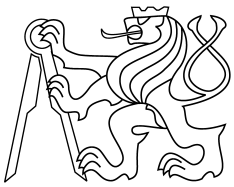
CENTER FOR
MACHINE PERCEPTION



CZECH TECHNICAL
UNIVERSITY

RESEARCH REPORT

ISSN 1213-2365

NIFTi Lidar-Camera Calibration

Vladimír Kubelka and Tomáš Svoboda

kubelvla@fel.cvut.cz, svoboda@cmp.felk.cvut.cz

CTU–CMP–2011–15

December 13, 2011

Available at
ftp://cmp.felk.cvut.cz/pub/cmp/articles/svoboda/Kubelka-TR-2011-15.pdf

This work was supported by EC project FP7-ICT-247870 NIFTi.
Any opinions expressed in this paper do not necessarily reflect the
views of the European Community. The Community is not liable
for any use that may be made of the information contained herein.

Research Reports of CMP, Czech Technical University in Prague, No. 15, 2011

Published by

Center for Machine Perception, Department of Cybernetics
Faculty of Electrical Engineering, Czech Technical University
Technická 2, 166 27 Prague 6, Czech Republic
fax +420 2 2435 7385, phone +420 2 2435 7637, www: http://cmp.felk.cvut.cz

NIFTi Lidar-Camera Calibration

Vladimír Kubelka and Tomáš Svoboda

December 13, 2011

Abstract

The NIFTi robot is equipped – among others – with a rotating laser scanner and an omnidirectional camera. This paper describes a practical approach for mutual calibration of the sensors. We make use of few publicly available packages and add few utilities for data conversion. A planar checker board pattern is shown at several 3D positions to the laser sensor and the camera simultaneously. Necessary correspondences between Lidar 3D points and images are assigned manually. The procedure yields a complete geometric calibration of the camera and transformation between the camera and the Lidar coordinate system.

1 Introduction

NIFTi project robot is – among others – equipped with a lidar and an omnicamera. The omnicamera is composed of 6 wide-angle cameras aligned to cover space around the robot, see Fig. 1. The output of each camera is an image composed of pixels $x_p = [u, v]^T$, which correspond to 3D points observed by a single camera. Let $X_C = [x_C, y_C, z_C]^T$ be such a point in a coordinate frame originating in the camera viewpoint and ${X_C}' = \frac{1}{\|X_C\|}[x_C, y_C, z_C]^T$ a unit vector pointing from the origin to X_C. A relation between ${X_C}'$ and the corresponding pixel can be described by a function F:

$$x_p = F({X_C}') \tag{1}$$

$${X_C}' = F^{-1}(x_p) \tag{2}$$

The function F depends on the camera and it encapsulates a transformation process precisely described in [1], in the section *Description of the calibration parameters* and summarized in [3], pages 9 and 10. The lidar (also in Fig. 1) provides 3D scans of the robot surroundings in a form of a pointcloud, i.e.

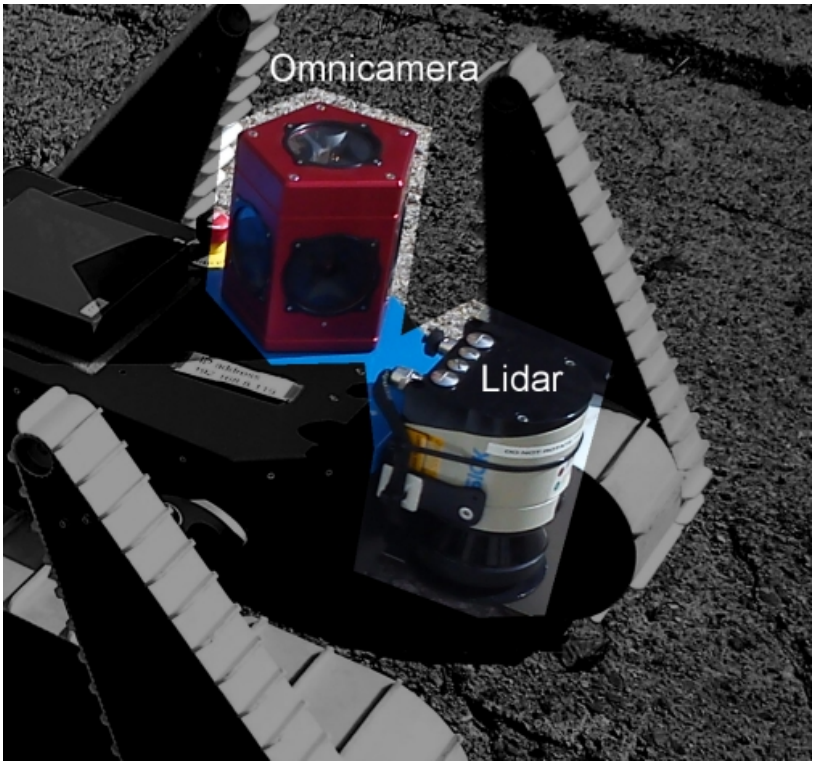


Figure 1: An omnicamera and a lidar attached to the NIFTi robot.

a set of 3D points $X_L = [x_L, y_L, z_L]$ expressed in the robot coordinate frame defined by the robot manufacturer. The two coordinate frames (of the camera and of the robot) do not coincide; thus, for a *single* point observed both by the lidar and by the camera applies

$$X_L \neq X_C \tag{3}$$

However, there is a rigid transformation between these two frames, which consists of rotation R and translation t and corresponds to the physical attachment of the camera relatively to the origin of the robot coordinate frame. Each point can be transformed simply as follows

$$X_C = RX_L + t \tag{4}$$

$$X_L = R^{-1}(X_C - t) \tag{5}$$

Basic transformation is provided by the Robot Operating System; is is probably based on the known physical dimensions of the robot. Thus, it can't grasp alignment imperfections of the omnicamera or lidar during robot assembly nor future manipulation (e.g. part replacement). For a more precise

3D computation, we need a better calibration procedure. We proposed use of the Laser-Camera Calibration Toolbox for MATLAB [3]; which is available online and also provided with this software. The toolbox cooperates with the Camera Calibration Toolbox [1], which provides data necessary for the calibration process.

The main idea of the calibration is a simultaneous observation of a checkerboard by the lidar and by a single camera. Under condition of taking several scans (15 advised) of the checkerboard at various orientations and distances – resulting into a set of 3D scans and camera images – the Camera Calibration Toolbox can determine the checkerboard plane normal and distance from the camera coordinate frame origin. The checkerboard plane be described by a plane equation

$$\theta_C^T X_C - \alpha_C = 0 \tag{6}$$

where θ_C is the normal of the plane and α_C is the distance from the camera frame origin. The Laser-Camera Calibration toolbox consequently lets user manually select 3D scan points lying in the checkerboard plane and determines the checkerboard plane normal and distance from the robot coordinate frame origin:

$$\theta_L^T x_L - \alpha_L = 0 \tag{7}$$

Finally, the calibration process evaluates translation t minimizing a difference between α_C and α_L for each plane using the least squares method. Rotation R is computed similarly minimizing a difference between θ_C and θ_L for each plane as well. The rotation R and translation t are subsequently refined minimizing an objective function of distance from the user selected 3D scan points to corresponding planes. Theoretically, a sufficient number of scan pairs would be 3; however, [3] advices taking at least 15 scan pairs which cover as many calibration plane orientations as possible in various distances.

We extended the procedure by a set of utilities for data processing as well as by a validation utility. In the current version of [3], a verification function is not implemented yet. Our validation utility transforms laser points to the camera frame and computes distance of each point from the corresponding plane. A histogram of all distances is plotted for each lidar-camera pair.

