A Portable Low-Cost Archaeological 3D Imaging and Cataloging System

Vaibhav Bajpai, Anuj Sehgal, and Daniel Cernea

Computer Science, Jacobs University Bremen
Campus Ring 1, 28759 Bremen, Germany
{v.bajpai, s.anuj, d.cernea}@jacobs-university.de

Abstract. This paper introduces a cross-platform low-cost system for 3D object reconstruction using a projection-based laser scanner. It uses contact-free measurement techniques for 3D object reconstruction and fast surface registration using Iterative Closest Point (ICP) [1]. The only hardware requirements are a simple commercial hand-held laser and a standard camera. The camera is initially calibrated using Zhang's camera calibration method so that its external and internal parameters are exactly known. The visible intersection with the background is used to find the exact 3D pose of the laser plane. This laser plane is used to triangulate new 3D point coordinates of the object's surface. The point clouds obtained are processed using 3D Toolkit (3DTK) [2] which includes an automatic high-accurate registration process and a fast 3D viewer.

Keywords: Camera Calibration, Hough Transformation, 3D Reconstruction, Scan Registration, Hand-Held Laser, Image Filters, Color Models

1 Introduction

introduction

2 Related Work

related work

The David Laser Scanner [3] is a software package that allows generation of 3D models of objects using a handheld laser and a cheap camera. The software package although originally free, is now available at a high price point. In addition, it can only run on Windows since currently it is heavily dependent on the .NET framework. Our paper intends to provide a free alternative to the David Laser Scanner. It is written in standard C++ and is dependent on OpenCV and 3DTK toolkits [2] making it a widely-available as a cross-platform solution.

3 Approach

The complete pipeline starting from capturing the video of the laser sweeping across an object to the end result of visualizing the 3D model of reconstructed point cloud is primarily divided into five major steps as shown in figure 1

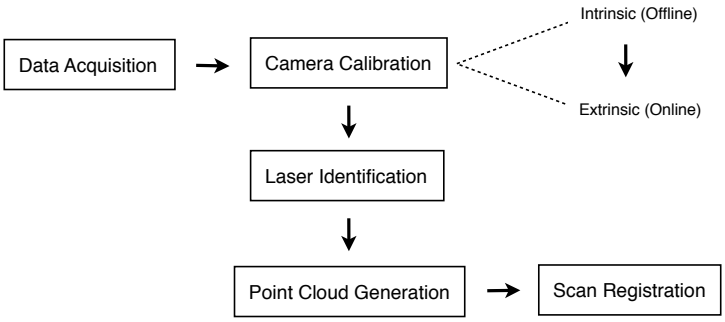


Fig. 1. Software Pipeline

3.1 Data Acquisition

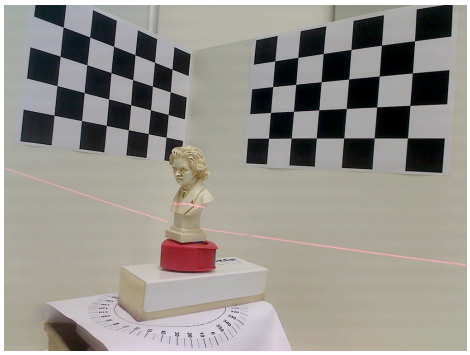


Fig. 2. Data Acquisition

We used a standard digital camera to capture multiple runs of a hand-held laser sweeping across the object as shown in figure 2. Since the videos were stored in raw format which cannot be directly processed by OpenCV [4], we used `mplayer` to extract individual frames at the rate of 5 frames per second.

```
$ mplayer -demuxer rawvideo \
          -rawvideo fps=5:w=1600:h=1200:yuy2 \
          -vo pnm:ppm $FILE
```

These frames were then later read into memory by calling the OpenCV routine `cvLoadImage()` with a `CV_LOAD_IMAGE_UNCHANGED` flag. The routine allocates an image data structure and returns a pointer to a struct of type `IplImage`

3.2 Camera Calibration

The first step in the process of reconstructing the 3D geometry of the object is to establish a mathematical relationship between the natural units of the camera with the physical units of the 3D world. We used camera calibration to learn the internal parameters of the camera and its distortion coefficients. The geometry is described in terms of camera's optical center and focal length.

