Most rays get assigned their column indices in the column index decoding step but there can remain a number of rays, mainly near step discontinuities, which are unresolved. Constrained region growing is used to find their column indices. Indices are assigned only to those points which satisfy the ordering constraint on epipolar lines and which, at the same time, link their north and south neighbors with the same column index.

5. **Triangulation**. Once column indices are found all information needed, i.e. image coordinates (u, v) and a column index j to compute 3-D coordinates of ray-surface intersections is available. In the process of calibration a set of 63 sensor matrices is computed. All projector rays lying in one vertical plane share the same sensor matrix. Homogeneous 3-D coordinates of rays are computed by multiplication between their sensor matrices (uniquely determined by the column index) and the image coordinates written as a homogeneous vector.

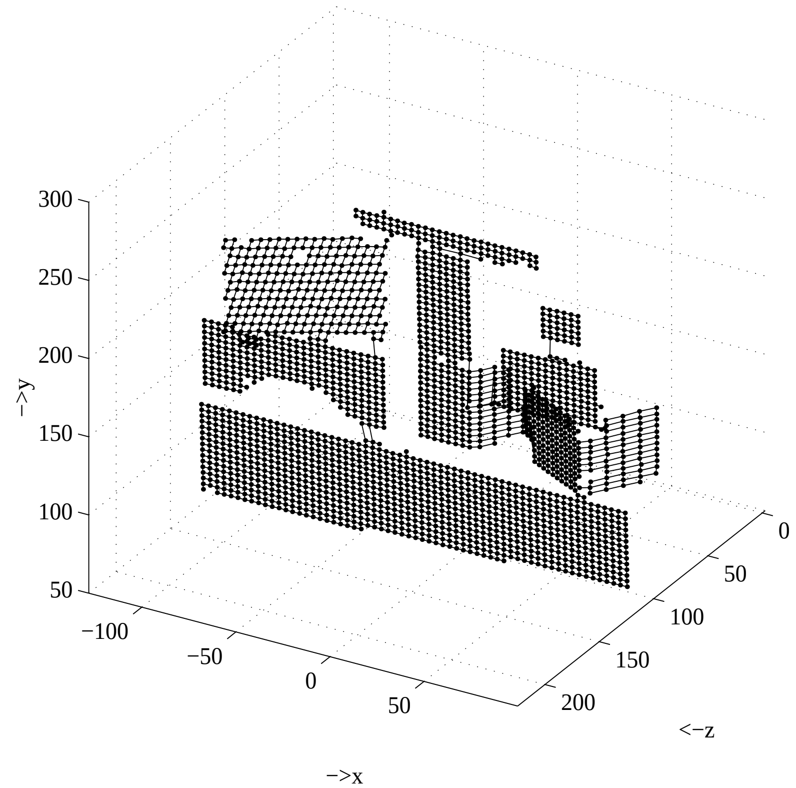
For a detailed description of the algorithm see [Vuy87]. The following section presents a few examples of the system's performance.

2.3 Examples

The system has been tested on a wide gamut of scenes. Fig. 3 shows a scene containing polyhedral wooden toy blocks with albedo almost constant over all surface patches. As can be seen the 3-D reconstruction is perfect except of a few missing points. Fig. 4 presents the reconstruction of a curved surface, namely a human hand. A similar example is shown in Fig. 5. Based on the described system pig slaughter quality evaluation setup has been developped [Van94]. The image was then processed off-line on our system. Finally, Fig. 6 shows the measurement of two Cola cans. The albedo of the surface is very nonuniform here. We can see that some points near contrast edges of "Cola" characters are disturbed and some points near specular reflections were lost. However, overall performance is still rather satisfactory and the resulting surfaces can be successfully matched as it was shown in [PVG95].

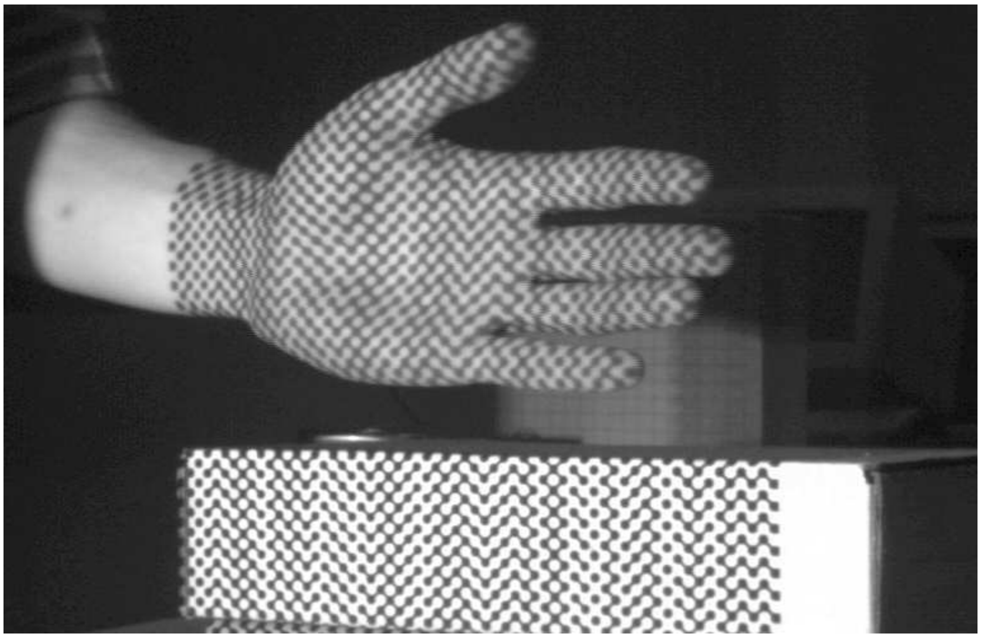


Illumination coded image.

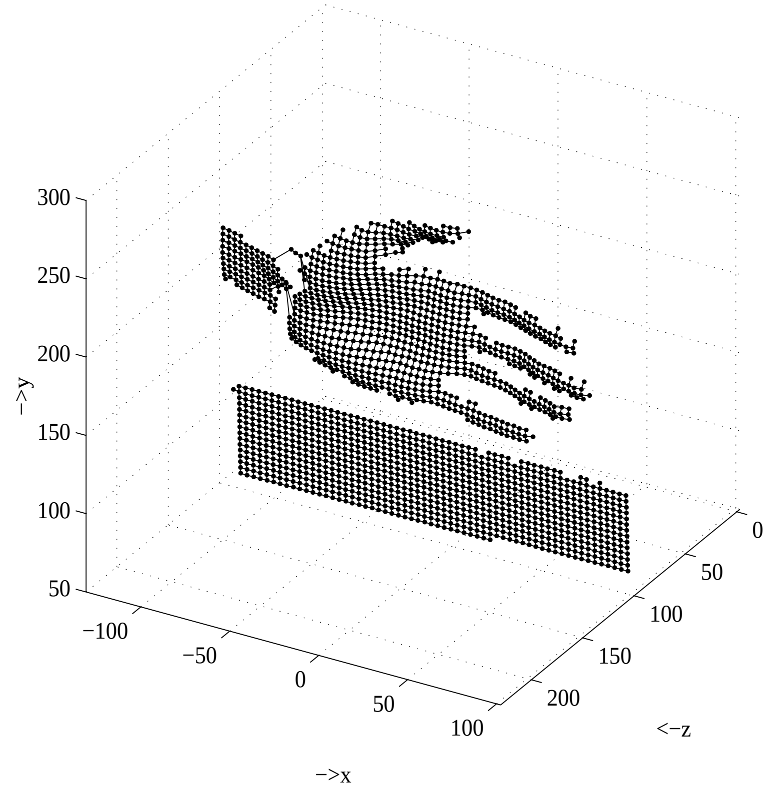


3-D reconstruction.

Figure 3: Scene with polyhedral objects.

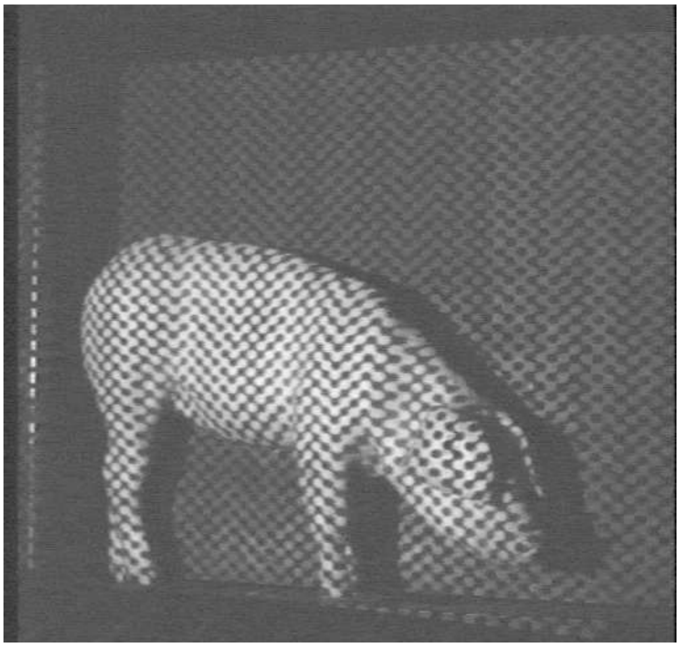


Illumination coded image.