This report is composed of the following sections: The second section describes ways of installing necessary toolboxes and gives a reference to their documentation. It also mentions a modification to the [3], which allows usage of our validation utility. The third section is meant to be a step-by-step instruction set describing the calibration process. The fourth section describes all utilities provided in our software and finally, the fifth section introduces our validation utility.

2 Camera and Laser-Camera Calibration MATLAB Toolboxes

The Camera Calibration Toolbox provides the calibration plane normals and distances from coordinate frame origin, which are necessary for the Laser-Camera Calibration Toolbox. We use a version from July 27, 2010, which can be downloaded from its author web page [1]. To install the toolbox, follow instructions on the web page. It is a standard procedure; move a directory with the toolbox to your toolbox directory present in the Matlab main folder. Then add its path to the Matlab path list. The GUI is started by typing *calib* or *calib_gui*. A thorough camera calibration tutorial is available on the web page and it is strongly recommended to read it through and test it using a set of calibration images available on the web page as well.

The Laser-Camera Calibration toolbox installation is very similar to the previous one. It is described in [4]. If it is desired to verify results, the toolbox offers a point cloud colourisation based on the corresponding image. It may serve as a basic result verification. However, to statistically capture the calibration accuracy, we provide a simple validation utility. For it to work properly, there is one minor modification of the Laser-Camera Toolbox to be done after toolbox installation. It concerns rewriting m-file *select_from_range_fig.m* with our version (located in folder *laser_lidar_toolbox_mod*), which adds several code lines making the toolbox save selected laser points to *.mat* files. It is the only way to extract this information, the toolbox immediately computes plane coordinates and doesn't store the sets of selected points. A step-by-step tutorial concerning the Laser-Camera Calibration is described in [4].

3 Calibration Procedure

The calibration procedure can be divided into three main steps:

1. *Data acquisition* concerns series of checkerboard scans using *CTU Data Logger Node* for data collection and data conversion into appropriate formats. Capturing of a laser-camera scan is shown in Fig. 2 and discussed in detail in subsection 3.1.

2. *Camera Calibration* is done using The Camera Calibration Toolbox and results in *Calib_Results.mat* file, which is needed for the next step. A part of the process is show in Fig. 2 and discussed in subsection 3.2.

Figure 2: A single scan pair acquisition(left), checkerboard corners extraction (right)

3. *Laser-Camera Calibration* requires selection of sets of laser points lying in the checkerboard plane (Fig. 3), which is followed by the transformation matrix evaluation. This part is discussed in subsection 3.3.

3.1 Data acquisition

To perform a laser-camera scan acquisition, it is necessary to print a checkerboard pattern (the bigger the better, our was 500 x 650 mm) and fix it to a planar surface. There are two possible approaches, the first is to fix the pattern to a solid pad and change its proximity and orientation to the robot while taking scans; and the second one is to fix the pattern to the wall and move the robot around while taking scans from various positions. We have tested the first method because the actual version of the pointcloud producing ROS node took in account odometry, so the pointcloud was transformed relative to the origin of the odometry frame. The current version of the node has the ability to create pointclouds fixed to the robot frame; thus, the second approach is also possible.

While taking the scans, it is advised to stow the robot flippers the way lidar cannot see them (not the way showed in Fig. 2, where the flippers are not stowed enough and the lidar scans them instead of interesting surroundings) . The relative position of the checkerboard to the robot should cover various planes in various distances, including steep angles assuring accurate rotation matrix estimation; however, it must be possible for the Camera Calibration Toolbox to recognize the checkerboard pattern. It is shown in Fig. 4; the camera calibration was not able to detect pattern correctly in the scan on the right. The condition of various calibration plane orientations should be especially considered with the checkerboard fixed and robot moving - utilize

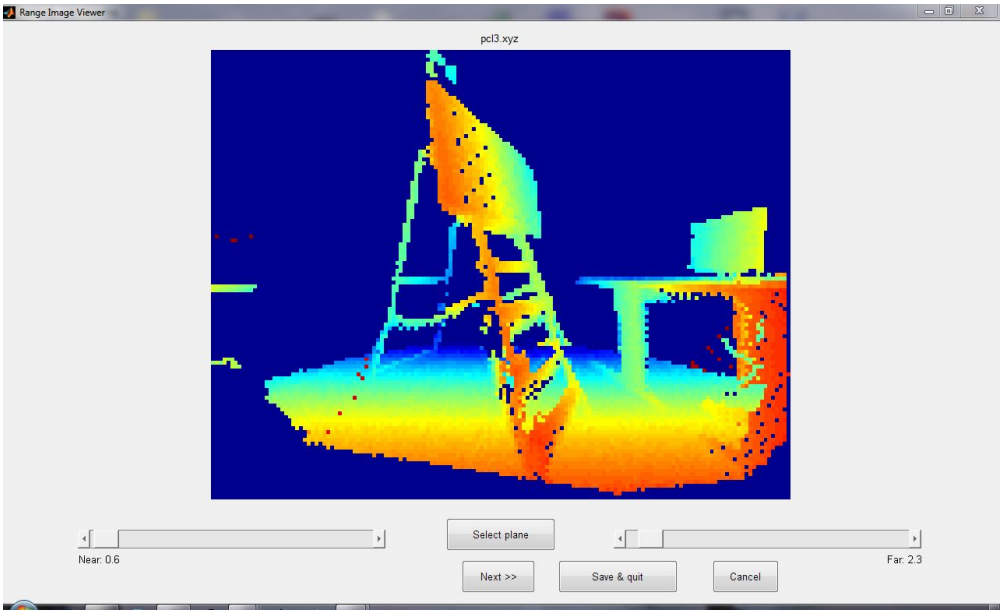


Figure 3: A checkerboard plane selection

the flippers to achieve more viewing angles.

In the Laser-Camera Calibration toolbox documentation, it is advised to take approximately 15 scans. For the initial calibration, we took 50 scans expecting some of them to be excluded. Finally, the calibration was done using 32 scans (it is for a further examination to determine optimal number of scans, because to acquire and process this amount of scans is considerably time-consuming). The entire checkerboard pattern must be visible in the camera image, this has shown to be tricky in the case of camera 4 because of the robot caterpillar tracks and rotating lidar. The tracks must be taken in account while positioning the board and the image has to be taken in the moment when the lidar reaches its extreme position so it doesn't block the view. Because of the viewing angle of the cameras, it is advised to choose positions of the checkerboard close to the robot, the size of the checkerboard in the camera image gets smaller very quickly with the increasing distance. In Fig. 5, there are all 32 planes used for calibration plotted in the camera coordinate frame.