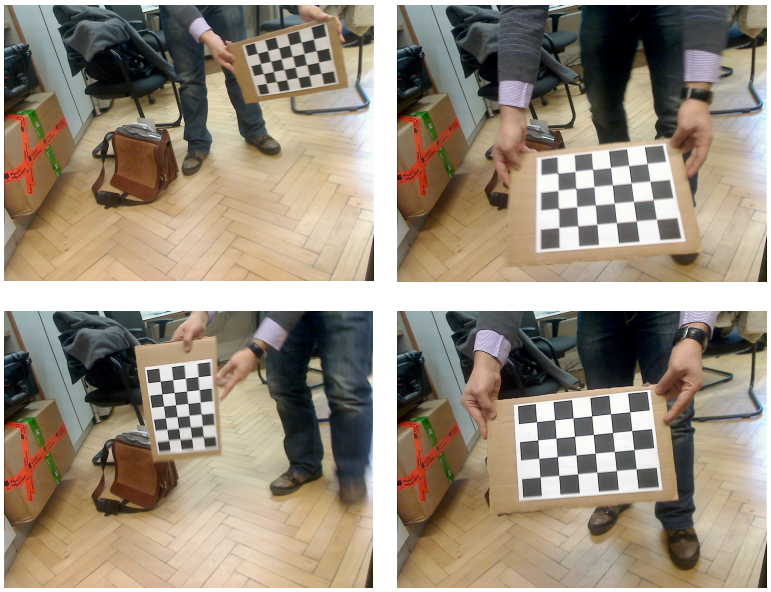


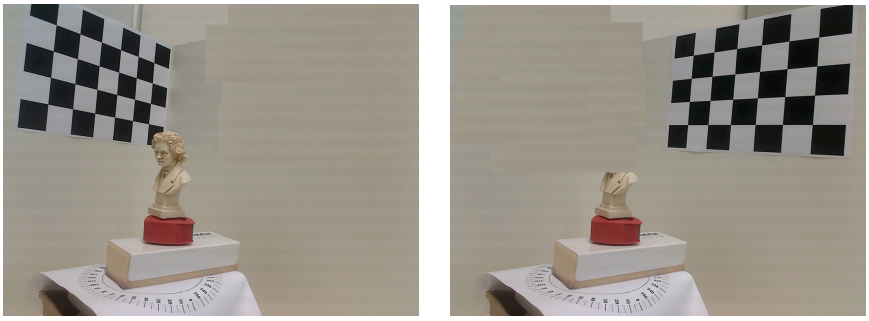
Fig. 3. Calculating the Camera's Intrinsic Parameters

We used OpenCV routines that are based on [5] [6] and used a planar chessboard pattern as our calibration object. We rotated and translated the pattern to provide multiple views to get the precise information about the intrinsic parameters of the camera as shown in figure 3. We used OpenCV routine `cvFindChessboardCorners()` to locate the corners and once we had enough corners from multiple view images, we used `cvCalibrateCamera2()` to get the intrinsic matrix A as shown in equation 1.

$$s \times \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = A \times \begin{bmatrix} R \mid T \end{bmatrix} \times \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \tag{1}$$

$$\text{where } A = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

The intrinsic matrix A was later used to describe the *pose* of the objects being scanned by the laser relative to the coordinate system of the camera. In order to determine this pose on both sides of the target object, the patterns were masked out to allow individual calculation as shown in figure 4. The parameters represented by $\left[R \mid T\right]$ could then be separately calculated for both the sides by calling the OpenCV routine `cvFindExtrinsicCameraParams2()`.