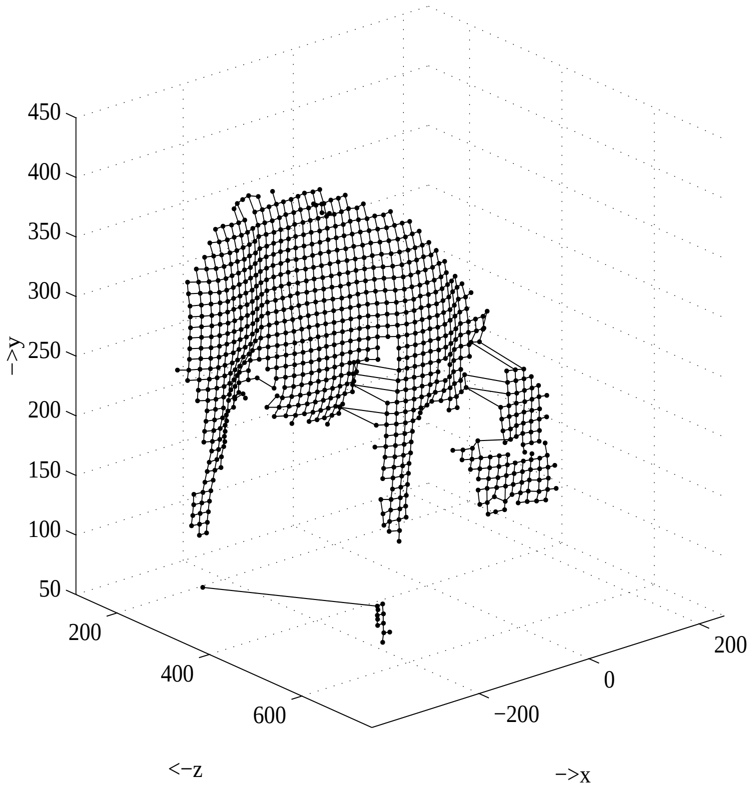


3-D reconstruction.

Figure 4: Human hand in motion.

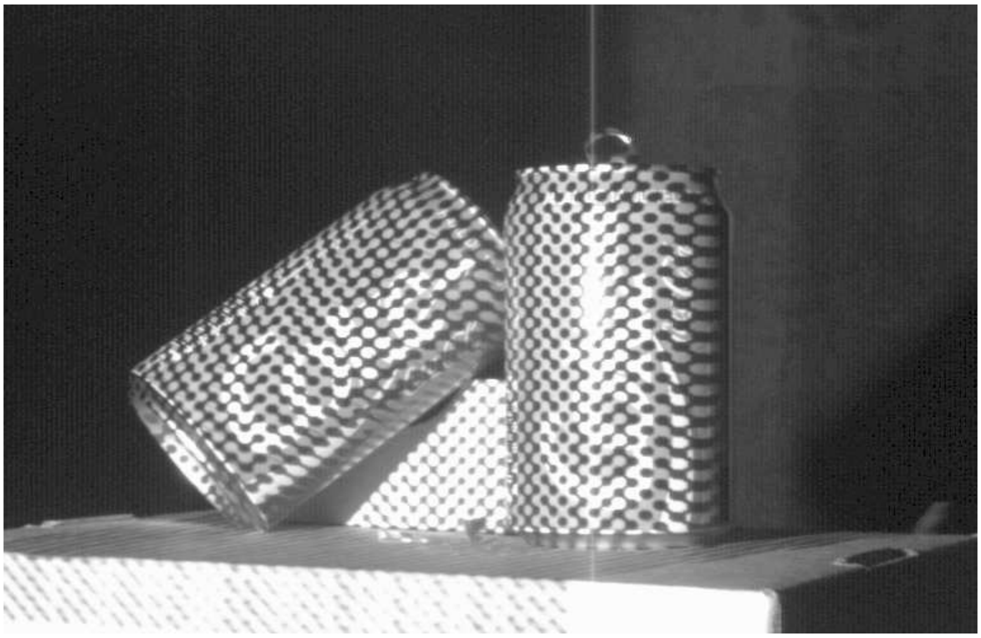


Illumination coded image.

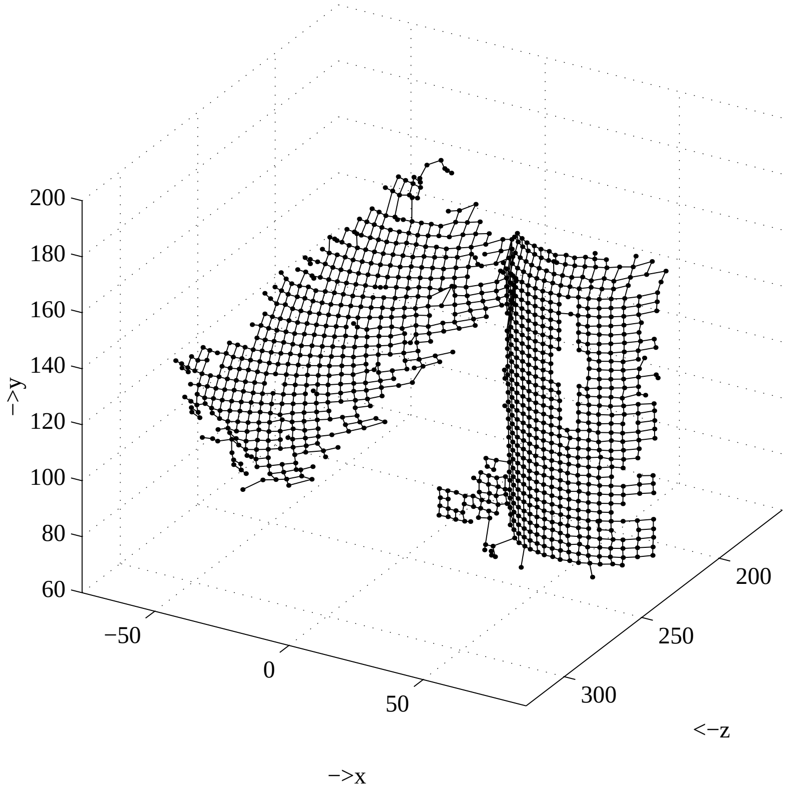


3-D reconstruction.

Figure 5: A pig measured in vivo.



Illumination coded image.



3-D reconstruction.

Figure 6: Two CocaCola cans - an example of a textured surface.

3 Previous implementations

The original system was coded by the author and as such was not used in further applications. Up to my knowledge, both real applications of the system ended up with recoding the original software and tailoring it to their needs.

The first application, coded by Marc De Groof, used the surface reconstruction in order to detect and evaluate scoliosis. The second one, coded by Immanuel Van Der Stuyft, utilizes 3-D measurement system to evaluate the slaughter quality of pigs in vivo [Van94].

3.1 Original implementation by Pieter Vuylsteke

The original system was coded in PASCAL programming language for VAX-750 minicomputer. The source code consists of 10,000 lines split into approximately 60 files comprising all necessary computation and visualisation routines. The implementation was closely tailored to experimental setup of that days. The image size was fixed to 512×512 pixels and this information was "randomly" scattered in the code. Only limited number of input parameters were interfaced to users and programmers. The source code was well commented but insufficiently structured. As proposed by Vuylsteke the whole algorithm has a "greedy" nature and consists of the stream of operations. I have found a few important parts of the source code missing. Particularly, there was no procedure computing the code bit sequence and forming lookup tables used in the process of ray index decoding.

3.2 Scoliosis detection by Marc De Groof

The scoliosis evaluation was implemented by Marc De Groof from MI2 group in PASCAL on an Apple Macintosh system. The original computational code was preserved while, at the same time, encapsulated into the Macintosh operating system using advanced window user interface. New routines for fitting splines to data and the scoliosis evaluation going behind the scope of the original work were added. The system was in operation in real conditions at Gasthuisberg hospital in Leuven. Now, some parts of the equipment, like the the projector and stages, are used in my experimental setup.

3.3 Pig slaughter quality evaluation by Immanuel Van Der Stuyft

Van Der Stuyft's system [Van94] is intended to work in real industrial environment. It is implemented on an IBM-PC compatible computer using Microsoft C language. Van Der Stuyft implemented its own environment for image processing involving memory management for conventional DOS memory map. The computational part of the range finder remained almost without changes but the system comprizes only the code for the reconstruction. Tha calibration is not involved and, when needed, it is done using my reimplementation at ESAT.