3.1.1 CTU Data Logger Utilization

The procedure is straightforward:

1. Turn the robot on and let it boot up

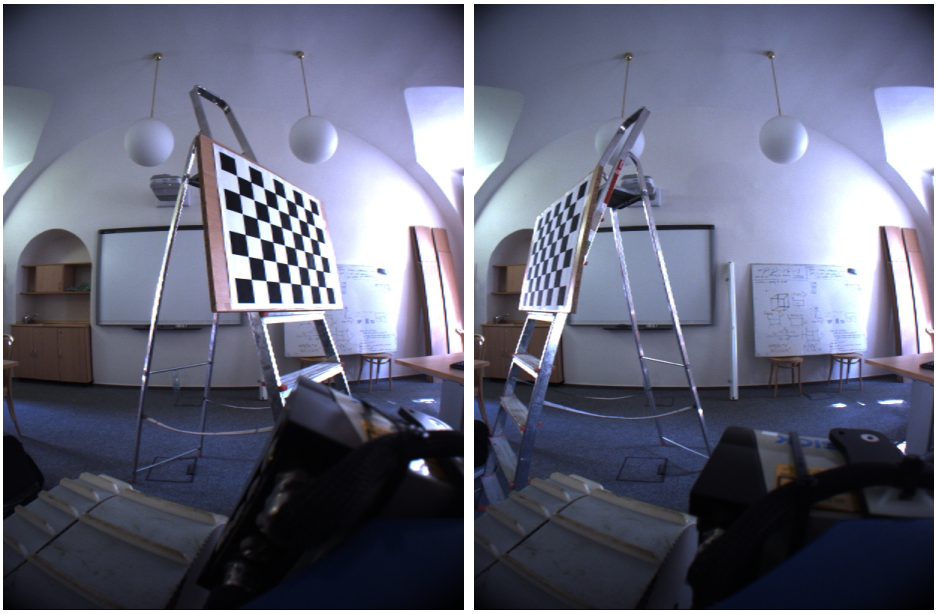


Figure 4: Two checkerboard scans, one used for the final calibration (left), second excluded for an angle too steep for pattern extraction (right)

2. Open a new terminal (directly on the robot or via ssh)

3. Launch the *CTU Data Logger node*, type: *roslaunch ctu_data_logger ctu_data_logger.launch*, wait several seconds until the launch file has launched all the nodes

4. Enable rotation of the 3D scanner using the remote controller: hold button 2 and set the rotation speed of the scanner using left and right cross-buttons. The rotation rate should be 180° per 10 seconds, the lower the rate is, the greater number of scanned points is saved making the calibration more reliable.

5. Launch the *Laser Assembler node*, type: *roslaunch nifti_laser_assembler nifti_laser_assembler_fixed_frame.launch*. Note that this is a modified version of the original launch file disabling an odometry transformation of the 3D scan. It is provided as a part of our software.

6. Press the button #3 on the remote to save a single scan-pair.

7. Copy the new files from the robot to your local folder. The files location is specified in the Logger launch file: *ctu_data_logger.launch*

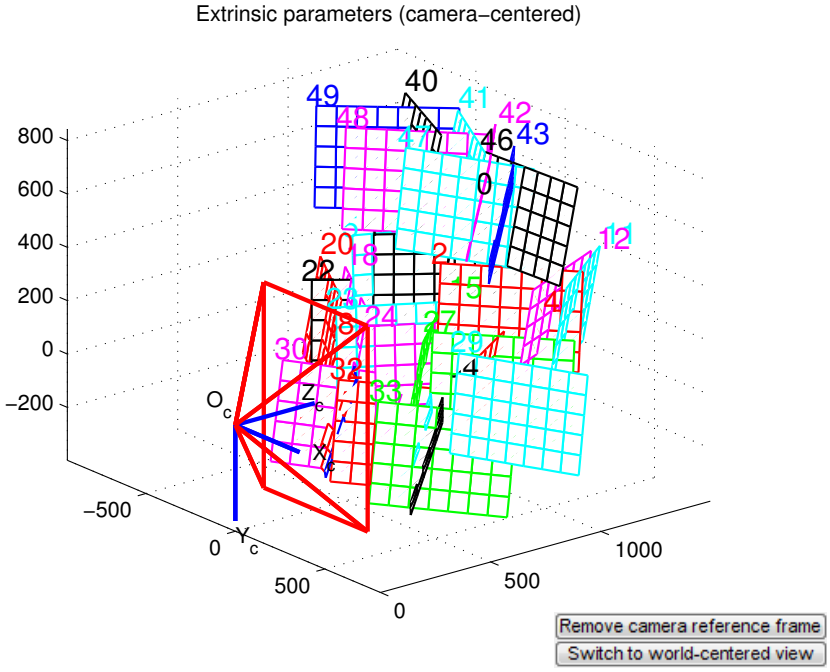


Figure 5: Example of plane positions during calibration, showed in the camera coordinate frame, axes unit is one millimetre, numbers correspond to scan filenames

3.1.2 Data Formatting

After completing the checkerboard scanning procedure, the user should copy files he acquired to a dedicated folder; possibly backup the files somewhere else as well; and do following:

- *File Indexing* should be modified the way the first number is 1. MATLAB utilities we provide will start searching for files with numbers from 1 to the maximal entered filenumber. This is to preserve MATLAB indexing conventions. If the user's dataset starts with 0, just rename the two files (pointcloud and image) to the highest number plus one.

- *CTU Data Logger PointCloud Format* needs to be converted into easily accessible *.asc* format. To do this, we provide *prcs_point_cloud_file.exe* utility. In the Windows environment, just select all 3D scan files to be converted (they have no suffix) and drag them over the *.exe* file. The utility will ask the user to choose to keep all points or to crop points lying on the robot hull, the second choice is advised. Then, the

application will create two new files for each 3D scan, *.asc* format is to be kept and *.pcd* may be deleted as it is intended for future work with PCL library.

- *Camera Images* must be saved in a format recognised by the Camera Calibration Toolbox. For the complete list, refer to the toolbox manual provided. There is a possibility in the ROS environment to compactly save all 6 images of the omnicamera to a single file (as in Fig. 14). Provided MATLAB utility *split_omni_pic.m* will split these files into single camera images and save them as *cam#_#.bmp*. The first number denotes the camera, the second one denotes the scan index. To use the utility, browse into the calibration folder under the MATLAB environment and type *split_omni_pic*.

- *3D scan .asc* files: the Laser-Camera Calibration toolbox was designed for a slightly different type of a lidar, a 3D scan produced by the NIFTi robot lidar surrounds coordinate frame origin instead of seeing forward only. This complicates the depth-view visualization necessary for manual plane selection. Thus, we provide a simple MATLAB script, that crops 3D scan to the viewing angle of the appropriate camera. As for all our MATLAB utilities, browse into the calibration folder in the MATLAB environment, type *import_and_crop* and follow the instructions. The script will create *.xyz* files corresponding to the original ones.

In the end of this calibration phase, the user should possess a set of 3D scan files in the *.xyz* format indexed from 1 as well as a set of images from the calibrated camera index correspondingly to the 3D scan file set. It is necessary that all files are named as *something#.xyz* and *somethingElse#.bmp* or another allowed image suffix.

3.1.3 Data Formatting Process Demonstration

This section demonstrates the data processing step-by-step. The dataset was reduced to 8 scan pairs to decrease time demands of this demonstration, the dataset is provided in the directory named *demonstration* and covers calibration of the camera #4:

We created a new directory in our MATLAB workspace and copied files provided by the *CTU Data Logger* into the folder. The second step was to convert 3D scan files into the *.asc* format. We did that by dragging the files over the *prcs_point_cloud_file.exe* and choosing to crop the points near origin by typing *a* and pressing enter (see Fig. 6 and 7). *Asc.* and *.pcd* files were created, the *.pcd* could be deleted and the original files were moved to a new

directory named *bck* (as *Backup*). After these steps, there should have been only *.png* image files and the new *.asc* files (see Fig. 8).

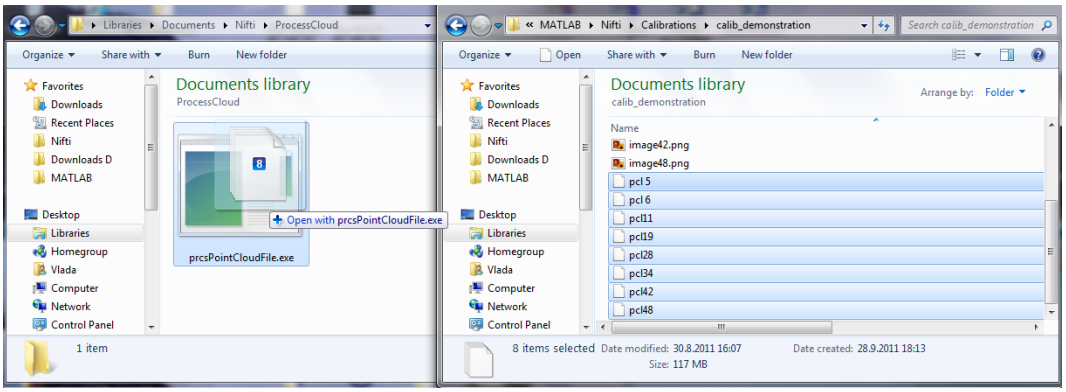


Figure 6: Dragging CTU Data Logger 3D scan files over the executable

The rest of the calibration process involved MATLAB, we navigated to the directory containing the calibration files as shown in Fig. 9. The first step in the MATLAB was to extract camera 4 images from the *.png* files. We did that by using the *split_omni_pic.m* script:

```
>> split_omni_pic
Enter omnicamera picture file basename (without number): image
Basename: image
Enter omnicamera filename suffix (without .): png
Suffix: png
Enter the highest filename number to process: 48
File image1.png not found, skipping.
File image2.png not found, skipping.
File image3.png not found, skipping.
File image4.png not found, skipping.
Warning: Image is too big to fit on screen; displaying at 67%
> In imuitools\private\initSize at 73
  In imshow at 262
  In splitOmniPic at 66
Saving cam0_5.bmp...
```

The script split the images into several parts, we chose only *cam4* files and deleted the rest since we were seeking for transformation between the camera 4 and the lidar frames. After this step, the directory looked like

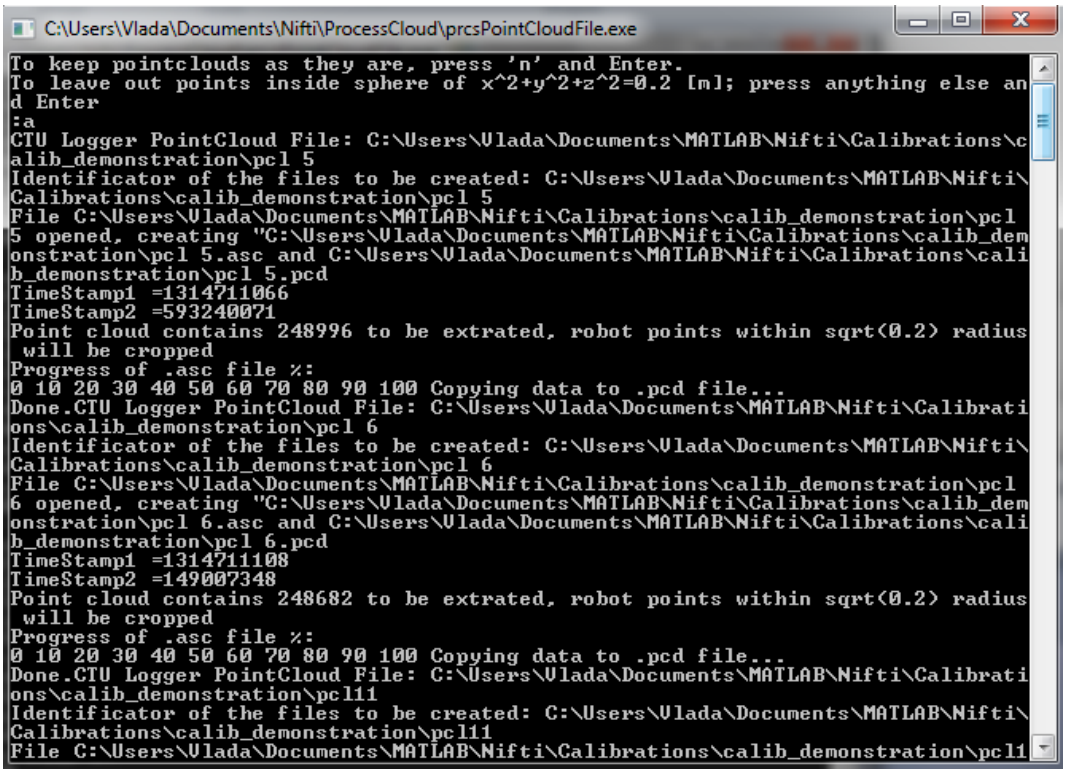


Figure 7: Converting the 3D scan files

Fig. 10. The next step was to crop 3D scans into the view angle of the forth camera by typing

```
>> import_and_crop
Enter the camera number (0, 1, 2, 3 or 4): 4
Enter basename of the lidar file (without number or suffix): pcl
Basename: pcl
Enter lidar file suffix (without .): asc
Suffix: asc
Enter the highest file number: 48
File pcl1.asc not found, skipping.
File pcl2.asc not found, skipping...
```

and deleting the *.asc* files leaving the new *.xyz* cropped 3D scan files; see Fig. 11. After these steps, we moved to a next phase involving the calibration toolboxes.

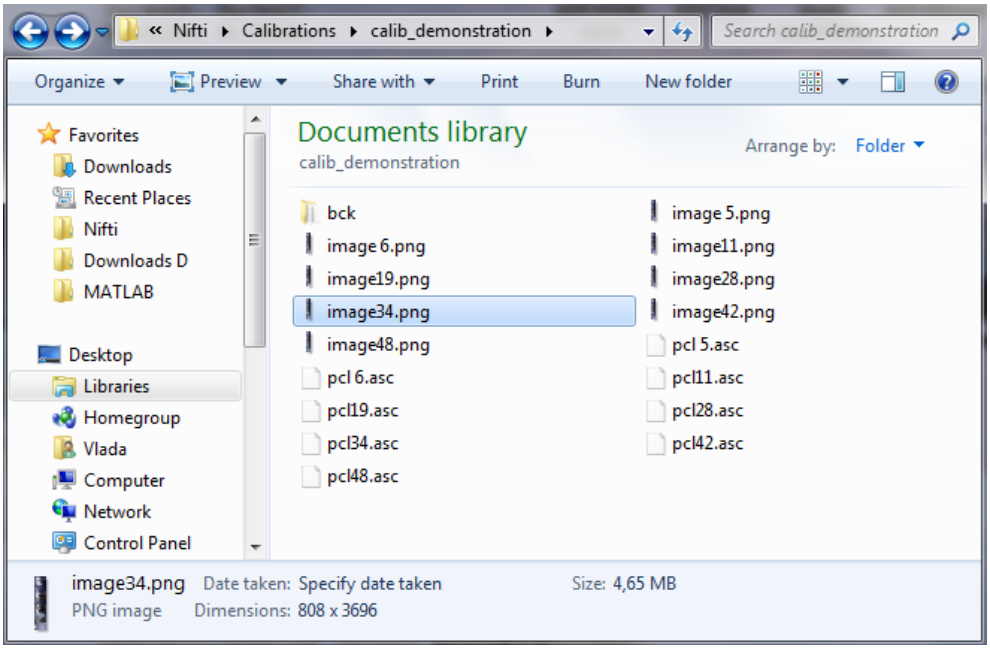


Figure 8: Original files backed up, *.pcd* files deleted and *.asc* files ready for the next step

3.2 Camera Calibration

With the Camera Calibration Toolbox installed, the camera calibration should follow the provided tutorial. The process consists of an initial manual checkerboard corners extraction and subsequent parameters tuning and launching calibration process. In the corner extraction process, the user is asked if satisfied with the initial distortion guess. For the ladybug camera, a value of $k = -0.2$ was found satisfactory for the most of images. The calibration can be finished when the user founds corners reprojection error satisfactory. This can be achieved by changing parameters available or suppressing problematic images. It is not mandatory that all scan images are used, the Laser-Camera Calibration Toolbox takes this possibility in account.