4 Reimplementation at ESAT

High level routines and all routines for visualisation are coded in Matlab[1] (as m-files). Computationally intensive or highly structured low level routines are coded in C. The whole stream of

[1] Matlab (c) Copyright 1984-93 The MathWorks, Inc. system originally designed for numerical computation and simulations in control engineering has become powerful tool for experimenting even with data of image size 512×512 elements. The main pro of Matlab is wide gammut of routines for linear algebra computation,

operations starting from loading an illumination coded image up to the code bit mesh recovery and decoding column indices is coded in C. It is interfaced to Matlab via the interface routine (FINDBITX.C) and it uses the Matlab matrix data structure. The visualisation (BCRFS.M), the triangulation (TRIANG3D.M) and the camera-world-projector calibration (BCRFC.M, BCRFCC.M, BCRFPC.M, CAMCAL.M, PROJCAL.M, etc.) are coded in Matlab language.

The system is split into the five main tasks (See Fig. 7 for data flow diagram.):

1. system calibration (BCRFC.M),
2. 3-D data measurement and surface reconstruction (BCRFM.M),
3. results visualisation (BCRFS.M),
4. demonstration (BCRFD.M),
5. the projector slide and the camera calibration pattern generation (CODEPTRN.M).

4.1 Calibration - BCRFC.M

Before any measurement can be done the setup has to be calibrated. The calibration task (BCRFC.M) captures calibration images and computes all necessary calibration matrices of the system. The output from the calibration is stored into the system calibration matrices SMATASC.`CalibrSessionId` and SPARASC.`CalibrSessionId`. The first one contains "sensor matrix". The second one keeps additional parameters defining the shape and size of linking search area. The variable `CalibrSessionId` identifies files belonging to the same calibration session.

In the process of calibration, the camera and the projector coordinate systems are related to a world coordinate system, Fig. 8. The world coordinate system is defined by a set of sample points, coordinates of which are a priory known (defined in the file PANELCAL.M). In order to define known points in a space, special calibration pattern is developed and "glued" on the calibration panel. The pattern contains 63 points, xy coordinate of which are known. The points are uniquely marked using the same binary code as is casted on scenes. Z coordinates of points are given by placing the panel into certain "panel slots". When doing calibration, the camera has to be calibrated first by taking images of the calibration pattern in at least two slots. Once the camera is calibrated, the projector calibration can be accomplished. The projector pattern has to be casted on a white board (use back side of the camera calibration panel). Again, at least two positions of the panel are required. Images can either be grabbed during the process of calibration or they can be loaded from a BCRF image directory (defined by `BcrfImagDir` in BCRFINST.M). In the latest case, camera calibration images must have names CCAL`CamCalSlotNumbers.ImgExt`, and projector images must have names PCAL`CamCalSlotNumbers.ImgExt`. The calibration task, `bcrfc`, allows to perform camera and projector calibtration separately by setting appropriate parameters in the conrol parametr file BCRFCCTR.M. The following command runs the calibration using the standard parameter file BCRFCCTR.M:

optimization, data examination, and visualisation available. The main con is its relative low efficiency compared to C concerning data representation (there is one unique data structure - 2-D matrix containing double precision elements). Matlab programming language is a procedural, C like, and interpretted. Every procedure has to be in its separate file. Fortunately, it is possible to create Matlab callable C or Fortran modules (.mex, .mexhp7, etc.) [Mat92] and thus overcome above mentioned disadvantages.

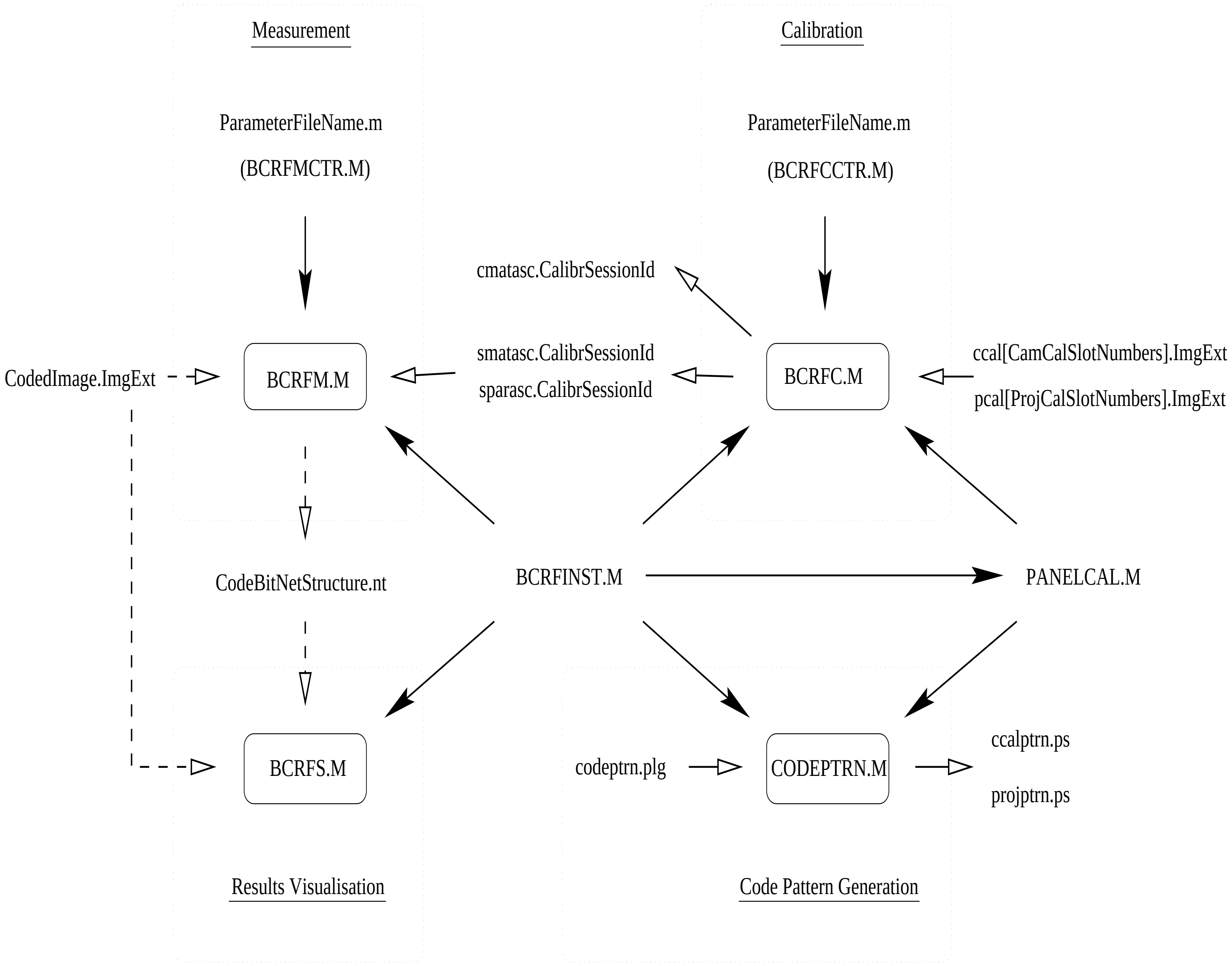


Figure 7: Process and data flow diagram. Solid links with filled arrow represent m-file calling, dashed links mean file or workspace variable loading, and solid links with empty arrow represent data loading or saving of data files.