The calibration is completed by saving results into *Calib_results.mat* file. A Calib_results.m file is also created; yet, it is not necessary for further steps.

3.3 Laser-Camera Calibration

After installation of the Laser-Camera Calibration Toolbox, make sure that the file select_from_range_fig.m *(in* laser_lidar_toolbox_mod *folder) has been*

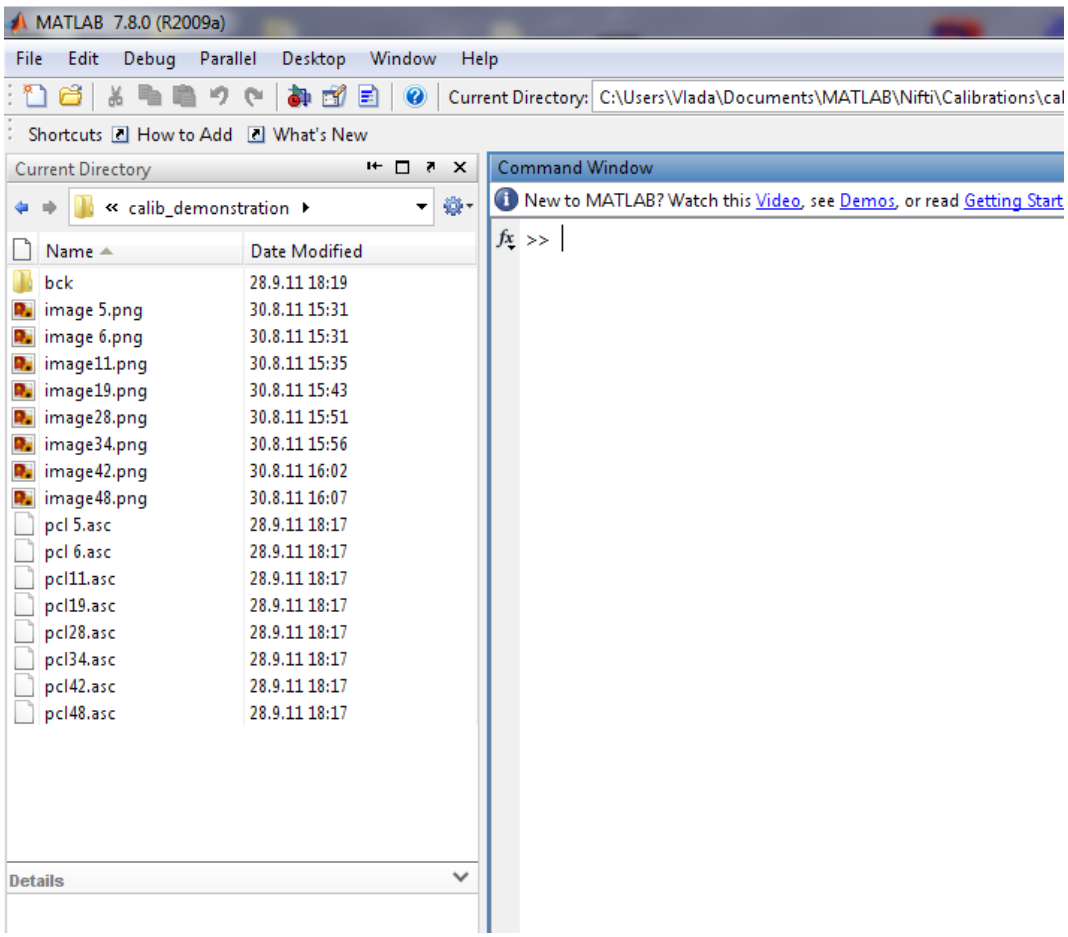


Figure 9: MATLAB environment and the current folder with calibration files

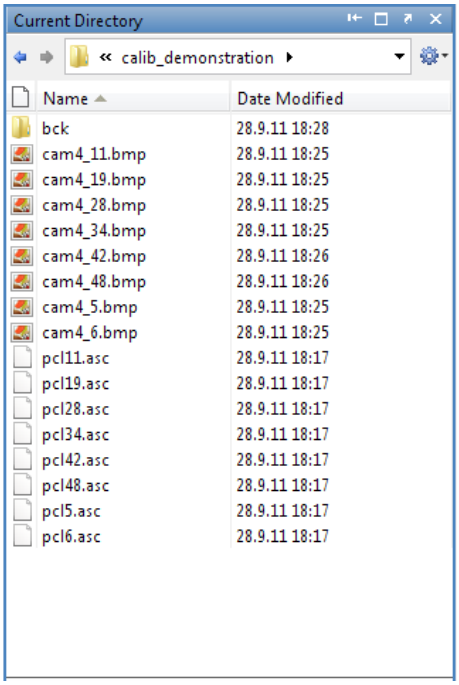


Figure 10: Images split and all but *cam4* deleted.

replaced by our version. This makes further calibration validation possible.

The toolbox is launched by typing *lasercamcalib*. It is also GUI based and the whole procedure is described in [4]. The user needs only the *Calib_results.mat* file and all the *.xyz* files that were just created. The actual version launched in MATLAB R2009a throws an error after clicking the *Select planes* button in the main menu. It can be ignored. The toolbox creates new *selected_plane_of_something#.xyz* files during plane selection as a result of our modification. Be patient, this may cause the GUI be a little slower after selecting plane.

The plane selection tool offers possibility to visualize depth range between two distance values set by sliders. Default values are 1 and 10 meters. They should be set so only the checkerboard is visible (as well as everything in the same range from the origin). This makes the selection easier. The difference is shown in Fig. 12 and Fig. 13.

After selecting checkerboard planes, the calibration is finished by clicking on the *Calibration* button and saving the result into two *.m* files. The result can be preliminarily validated by colourisation of a 3D scan. This function of the Laser-Camera Calibration Toolbox becomes available after finishing the calibration. Resulting *.wrl* file (VRML version 1.0) can be observed in

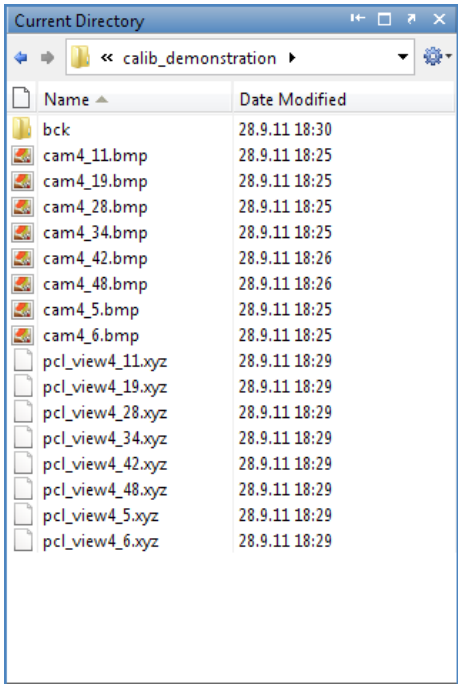


Figure 11: The directory ready for the Camera Calibration Toolbox

Cortona 3D viewer. Because of the format version, it may be necessary to translate it to VRML 2.0, which is required by certain 3D viewers (e.g. [2]).

In this moment, the calibration is finished and the resulting rotation a translation is saved in the two *.m* files created by the toolbox. Several additional *selected_plane_of_something#.xyz* files have been created, store them for further calibration verification.