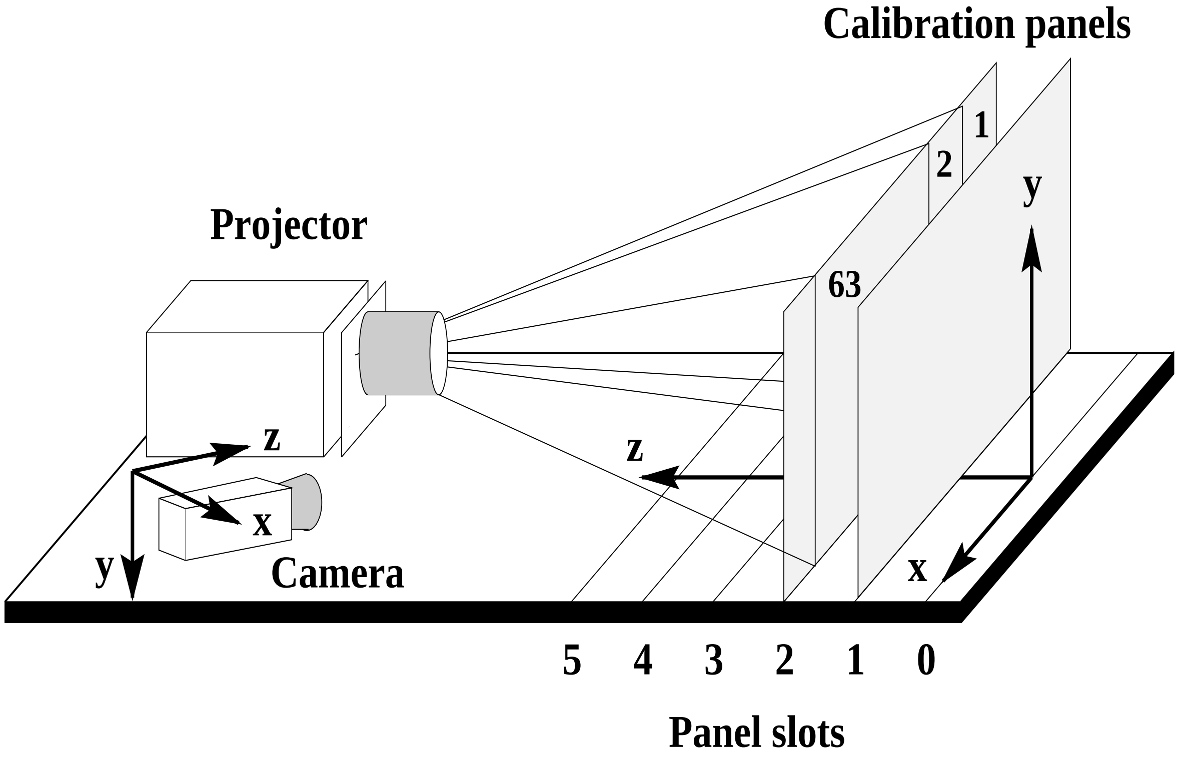


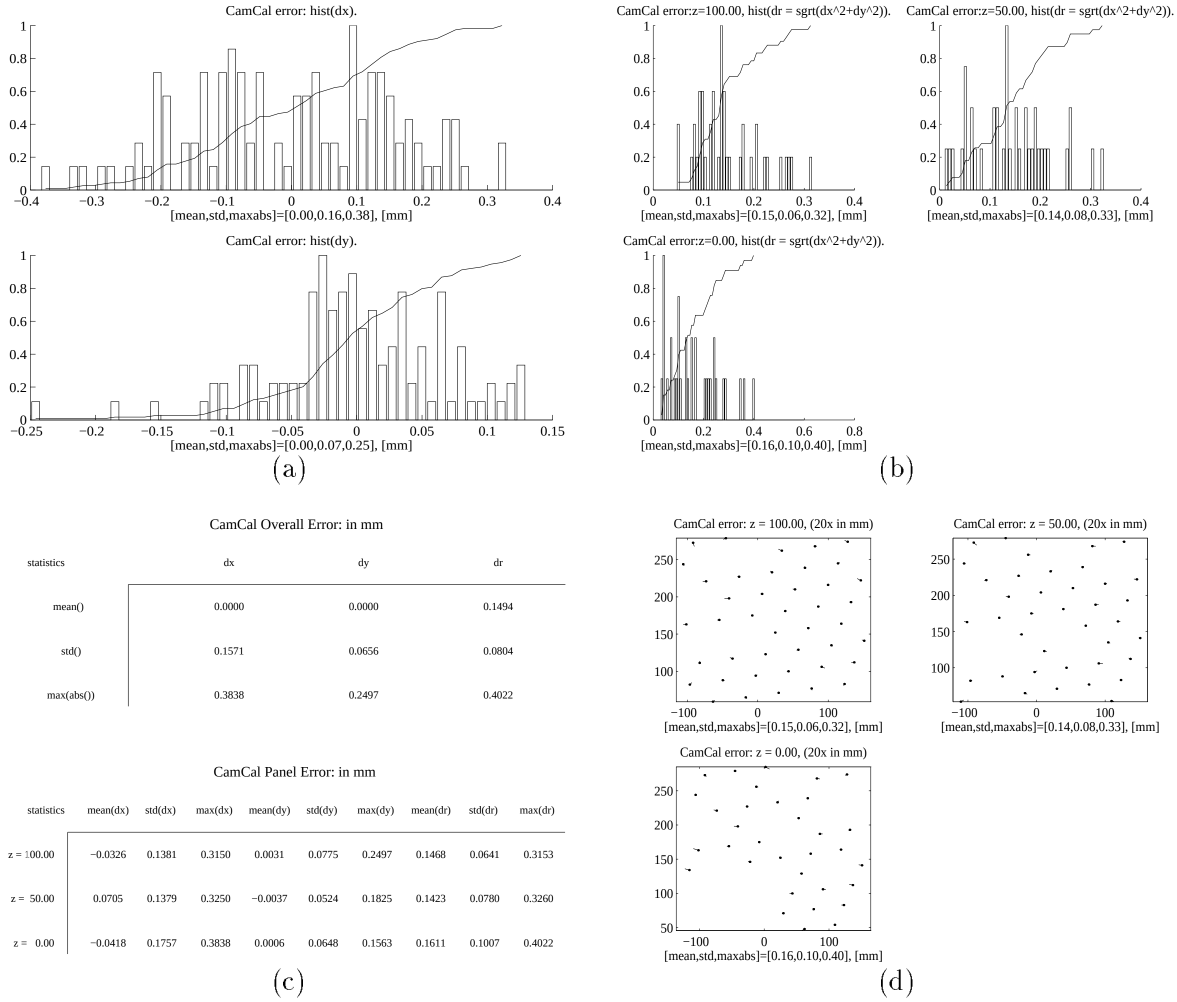
Figure 8: The experimental setup. World and camera coordinate systems are shown as well as the positions of calibration panels. The rays emanating from the projector has to have their column indices increasing from the left to the right when viewed from the side of the camera. Be carefull to have the slide in the projector properly oriented.

```
>>bcrfc;
```

The results of the calibration are shown in various ways allowing to check their validity. Results of the camera calibration are presented in Figure 9. Figure 9(a) shows the histogram of xy coordinate errors on all camera calibration panels. The erros are computed as differences between known calibration points and points reconstructed from the backprojection using the knowledge of panel z positions. Figure 9(b) shows similar histograms separately for each calibration panel. Here, Euclidean distances are show instead of coordinate differences. Figure 9(c) gives the results of different statistics run on error samples. The top table gives quantitative values for the histogram on Figure 9(a). The bottom table shows the same values for Figure 9(b). Labels "mean()", "std()", and "max(abs())" stand for mean value, standard deviation, and maximum absolute value, respectively. Finally, Figure 9(d) shows the errors projected on the calibration panels as oriented vectors. Calibration points are marked as bold dots. Line segments emanating from dots show directions of error vetors. The size of lines is 20 times the error in milimetres [2].

Results of the projector calibration are shown on Figure 10. Figure 10(a) shows the histogram of z coordinate errors for all reconstructed projector points. Here again, known positions of calibration panels and the calibrated camera allow to reconstruct and compare

[2] Units depend on units in which calibration points are measured.



CamCal Overall Error: in mm

statistics	dx	dy	dr
mean()	0.0000	0.0000	0.1494
std()	0.1571	0.0656	0.0804
max(abs())	0.3838	0.2497	0.4022

CamCal Panel Error: in mm

statistics	mean(dx)	std(dx)	max(dx)	mean(dy)	std(dy)	max(dy)	mean(dr)	std(dr)	max(dr)
z = 100.00	−0.0326	0.1381	0.3150	0.0031	0.0775	0.2497	0.1468	0.0641	0.3153
z = 50.00	0.0705	0.1379	0.3250	−0.0037	0.0524	0.1825	0.1423	0.0780	0.3260
z = 0.00	−0.0418	0.1757	0.3838	0.0006	0.0648	0.1563	0.1611	0.1007	0.4022

(c)

Figure 9: Camera calibration results.

z coordinates of all points casted from the projector onto the panels. Figure 10(b) gives the same histograms seprately for all calibration panels. Figure 10(c) presents various statistics computed for z coordinate errors. Finally, Figure 10(d) shows results of plane fitting through rays with common column indices using a robust estimator, Least Winsorized Squares. The top sub-figure shows the plot of angles between planes and world coordinate z axis. The second sub-figure shows intesectios of the planes with the world xy plane. The third sub-figure gives the same for the world xz plane. The fourth sub-figure shows the number of inliers used to compute Least Square plane fit. Finally, the last, fifth, sub-figure shows the number of outliers detected in the process of fitting. All graphs are plotted against the column numbers of planes[3].

4.2 Measurment - BCRFM.M

The main task (BCRFM.M) performs direct range data measurment on a calibrated setup. The parameter file `ParameterFileName`.M (implicitly BCRFMCTR.M) allows to control the

[3] Each plane has assigned the column number of their rays.