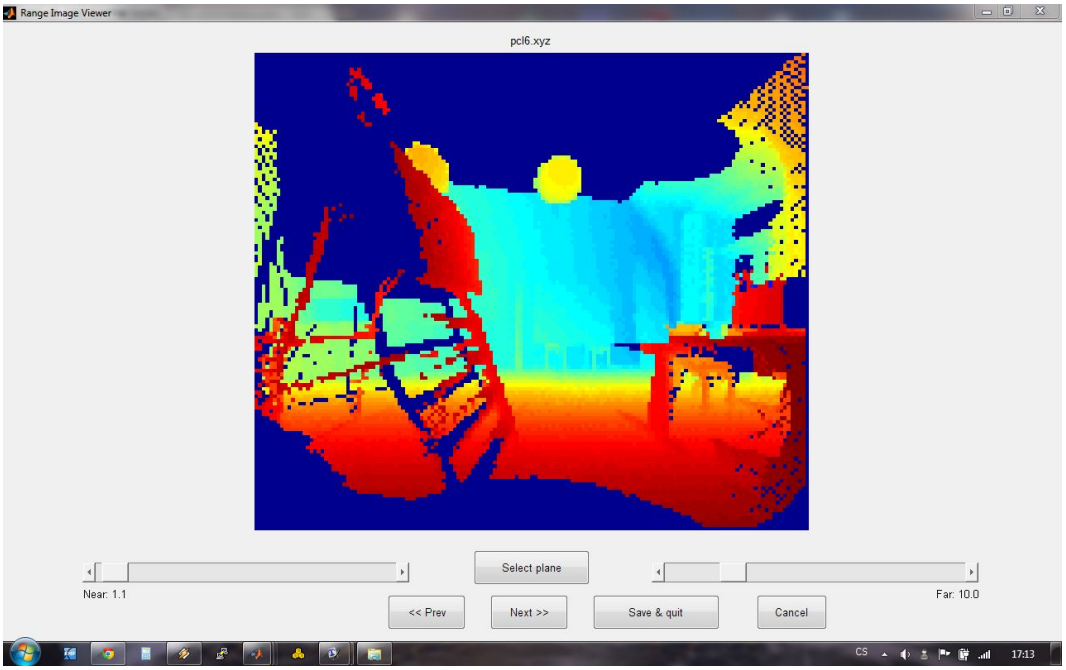


Figure 12: Example of a plane selection tool. NEAR and FAR sliders are set to their default values of 1 and 10 meters. In this case, different values are obviously more appropriate.

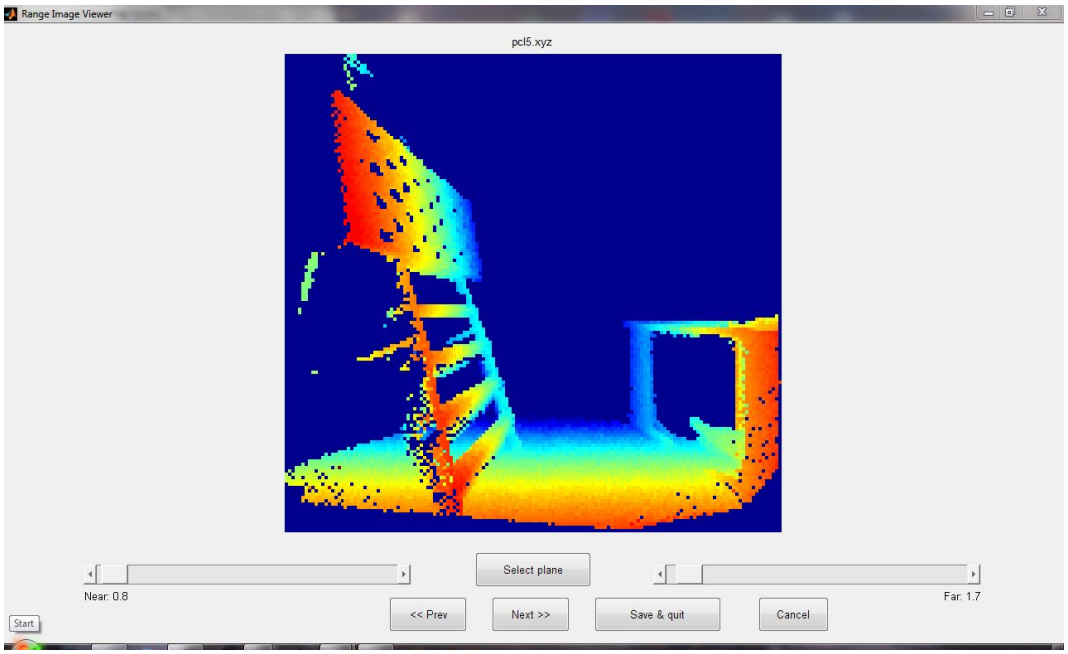


Figure 13: Example of a plane selection tool after modifying FAR and NEAR values according to the distance of the checkerboard.

4 Data Utilities

To make calibration easier, we have programmed several utilities to process captured data. Except one, all of them are MATLAB scripts. To be able to launch them from any directory in the MATLAB environment, copy directories *camUtils*, *laserUtils* and *validation* to a directory of your choice and add a path to this directory to your MATLAB path list. A more detailed description of data processing utilities follows:

prcs_point_cloud_file is an console application converting files created by *CTU Data Logger Node* (Appendix A) containing 3D scans into a simple ASCII *.asc* and *.pcd* format, cropping useful parts of the 3D scans and saving them to the folder with the input files. The application expects filenames of the input files as launch parameters. After starting the application, the user is asked to enter n in the case he wants to preserve all points of the cloud. Any other input will cause the application to crop out all points closer to the coordinate frame origin than a set threshold. This was found useful, partly because the scan of the robot itself is useless, partly because the incriminated area concerns significant amount of the laser points and thus increasing the 3D scan file size. A source code for a Linux and Win32 environment is available. The procedure concerning *CTU Data Logger Node* and scan acquisition will be described in Appendix A. The *.pcd* format is for future work with the PCL library (`http://pointclouds.org`).

In the Win32 environment, it is possible to simply select files to be processed and drag them to the executable. The system automatically provides filenames as parameters, so the user has to only choose whether he wants to crop robot surrounding or not. The process continues automatically. If the user prefers command line or works in a Linux environment, the filenames have to be passed as input parameters manually (if the 3D scan files are located in the same directory as the executable, only filename is to be passed, otherwise, a whole path to the file has to be passed as a launch parameter). The following command line printout demonstrates the usage:

```
C:\Users\Vlada\Documents\Nifti\ProcessCloud>dir
 Volume in drive C has no label.
 Volume Serial Number is 34F8-22E1

 Directory of C:\Users\Vlada\Documents\Nifti\ProcessCloud
```

```
28.09.2011  17:05    <DIR>          .
28.09.2011  17:05    <DIR>          ..
30.08.2011  15:34        15 668 093 pcl10
30.08.2011  15:35        15 707 138 pcl11
07.09.2011  13:18            33 280 prcs_point_cloud_file.exe
               3 File(s)     31 408 511 bytes
               2 Dir(s)  33 121 452 032 bytes free

C:\Users\Vlada\Documents\Nifti\ProcessCloud>prcs_point_cloud_file pcl10 pcl11

To keep pointclouds as they are, press 'n' and Enter.
To leave out points inside sphere of x^2+y^2+z^2=0.2 [m];
press anything else and Enter
:a

CTU Logger PointCloud File: pcl10
Identificator of the files to be created: pcl10
File pcl10 opened, creating "pcl10.asc and pcl10.pcd
TimeStamp1 =1314711284
TimeStamp2 =888448257
Point cloud contains 249876 to be extrated,
robot points within sqrt(0.2) radius
will be cropped
Progress of .asc file %:
0 10 20 30 40 50 60 70 80 90 100 Copying data to .pcd file...
Done.

CTU Logger PointCloud File: pcl11
Identificator of the files to be created: pcl11
File pcl11 opened, creating "pcl11.asc and pcl11.pcd
TimeStamp1 =1314711357
TimeStamp2 =544467254
Point cloud contains 250506 to be extrated,
robot points within sqrt(0.2) radius
will be cropped
Progress of .asc file %:
0 10 20 30 40 50 60 70 80 90 100 Copying data to .pcd file...
Done.
```

import_and_crop.m is a Matlab script converting *.asc* files into *.xyz* files ready for the Laser-Camera Calibration Toolbox. This script crops

points visible by a single camera (visibility determined approximately by dividing 2π range around the robot to five parts corresponding to the camera alignment) to allow the toolbox to visualize the pointcloud properly. After running the script in the directory containing the input files, the user is asked the number of the camera, the basename of the *.asc* files (a part of its name without number or suffix), then its suffix and the highest file number present. Then it proceeds through all *.asc* files that fulfil the conditions as shown below:

```
>> import_and_crop
Enter the camera number (0, 1, 2, 3 or 4): 4
Enter basename of the lidar file (without number or suffix): pcl
Basename: pcl
Enter lidar file suffix (without .): asc
Suffix: asc
Enter the highest file number: 48
File pcl1.asc not found, skipping.
File pcl2.asc not found, skipping...
>>
```

ply_to_asc_and_pcd.m converts *.ply* pointcloud file format to *.asc* and *.pcd* file formats. It was found useful since the application we used for pointcloud manipulation (registering, transformations) – MeshLab [2] – doesn't export pointclouds to *.asc* format and there are problems exporting to *.wrl* format. Therefore, we save the pointcloud in a *.ply* format and convert it back to *.asc* with this script. In MATLAB, the script is launched typing

```
>> ply_to_asc_and_pcd
```

and then choosing the *.ply* file in a GUI menu that appears. The output files are created in the actual MATLAB directory.

split_omni_pic.m is a tool for splitting *.png* images composed of all 6 camera views, which are one possibility of saving omnicamera output. The script output format is a windows bitmap *.bmp*. An example is shown in Fig. 14 and usage below:

```
>> split_omni_pic
Enter omnicamera picture file basename (without number): image
Basename: image
```

```
Enter omnicamera filename suffix (without .): png
Suffix: png
Enter the highest filename number to process: 5
Warning: Image is too big to fit on screen; displaying at 67%
> In imuitools\private\initSize at 73
  In imshow at 262
  In splitOmniPic at 66
Saving cam0_1.bmp
...
...
...
Saving cam4_4.bmp
Warning: Image is too big to fit on screen; displaying at 67%
> In imuitools\private\initSize at 73
  In imshow at 262
  In splitOmniPic at 66
Saving cam5_4.bmp
File image5.png not found, skipping.
Finished
```

5 Calibration Verification

Although the Laser-Camera Calibration Toolbox gui contains a *Plot Error* button, that function has not been implemented yet. For a more rigorous calibration error estimation than a 3D scan colourisation, we provide a verification MATLAB utility which provides this functionality accessible through gui (Fig. 15):

- *CamError* plots checkerboard corners reprojection error. It is the same tool [1] provides. In the figure, clicking the left mouse button on a reprojection error indicating cross makes the utility print originating image and error details to the MATLAB command window. Clicking the right mouse button terminates this picking mode. To use this function, the user is asked the original *Calib_results.mat* file exported by the Camera Calibration Toolbox.

- *CamPlanes* presents extrinstic parameters of all used calibration planes as a 3D plot. To run this function, the user has to provide the original *Calib_results.mat* file exported by the Camera Calibration Toolbox. This function originates from [1] as well.

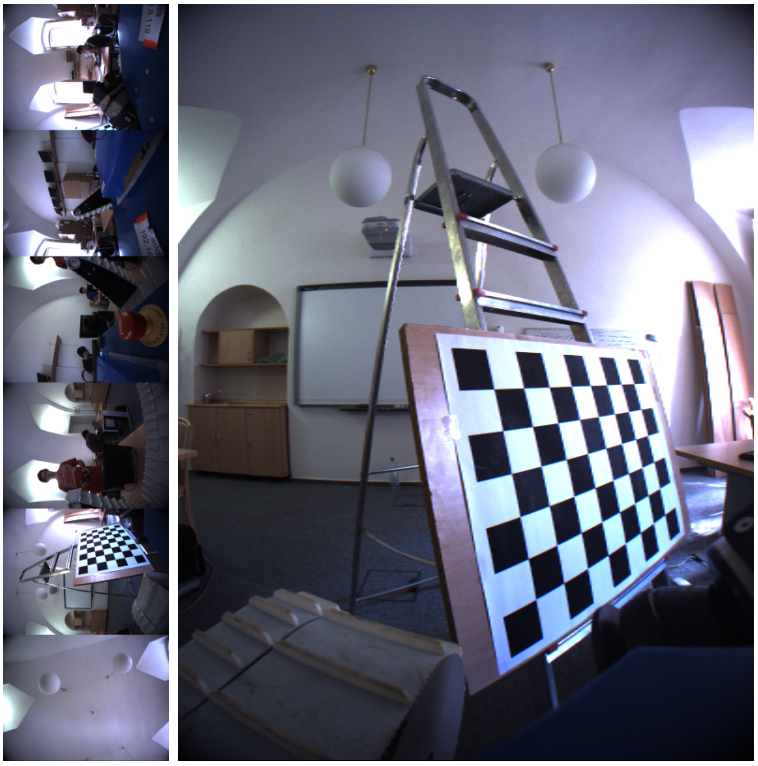


Figure 14: A single omnicamera scan saved as one image and a cropped view (camera 4)

- *RfError* is the main verification tool. It transforms laser points lying in the calibration plane to the coordinate frame of a camera and computes distance of each point from a corresponding calibration plane obtained during camera calibration. Results are plotted as histograms for each laser-camera pair (Fig. 16) and summarized in the last histogram plotted (Fig. 17). To use this function, the user is gradually asked for three *.mat* files; one containing selected laser points, one describing the calibration planes and finally one containing the rotation matrix and translation vector being verified. These files can be created by provided MATLAB utilities:

  - *process_laser_points.m* has to be launched in the folder containing *selected_plane_of_....xyz* files created during laser-camera calibration. After entering asked information, the utility creates the *.mat* file.

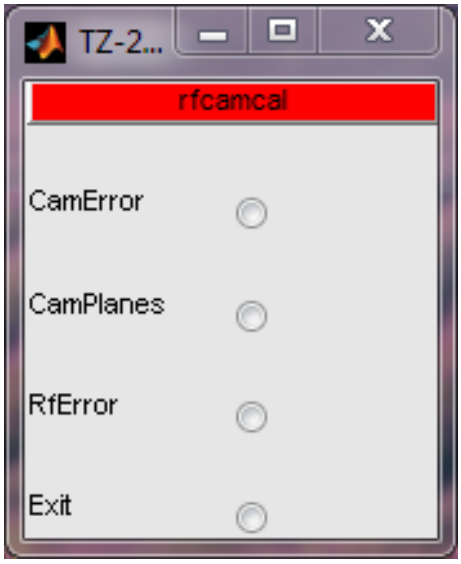


Figure 15: Calibration error estimation tool gui providing the functionality.