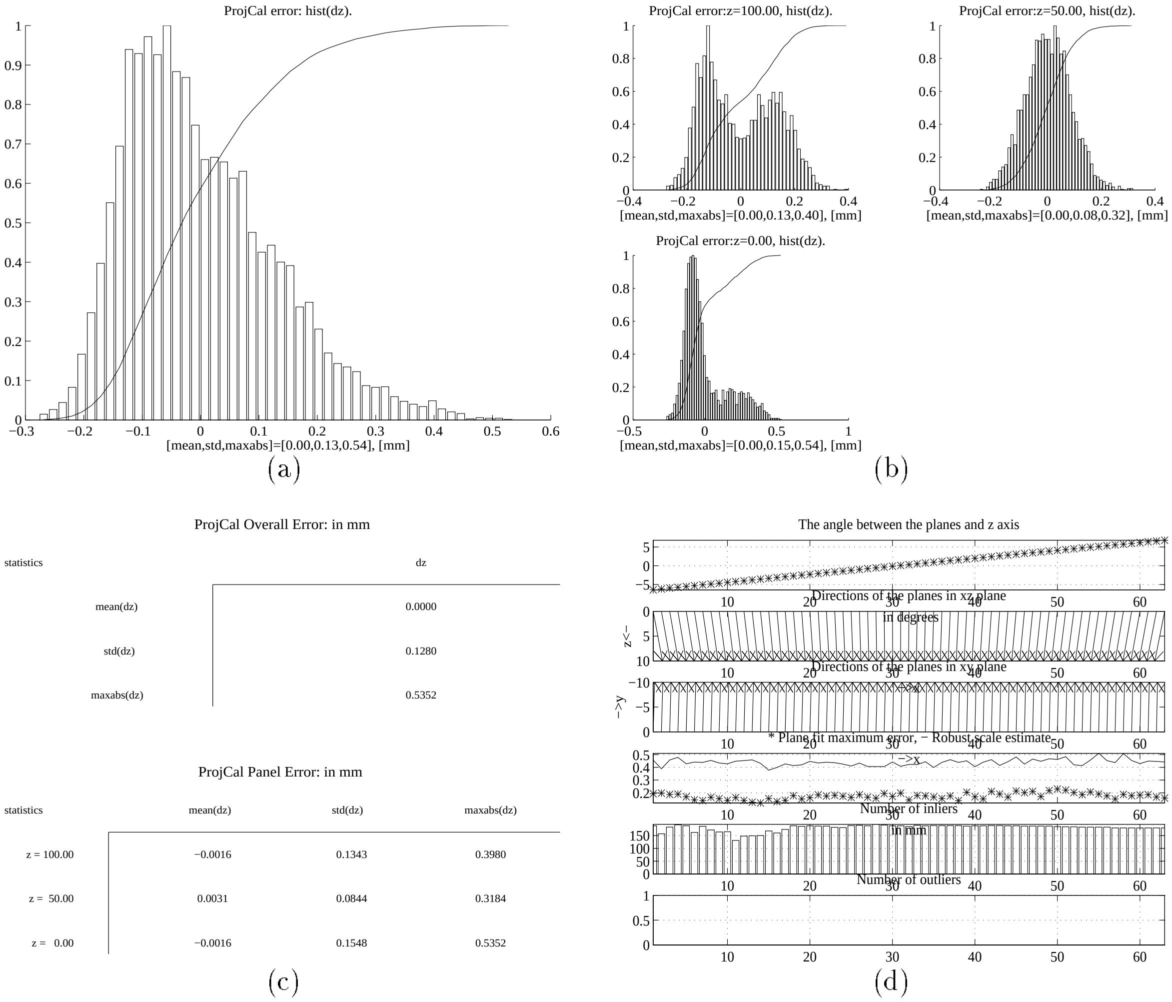


Figure 10: Projector calibration results.

functionality of the system. It can, and mostly does, change from scene to scene. The input image `CodedImage.ImgExt` can be either loaded from a disk as well as from the Matlab workspace if it is already there. The following Matlab command:

```
>>[nT,inputImage] = bcrfm('inputFileName','parameterFileName');
```

loads and process the image file `inputFileName.ImgExt` using the parameter script `parameterFileName`.M. The image extension – `ImgExt` – is automatically appended by the procedure `bcrf2m` according to the variable `BcrfLocalization` which is defined in the file BCRFINST.M. New methods for loading images into the Matlab workspace can be added to `BCRF2M.M` if new data formats are to be handled. All images are automatically loaded from "image" directory defined by the variable `BcrfImagDir` in the file BCRFINST.M.

Besides an input image, two other files are required in order to run the measurement. The first one, CMATASC.`CalibrSessionId`, contains the sensor calibration matrix. The second one, SPARASC.`CalibrSessionId`, contains the search area parameters for code bit mesh linking. Both files are created in the calibration process by the task BCRFC.M. They are identified by the calibration identification number `CalibrSessionId` appearing as file extensions. For particular measurment, the number is defined by the variable, `CalibrSessionId` in the parameter file `parameterFileName`.M. The same variable `CalibrSessionId`, appeares in the calibration parameter file defining current calibration session. If a particular calibration is to be used, both identification numbers must be equal.

The measurment task returns "Code-Bit-Topological-Network-Structure" (CBTN) as a workspace variable and, if requested by setting `SaveResults = 1;` in the parameter file, data are saved as a Malab data file into the directory `BcrfDataDir` (defined in the file BCRFINST.M). The second parameter returned by `bcrfm`, `inputImage`, is a matrix containing the input image of the measured scene.

The CBTN structure is encapsulated into a Matlab matrix. All valid points in 3-D can be obtained by using the procedure `tnodes`. The following sequence of commands retrieves the coordinates of 3-D points and plots them as stars in a 3-D space:

```
>>tT = tnodes(nT);
>>X  = tT(:,NODE_X:NODE_Z);
>>show3dp(X,'*','all');
```

Each row in the matrix `tT` holds one point and carries altogether 19 attributes:

```
              1           2        3             4              5
tT(i,:) = [node index, u coord., v coord., feature intensity, bit value,

              6           7        8        9                 10
          bit signum, bit pair, iteration,  node status,  column index,

             11          12                13         14       15
          x coordinate, y coordinate, z coordinate, NORD nbr., WEST nbr.,

             16          17            18           19
          SOUTH nbr., EAST nbr., previous node, next node]
```

The attributes can be accessed through the global variables defined in the BCRFGVAR.M

4.3 Visualization of results - BCRFS.M

The results of measurement, i.e. the input image and a reconstructed surface, can be visualized using the task `bcrfs` (BCRFS.M). The following Matlab commands measure a scene and show the results:

```
>>[nT,img] = bcrfm('inputFileName','controlfileName');
>>bcrfs(nT,img,'all');
```

4.4 Demonstration - BCRFD.M

The demonstration program (BCRFD.M) provides the basic demonstration of system's performance on several scenes. The demonstration is invoked by the command:

```
>>bcrfd;
```

It shows three scenes, Fig. 3, Fig. 4, and Fig. 5 and also includes four figures of intermediate processing steps. The demonstration task can be also used if the processing and visualisation are to be combined into one routine. The following command:

```
>> [nT,img]=bcrfd('inputImageName','parameterFileName');
```

performs a measurement and visualisation. The reconstruction is returned as a workspace variable.

4.5 Code patterns generation - CODEPTRN.M

There are two special patterns used in the system. The projector pattern is printed on a transparent slide and gives rise to coded illumination. The camera calibration pattern is printed on a white paper and allows to define calibration points in a 3-D space and their authomatic identification in images.

Both patterns can be generated by the task `codeptrn` (CODEPTRN.M). The following command:

```
>>codeptrn(64,3,30/64,0.17,3,3);
```

generates the projector pattern with 64 rows (it has always 63 columns) and chesboard square size equal to $\frac{30}{64}$ mm. The size of bit squares is 0.17 times the size of chesboard squares. The camera calibration pattern has chesboard square size equal to 3 mm and the size of bit squares is 0.17 times the size of chesboard squares. Each calibration point is marked by a code word giving rise to a $2 \times n$ code chessboard pattern (n must be odd). In this example $n = 3$. Calibration points are associated with the middle upper bit point in each pattern. Placement of calibration points is controlled by camera calibration coordinates supplied by the procedure `panelcal` (PANELCAL.M). Be careful to set correctly localization of the system. It afffects coordinates used in the pattern generation as well as coordinates used in the calibration process.

Both patterns are generated in PostScript format and saved into the directory `BcrfDataDir`. The projector pattern is saved into the file PROJPTRN.PS and the camera calibraton pattern is saved into the CCALPTRN.PS. It is important to realize that the projector pattern has to be correctly placed into the projector. If printed on a white paper, the left upper corner when projected onto calibration panels has to appear as the left upper corner of the pattern's projection.