- *process_cam_calibration.m* asks the user to enter the *Calib_results.mat* file created during camera calibration and outputs the RfError demanded *.mat* file.
- *There is no* utility to create a *.mat* file containing the rotation matrix and translation vector since it is much faster to simply save these two variables to a *.mat* file in the MATLAB environment by selecting them both, right clicking on them and choosing *Save as..*

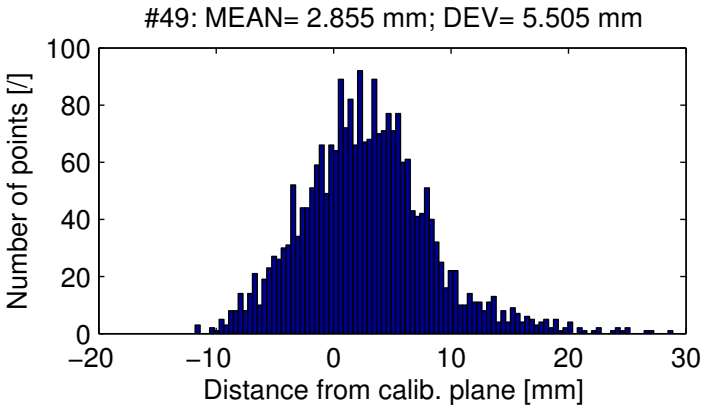


Figure 16: Histogram containing a single camera-laser pair distances between a laser point and the corresponding calibration plane. DEV stands for a standard deviation.

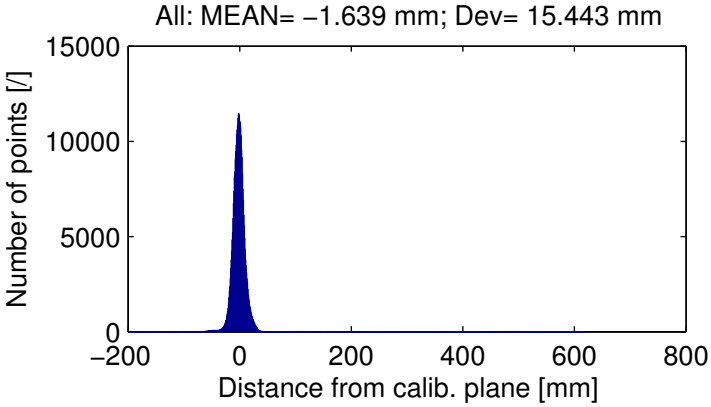


Figure 17: Histogram containing all distances between a laser point and the corresponding calibration planes. DEV stands for a standard deviation.

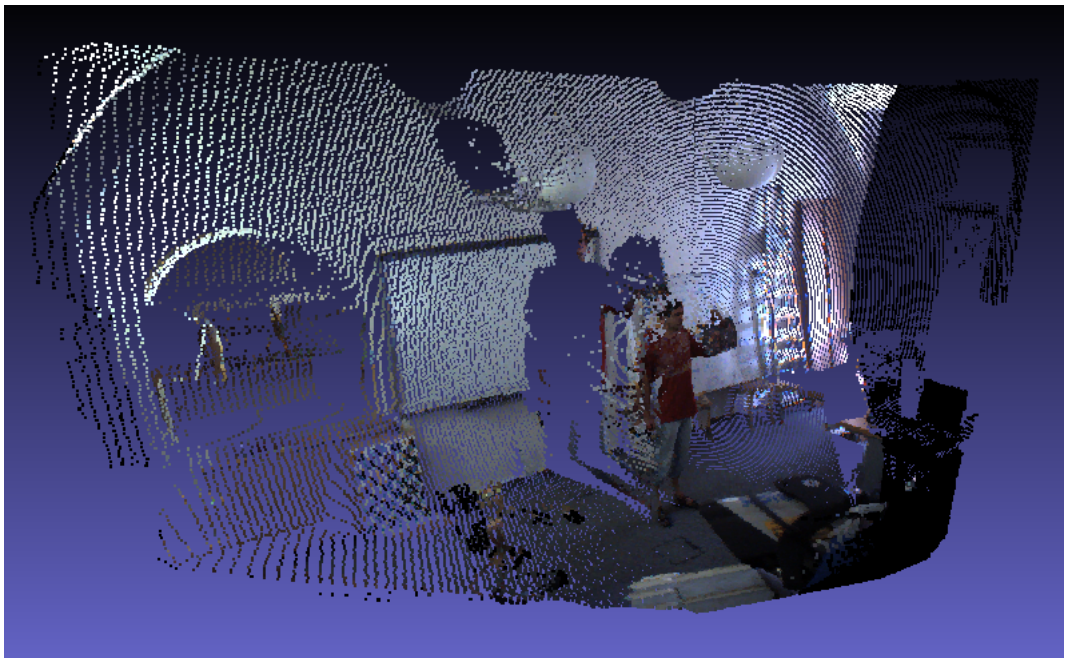


Figure 18: Colourised 3D scan.

A 3D Scan Colourisation

The Laser-Camera Toolbox offers a possibility to colourise a 3D scan based on the calibration results. This function can be ran immediately after finishing the calibration or subsequently by loading the calibration results. The procedure is described in [4]. If done properly, it should look like Fig. 18.

B Contents of the SVN Folder

This section describes a file structure of the calibration project SVN folder:

- *calibration_results* contains results of a calibration process performed on our NIFTi robot
  - *Calib_Results.m* is a MATLAB script file created by the camera calibration toolbox. It describes camera #4 intrinsic parameters.
  - *Calib_Results.mat* is a MATLAB *.mat* file created by the camera calibration toolbox. It contains a whole camera calibration toolbox workspace after finishing the camera calibration.
  - *laser_to_cam4_transformation.m* is a result of the laser-camera calibration toolbox procedure. It stores the rotation matrix and the translation vector.
- *camUtils* contains a single file:
  - *split_omni_pic.m* is the utility described in the section 4.
- *demonstration* contains example of files saved by the *CTU Data Logger*. These files are utilized in section 3.1.3.
- *documentation* contains this document.
- *laser_lidar_toolbox_mod* contains a single file:
  - *select_from_range_fig.m* replaces an original file in the laser-camera calibration toolbox and enables further calibration verification.
- *laserUtils* contains MATLAB utilities working with 3D scans:
  - *import_and_crop.m* loads a 3D scan in a *.asc* format and crops it to view-range of a specified camera.
  - *ply_to_asc_and_pcd.m* converts 3D scan formats according to its name.
- *prcs_point_cloud_file_win32_unix* contains C++ source of the 3D scan converting utility described in the section 4.
  - *CMakeLists.txt* is a file necessary for application *CMake* in Linux environment.
  - *prcs_point_cloud_file.cpp* is the source file. It is common for a Win32 compilation as well as for a Linux compilation.

  - *readme.txt* describes the compilation process for both systems (Win32 and Linux).

- *validation* contains files of the MATLAB calibration verification utility described in the section 5.
  - *LaserPoints* contains a MATLAB utility converting a set of *.xyz* files into one *.mat* file.
  - *PlaneCoordinates* contains a MATLAB utility extracting plane normals from a *Calib_Results.m* file and saving them to a new *.mat* file.
  - *tz2011* contains verification utility internal files.
  - *rfcamcal.m* launches the MATLAB verification utility GUI.
  - *validate.m* is another internal file.

References

[1] Jean-Yves Bouguet. Camera calibration toolbox for matlab. http://www.vision.caltech.edu/bouguetj/calib_doc/index.html, July 2010. Version: (27-July-2010).

[2] Guido Cignoni, Paolo; Ranzuglia. Meshlab. http://meshlab.sourceforge.net/, February 2011.

[3] Ranjith Unnikrishnan. The laser-camera calibration toolbox. http://www.cs.cmu.edu/~ranjith/lcct.html, July 2011. Version 1.1 (11/10/2006).

[4] Ranjith Unnikrishnan and Martial Hebert. Fast extrinsic calibration of a laser rangefinder to a camera. http://www.ri.cmu.edu/pub_files/pub4/unnikrishnan_ranjith_2005_3/unnikr%ishnan_ranjith_2005_3.pdf, July 2005.