4.6 Parameter files

All parameters of the system are interfaced to the user. There are four parameter files (Matlab callable scripts) containing all parameters that could be eventually changed. The files contain definitions of the parameter variables used in particular tasks. The following parameter files must be provided to the system:

4.6.1 BCRFINST.M

The file BCRFINST.M contains the information about system localization, directory map, and stable parameters which are not supposed to change from scene to scene. This file is called by all tasks in the system.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                           %
%       Binary Coded Range Finder     -   BCRFINST.M                        %
%                                                                           %
%       Instalation & localization.                                         %
%                                                                           %
%       Author:         T. Pajdla 1994, 1995                                %
%                                                                           %
%       Language:       Matlab 4.1, (c) MathWorks                           %
%                                                                           %
%       03/06/95 ESAT-MI2, KU Leuven                                        %
%                                                                           %
% VariableName           = value;       % Comment                           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

BcrfLocalization         = 'LEUVEN';    % {'PRAGUE','LEUVEN',
                                        %   'IWONL3','FRUITS'}

%%%%%%%%%%%%%%% Code Table and Decoding Table Definitions %%%%%%%%%%%%%%%%%%%

CodeSubWordSize          = [4];         % code subword size in bytes

%%%%%%%%%%%%%%%%%%% Set path to all directories needed  %%%%%%%%%%%%%%%%%%%%%

BcrfHomeDir = [pmpath('home') pathsep '3DScan.new' pathsep 'BCRF' pathsep];
BcrfCtrlDir = [BcrfHomeDir 'BCRFCTRL' pathsep];
BcrfDataDir = [BcrfHomeDir 'DATA' pathsep];
BcrfImagDir = [pmpath('home') pathsep 'Images' pathsep];

%%%%%%%%%%%%%%%%%%% Set default text font and its size %%%%%%%%%%%%%%%%%%%%%

set(0,'DefaultAxesFontName','times')
set(0,'DefaultTextFontName','times')
set(0,'DefaultAxesFontSize',12)
set(0,'DefaultTextFontSize',9)
```

4.6.2 BCRFMCTR.M

The measurement process requires to input the name of a parameter file `ParameterFileName`.M (implicitly BCRFMCTR.M) containing a number of parameters influencing the computation as well as the visualization of results.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                           %
%       Binary Coded Range Finder     -   BCRFMCTR.M                        %
%                                                                           %
%       Measurement parameter file.                                         %
%                                                                           %
%       Author:         T. Pajdla 1994, 1995                                %
%                                                                           %
%       Language:       Matlab 4.1, (c) MathWorks                           %
%                                                                           %
```

```
%       03/06/95 ESAT-MI2, KU Leuven                                   %
%                                                                      %
% VariableName          = value;        % Comment                      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

BcrfImageSource          =  [ 0];        % image source: 0 ... from a file
                                         %               1 ... from a camera
ImageWindow              =  [-1];        % {-1,<1 rNum 1 cNum>}
                                         %  -1 = full image

%%%%%%%%%%%%%%%%%%%%%%%%%%%% Calibration %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

CalibrSessionId          =  [2];         % number of the calibration session

%%%%%%%%%%%%%%%%% Feature Point Intensity Computation %%%%%%%%%%%%%%%%%%%%

FeatIntMaskSize          =  [7];         % feature point detection mask size
                                         % values = [7 11 15 19 23]

FeatSelFact              =  [2];         % feature selection factor
                                         % recommended range = [0.5:2.0]

FeatSelThr               =  [5];         % feature selection threshold
                                         % recommended range = [0:10]

FeatIntZeroThr           =  [0];         % obsolete

FeatSelMaxMaxDist        =  [1];         % max-max feature selection dist.
                                         % recommended values = [1]

%%%%%%%%%%%%%%%%%%%%%%%%% Code Bit Localization %%%%%%%%%%%%%%%%%%%%%%%%%%%%

MinMaxDist               =  (FeatIntMaskSize-1)/2;
                                         % nonmaxima suppression,
                                         % minimum distance between two
                                         % maxima allowed (must be odd!)

SubPixSelectFact         =  [0.8];       % sub-pixel code bit localization
                                         % selection factor
                                         % range = [eps:1], eps > 0

DoSaddleTest             =  [1];         % 1 ... do saddle test

DoGrayVarTest            =  [0];         % obsolete

GrayVarThr               =  [0];         % obsolete

%%%%%%%%%%%%%%%%%%%%%%%%% Column Index Decoding %%%%%%%%%%%%%%%%%%%%%%%%%%%%

MaxSearchDepth           =  [ 3];        % maximal search depth used for
                                         % decoding, see C procedure
                                         % DecodingViaElimination
                                         % 2*(2*MaxSearchDepth+1)>MinBitNum
```

```
MinBitNum                =  [14];        % this can be a vector holding
                                         % decreasing values:
                                         % ~ [MinBitNumMax:MinBitNumMin]
                                         % from the set   =[6 8 10 12 14 16]
                                         % MinHammingDist =[1 2  3  4  4  5]

MaxIterNum               =  [30];        % maximum nuber of iterations for
                                         % for region growing in Iterative
                                         % index propagation

%%%%%%%%%%%%%%%%%%%%%%%%%%% Process Control %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

ShowProcess              =  [1];         % print elapsed time

SaveInterResults         =  [0];         % save intemediate results for
                                         % troubleshooting

ClearInterResults        =  [0];         % obsolete

SaveResults              =  [0];         % save results into a file

%%%%%%%%%%%%%%%%%%%%%%% Results Visualisation %%%%%%%%%%%%%%%%%%%%%%%%%%%%%

ShowBcgrImage            =  [0];         % show underlying images

ShowCodeBitDetection     =  [0];         % show intermediate results of
                                         % code bit detection
 ShowMeshPlots           =  [0];         % feature intensity (slow mesh)
 ShowContourPlots        =  [0];         % feature intensity (slow cintour)
 ShowHistograms          =  [0];         % histograms of feature intensity
 ShowBitSignum           =  [0];         % show bit signums
 ShowBitValue            =  [0];         % show bit values
 ShowCodeNetworkLinking  =  [0];         % show Code Bit Topological Net

ShowDecodedColumnIndex   =  [0];         % show decoded column index
 ShowBitPair             =  [0];         % mark bit pairs

ShowTriangResults        =  [1];         % show reconstructed points
Show3DPlotCube           =  'all';       % only points inside the cube
                                         % [minX maxX minY maxY minZ maxZ]
                                         % are plotted.
                                         % 'all' = show all points

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

4.6.3 BCRFCCTR.M

The calibration process requires to input the name of a parameter file `ParameterFileName`.M (implicitly BCRFCCTR.M).

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                         %
```

```
%       Binary Coded Range Finder    -   BCRFCCTR.M                          %
%                                                                            %
%       Calibration parameter file.                                          %
%                                                                            %
%       Author:         T. Pajdla 1994, 1995                                 %
%                                                                            %
%       Language:       Matlab 4.1, (c) MathWorks                            %
%                                                                            %
%       03/06/95 ESAT-MI2, KU Leuven                                         %
%                                                                            %
% VariableName          = value;       % Comment                             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

BcrfImageSource         =  [ 0];       % image source: 0 ... from a file
                                       %               1 ... from a camera
ImageWindow             =  [-1];       % {-1,<1 rNum 1 cNum>}
                                       %  -1 = full image

%%%%%%%%%%%%%%%%%%%%%%%%%% Calibration %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

CalibrSessionId         =  [13];       % number of the calibration session

%%%%%%%%%%%%%%%%%%%%%%%%%%% Camera %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DoCameraCalibration     =  [1];        % 0 ... load camera matrix
                                       % 1 ... calibrate camera

CamCalSlotNumbers       =  [1 2 3];    % panel slot indices

CamCalFeatIntMaskSize   =  [11];       % feature point detection mask size
                                       % values = [7 11 15 19 23]

CamCalFeatSelFact       =  [1];        % feature selection factor
                                       % recommended range = [0.5:2.0]

CamCalFeatSelThr        =  [2];        % feature selection threshold
                                       % recommended range = [0:10]

CamCalMinMaxDist        =  (CamCalFeatIntMaskSize-1)/2;
                                       % nonmaxima suppression,
                                       % minimum distance between two
                                       % maxima allowed (must be odd!)

CamCalMinBitNum         =  [14];       % this can be a vector holding
                                       % decreasing values:
                                       % ~ [MinBitNumMax:MinBitNumMin]
                                       % from the set   =[6 8 10 12 14 16]
                                       % MinHammingDist =[1 2  3  4  4  5]

CamCalMaxIterNum        =  [0];        % maximum nuber of iterations for
                                       % for region growing in Iterative
                                       % index propagation
                                       % MUST BE 0
```

```
CamCalNbrNum             =  [3];         % valid calibration point must have
                                         % at least 2*CamCalNbrNum nighbours
                                         % on both W-E sides

%%%%%%%%%%%%%%%%%%%%%%%%%% Projector %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DoProjCalibration        =  [1];         % 1 ... do projector calibration,
                                         % and compute the sensor matrix

ProjCalSlotNumbers       =  [1 2 3];     % panel slot indices

ProjCalFeatIntMaskSize   =  [11];        % feature point detection mask size
                                         % values = [7 11 15 19 23]

ProjCalFeatSelFact       =  [2];         % feature selection factor
                                         % recommended range = [0.5:2.0]

ProjCalFeatSelThr        =  [5];         % feature selection threshold
                                         % recommended range = [0:10]

ProjCalMinMaxDist        =  (ProjCalFeatIntMaskSize-1)/2;
                                         % nonmaxima suppression,
                                         % minimum distance between two
                                         % maxima allowed (must be odd!)

ProjCalMinBitNum         =  [14];        % this can be a vector holding
                                         % decreasing values:
                                         % ~ [MinBitNumMax:MinBitNumMin]
                                         % from the set   =[6 8 10 12 14 16]
                                         % MinHammingDist =[1 2  3  4  4  5]

ProjCalMaxIterNum        =  [0];         % maximum nuber of iterations for
                                         % for region growing in Iterative
                                         % index propagation

ProjCalCutOffNum         =  [2 50 2 2 85];
                         %[fitMethod #Samples absCutOff RelCutOff Quantile]
                                         %
                                         % fitMethod = plane fit method
                                         %  0 .. least squares
                                         %  1 .. last quantile of squares
                                         %  2 .. winsorized squares
                                         %
                                         % #Samples  = max. num. of samples
                                         %  for last quantile of squares
                                         %
                                         % absCutOff = absolute outlier thr.
                                         %  err > absCutOff => outlier
                                         %
                                         % relCutOff = absolute outlier thr.
                                         %  err > relCutOff*rs => outlier,
                                         %  rs = robust estimate of scale
```

```
                                        %
                                        % Quantile = percentage of samples
                                        %  contributing to error
                                        % ( 50 => least median of squares)

%%%%%%%%%%%%%%%%% Feature Point Intensity Computation %%%%%%%%%%%%%%%%%%%%%

FeatIntZeroThr          =  [0];         % obsolete
FeatSelMaxMaxDist       =  [1];         % max-max feature selection dist.
                                        % recommended values = [1]

%%%%%%%%%%%%%%%%%%%%%%%%% Code Bit Localization %%%%%%%%%%%%%%%%%%%%%%%%%%%

SubPixSelectFact        =  [0.8];       % sub-pixel code bit localization
                                        % selection factor
                                        % range = [eps:1], eps > 0

DoSaddleTest            =  [1];         % 1 ... do saddle test

DoGrayVarTest           =  [0];         % obsolete

GrayVarThr              =  [0];         % obsolete

%%%%%%%%%%%%%%%%%%%%% Code Bit Topology Net Linking %%%%%%%%%%%%%%%%%%%%%%%%

EpipolAng               =  [0.0];       % epipolar angle [0:pi/4]

Nx1                     =  [-20];       % nothern link search area
Nx2                     = - CamCalMinMaxDist; %
Ny1                     =  [-12];       %
Ny2                     =  [ 12];       %

Wx1                     =  [-4];        % western link search area
Wx2                     =  [ 4];        %
Wy1                     =  [-25];       %
Wy2                     = - CamCalMinMaxDist; %

%%%%%%%%%%%%%%%%%%%%%%% Column Index Decoding %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

MaxSearchDepth          =  [ 3];        % maximal search depth used for
                                        % decoding, see C procedure
                                        % DecodingViaElimination
                                        % 2*(2*MaxSearchDepth+1)>MinBitNum(1)

%%%%%%%%%%%%%%%%%%%%%%%%%%%% Process Control %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

ShowProcess             =  [1];         % print elapsed time

SaveInterResults        =  [0];         % save intemediate results for
                                        % troubleshooting

ClearInterResults       =  [0];         % obsolete
```

```
SaveResults              =  [0];        % save results into a file

%%%%%%%%%%%%%%%%%%%%%%%%%%%% Results Visualisation %%%%%%%%%%%%%%%%%%%%%%%%%

ShowCalProcess           =  [1];        % print elapsed time

DoShowCCalColIdx         =  [0];        % column indices from camera images

DoShowPCalColIdx         =  [0];        % col. ind. from projector images

ShowCamCalResults        =  [1];        % results of the camera calibration

ShowProjCalResults       =  [1];        % results of the proj. calibration

ShowSensCalResults       =  [0];        % obsolete

ShowBcgrImage            =  [0];        % show underlying images

ShowCodeBitDetection     =  [0];        % show intermediate results of
                                        % code bit detection
 ShowMeshPlots           =  [0];        % feature intensity (slow mesh)
 ShowContourPlots        =  [0];        % feature intensity (slow cintour)
 ShowHistograms          =  [0];        % histograms of feature intensity
 ShowBitSignum           =  [0];        % show bit signums
 ShowBitValue            =  [0];        % show bit values
 ShowCodeNetworkLinking  =  [0];        % show Code Bit Topological Net

ShowDecodedColumnIndex   =  [0];        % show decoded column index
 ShowBitPair             =  [0];        % mark bit pairs

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

4.6.4 PANELCAL.M

The file PANELCAL.M contains world coordinates of camera calibration points and z coordinates of projector calibration points. It strongly depends on actual setup of the experiment and has to be changed if calibration patterns are changed. The file is called by the calibration and supports calibration pattern generation process.

4.7 Procedures

There are basically two main classes of procedures. Matlab callable procedures, m-files, and C procedures. Below the list of all m-files is given. C procedures can be tracked rooting from the file FINDBITX.C and CALPOINT.C.

```
BCRF2M   BCRF-v1.0: Loads images into Matlab
BCRFC    BCRF-v1.0: Calibration
BCRFCC   BCRF-v1.0: Camera calibration - main
BCRFD    BCRF-v1.0: Demonstration
BCRFGVAR BCRF-v1.0: Global variables definitions
BCRFM    BCRF-v1.0: 3-D Surface reconstruction using active vision
BCRFPC   BCRF-v1.0: Projector calibration
BCRFPRD  BCRF-v1.0: Saves demonstration images into PostScript files
```

```
BCRFS    BCRF-v1.0: Plots results.
CALPOINT BCRF-v1.0: Selects calibration points (mex)
CAMBPROJ BCRF-v1.0: Reconstructs 3-D coordinates of camera calibration points
CAMCAL   BCRF-v1.0: Camera calibration - linear formulation
CODEPTRN BCRF-v1.0: Generates the projector pattern and the camera calibration pattern
CWPLUS   BCRF-v1.0: Defines positive code word
DECODTAB BCRF-v1.0: Defines decoding lookup table
LINK3    BCRF-v1.0: Plots lines joning points in 3-D
FEATZLEV BCRF-v1.0: Local zero level of feature intensity
FINDBIT3 BCRF-v1.0: Finds code bits and decodes column indices (calls FINDBITX)
FINDBITX BCRF-v1.0: Finds code bits and decodes column indices (mex)
NODES    BCRF-v1.0: Removes the header from CBTN matrix
PANELCAL BCRF-v1.0: Defines calibration points and panels
PROJCAL  BCRF-v1.0: Projector matrix calibration
SENSMAT  BCRF-v1.0: Constructs sensor matrix
SHOW3DP  BCRF-v1.0: Plots 3-D points
SHOW3LNK BCRF-v1.0: Plots lines between CBTN neighbouring nodes in 3-D
SHOWBLNK BCRF-v1.0: Plots lines between neighbouring nodes CBTN into an image
SHOWBPR  BCRF-v1.0: Marks bit pairs in an image
SHOWCCAL BCRF-v1.0: Plots results of the camera calibration
SHOWDCOL BCRF-v1.0: Plots decoded column indices
SHOWFBIT BCRF-v1.0: Plots results of feature detection and code bit localization
SHOWPCAL BCRF-v1.0: Plots results of the projector calibration
SHOWTRNG BCRF-v1.0: Plots reconstructed points in 3-D
TNODES   BCRF-v1.0: Returns nodes of CBTN with valid column indices
TRIANG3D BCRF-v1.0: Reconstructs 3-D points using the triangulation
UV2XYZ   BCRF-v1.0: 3-D triangulation (low level function called by TRIANG3D)
VNODES   BCRF-v1.0: Return non-empty nodes in CBTN matrix


BIN2DEC  Pajdla: Converts binary matrix columns into decadic number vectors
BMP2M    Smutny: Loads color images in BMP format (uncompressed only)
CLIP3D   Pajdla: Clips 3-D points outside the clipping cube
COLORMEN Smutny: Adds a color map menu to the current figure
CUMHIST  Pajdla: Plots a histogram and a cummulative histogram
DEC2BIN  Pajdla: Converts decadic numbers into binary vectors
DIGIPLOT Pajdla: Plots binary digital signal stored in matrix columns
IMGLINK  Pajdla: Plots lines between points in the image
IMGPLOT  Pajdla: Plots in image coordinates
IMGTEXT  Pajdla: Plots a text in image coordinates
MESHGRAY Pajdla: Plots gray mesh
PFIT     Pajdla: Plane fit using Least Squares or Least Winsorized Squares
PLOTLINK Pajdla: Plots a line links between points in 2D
PLOTTAB  Pajdla: Plots values from a matrix into a rectangular table
PRINTFIG Pajdla: Prints figures to the file or a printer
RAW2M    Pajdla: Reads raw binary images to Matlab
REPLICA  Pajdla: Replicates a matrix according to the pattern
ROTMAT   Pajdla: Rotates  columns in a matrix
SHOWIMG  Pajdla: Shows scaled images in the gray palette
SUBFIG   Pajdla: Opens figure in the specified part of the screen
UNISORT  Pajdla: Extracts all different values sorted in descending order
UNIVAL   Pajdla: Extracts all consequaent different values
VEC2LBLS Pajdla: Converts vector elements into a string matrix
```

```
LIM2M    Pajdla: Loads LILY images into Matlab
RQ       Pajdla: Returns a 3x3 upper triangular R and a unitary Q so that X = R*Q
PATHSEP  Pajdla: Path separator: Unix = '/', Dos = '\'
SDIV     Pajdla: Element-wise division with 1/0 replaced by 0
PMPATH   Pajdla: Returns the path to the standard Pajdla's directories
CHRFILL  Pajdla: Creates and fills matrix with a given character
```

Type `help nameOfProcedure` in Matlab to get more detailed information.

4.8 Portability

The system can be run on any machine under Matlab v-4.1 or higher version with external interface. C routines are written in ANSI-2 norm and have to be compiled by cmex script to build Matlab callable modules. In order to run the system on input image of size 761×512 pixels at least 12MB of virtual memory is requred for data storage. I have tested the system on HP-700, Silicon Graphics, Sparc-10, IBM-PC/486 and IBM-PC/Pentium compatible computers.

4.9 Installing BCRF

The distributon of the BCRF system can be obtained via ftp at

`ftp.esat.kuleuven.ac.be /pub/ROBOT/pajdla/bcrf.tar.Z`

In order to install the system do the following:

1. Load the file by ftp using binary mode. Uncompress and untar the file:

   ```
   >uncompress bcrf.tar.Z
   >tar -xvf bcrf.tar
   ```

   The following directory tree will appear in your current directory:

   ```
   ./BCRF          : Bcrf home directory (main routines).
   ./BCRF/BCRFCTRL : Standard control files and BCRFINST.M.
   ./BCRF/DATA     : Calibration data and output data are stored here.
   ./BCRF/DEMOIMGS : Images for the demonstration.
   ./BCRF/DOC      : Documentation (you read it now).
   ./BCRF/DOC/FIGS : Figures for the documentation.
   ./BCRF/OBSOLETE : Some obsolete routines (can be deleted).
   ./BCRF/UTILS    : Some other utility routines (do not delete).
   ```

2. Modify the definition of the type `CodeSubWordT` in the file BCRF.H. `CodeSubWordT`-s are used to form 64 bit long bit array for decoding and they have to be 32 bits long. Hence, on 32 bit machines use

   ```
   typedef unsigned long  CodeSubWordT;
   ```

   while on 64 bit machines use

   ```
   typedef unsigned int  CodeSubWordT;
   ```

The above example is only the demonstration. Actual type has to be chosen according the machine you use but it always should be a 32 bit unsigned integer.

3. Compile C routines. There are two make files, MAKEFILE.UNX and MAKEFILE.DOS. The first one has to be used on Unix machines with CC compiler while the second one applies for MS-DOS and MS-Windows with Watcom C compiler.

4. Set the variable `BcrfLocalization` in BCRFINST.M according to the place where you run the system.

   - **LEUVEN** must be used at Arrows. It forces the system to use the panel calibration data corresponding to the calibration pattern in Leuven. At the same time it determines that LILY image format is expected and that the image extension `ImgExt =` '.bcrf.lim'.
   - **PRAGUE** must be used at CVL, CTU, Prague. In this case different calibration data are loaded and BMP image format is used. The image extension `ImgExt` = '.bmp'.
   - **IWONL3** is used when doing off-line calibration of the setup for the Agricultural Department at KUL in order to measure and evaluate pigs. Raw, byte image of the size 512×512 pixels is expected with `ImgExt` = '.raw'. The panel calibration data has to be supplied in extra files CPNLXY.DTA and CPNLZ.DTA placed into `BcrfDataDir`.

   Edit variables `BcrfHomeDir` to point into the root directory of BCRF, e.g. `BCRF/`, `BcrfDataDir` to point into the data directory, e.g. `BCRF/DATA/`, and `BcrfImagDir` to point into the directory where you want to store images. The slash (or backslash on DOS machines) has to appear at the end of the strings.

5. Add `BcrfHomeDir`, `BcrfHomeDir`/UTILS, and `BcrfCtrlDir` into the Matlab search path. The best is to add the following code into MATLABRC.M:

```
path('user/BCRF',path);
path('user/BCRF/UTILS',path);
path('user/BCRF/BCRFCTRL',path);
```

   Then, BCRF directories are always included in the Matlab search path when Matlab is started. The above example adds to the search path BCRF home directory, utilities directory, and sets the control directory to the standard directory supplied with the distribution.

4.10 Running BCRF

To measure a range image I recommend the following strategy:

1. Prepare the experimental setup according to the Fig. 8 and [Vuy87, p. 162, Fig. 3.18]. If there is no calibration pattern and/or projector slide pattern run the task CODEPTRN.M to create two postscript files (CCALPTR.PS for the camera calibration pattern and PROJPTRN.PS for the projector slide pattern). Positions of camera calibration points are defined in PANELCAL.M.

2. Calibrate camera-world-projector coordinate systems using the task BCRFC.M. The task requires to specify which calibration panels are used by the variables `CamCalSlotNumbers` and `ProjcalSlotNumbers` in the `ParameterFileName`.M. These vectors are used to refer input calibration images. The output of the calibration into is saved the sensor matrix, SMATASC.`CalibrSessionId`, and linking search area parameters are saved into the file SPARASC.`CalibrSessionId`. Camera calibration matrix is saved into the file CMATASC.`CalibrSessionId`. The results of calibration are shown in a number of ways allowing to validate the feasibility of results.

3. When calibration is finished, measurement can be run using the task BCRFM.M. The task requires to input an image file name and a control parameter file name. If control file name is omitted BCRFMCTR.M is used. Calibration files SMATASC.`CalibrSessionId` and SPARASC.`CalibrSessionId` are loaded automatically. Be careful to set correctly the value of `CalibrSessionId` in the control parameter file. The output of the measurement "Code Bit Topological Network" (CBTN) is returned into the Matlab workspace or it can be saved into a Matlab data file (Set `SaveResults = [1]` in the control parameter file.) with the same name as the input image and '.nt' appended.

4. To visualize reconstructions, call BCRFS.M. The task requires to input original image either as a matrix or as a file name, a CBTN structure again either as a matrix in the workspace or as a file name, and a "clipping box".

5 Conclusion

BCRF range finder has two main advantages compared to active systems based on laser stripes, it is cheap and the acquisition time of one image is about 50ms. On the other hand, thare are still a number of possibilities how to improve the decoding as well as code bit network construction and code design itself. The first idea that comes to mind is to design more isotropic code residing for instance in an hexagonal instead of the rectangular grid. Moreover, since the errors in code words are mostly caused by occlusions of partially continuous surfaces they manifest themselves as block errors rather than completely random disturbations. This would suggest to optimize the code with respect to different criterion than that one used in Vuylsteke's thesis. Certain geometrical constraints like epipolar ordering constraints could be also used to avoid malfunction in undersampled parts of the scene or in higly noisy parts of the image. A lens of the projector equpped with the iris could increase the depth of focus field and thus improve overall performance of the system.

Acknowledgement

I want to thank to Rudi Baertels who provided me with the original source codes of Pieter Vuylsteke and Marc De Groof.

References

[Jar93] R. Jarvis. Range sensing for computer vision. In A.K. Jain and P.J. Flyn, editors, *Three Dimensional Object Recognition Systems*, volume 1 of *Advances in Image Communication*, pages 17–56. Elsvier Science Publishers B.V., 1993.

[Mat92] The MathWorks, Inc. *Matlab - External Interface Guide*, August 1992. e-mail: info@mathworks.com.

[PVG95] Tomáš Pajdla and Luc Van Gool. Matching of 3-D curves using semi-differential invariants. In *5th International Conference on Computer Vision*, Cambridge, MA, June 1995. (To appear, currently available upon request at Tomas.Pajdla@esat.kuleuven.ac.be).

[Van94] Emmanuel Van Der Stuyft. *Exterieurwaardering bij levende varens gebaseerd op driedimensionale*. PhD thesis, Katholieke universiteit Leuven, Belgium, May 1994.

[VO90] P. Vuylsteke and A. Oosterlinck. Range image acquisition with single binary-encoded light pattern. *IEEE Pattern Analysis and Machine Intelligence*, 12(2):143–164, 1990.

[Vuy87] P. Vuylsteke. *Een meetsysteem voor de acquisitie van ruimtelijke beelden, gebaseerd op niet-sequantielle belichtingscodering.* PhD thesis, Katholieke universiteit Leuven, Belgium., December 1987.