(a) $R_1 \mid T_1$ (b) $R_2 \mid T_2$

Fig. 4. Calculating the Camera's Extrinsic Parameters

3.3 Identification of 2D Laser Lines and Object Points

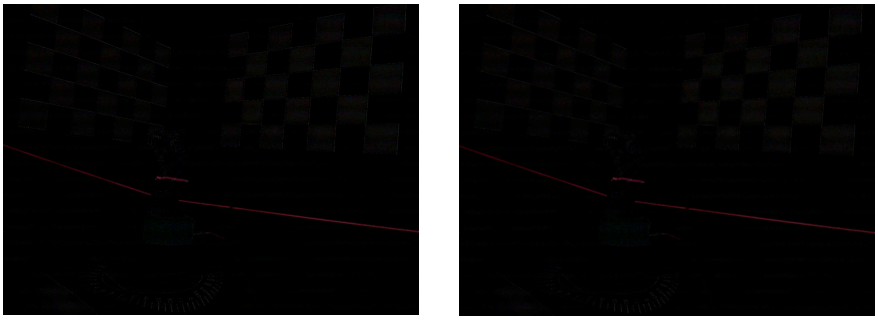
(a) A (b) B (c) C

Fig. 5. Using Image Difference to Find the Laser

We used OpenCV routine `cvAbsDiff()` to calculate the image difference of the laser image from the reference image using equation 2. The resulted image difference is shown in figure 5

$$C = A - B \tag{2}$$

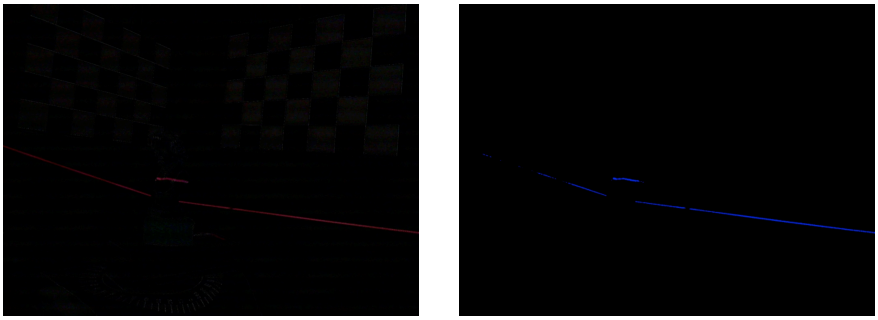
where A is the laser image in figure 5(a) and
B is the reference image in figure 5(b) and
C is the difference image in figure 5(c)



(a) Difference Image (b) Gaussian Smoothened Image

Fig. 6. Gaussian Smoothing the Difference Image

In order to reduce the noise in the difference image, we used the OpenCV routine `cvSmooth()` to convolve the image with a Gaussian kernel function as shown in figure 6. It not only helped us to remove the camera artifacts but also reduce the information content in the image.

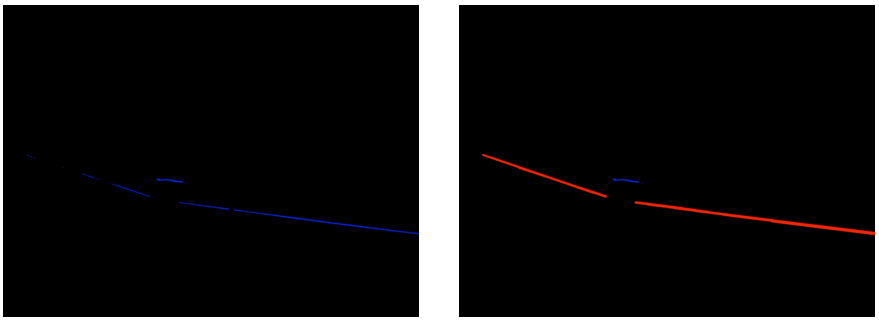


(a) Image with Outliers (b) Image without Outliers

Fig. 7. Color Thresholding to Remove Outliers

In order to blacken out (remove) all outliers and keep just the pixels representing the red laser line as shown in figure 7, we used a pre-defined threshold

value for the intensity of the red pixels. In order to restrict this thresholding only along the red channel, we used `cvSplit()` to split the three-channel (R,G,B) difference image into separate one-channel image planes. We used `cvGet2D()` and `cvSet2D()` to work on the scalar intensity values of the pixels.



(a) Before Transformation (b) After Transformation

Fig. 8. Hough Transformation

We used Probabilistic Progressive Hough Transform (PPHT) [7], [8] using OpenCV routine `cvHoughLines2()` to detect the laser lines on both sides of the target object. The line end points of each line thus obtained were used to draw the line using `cvLine()` as shown in figure 8. The difference image was initially passed through an edge detection phase, since hough transform not only expects a gray-scale image as input but the input is also treated as binary information where the non-zero points are edge points of the image. Therefore, we used OpenCV routine `cvCvtColor` to convert the RGB difference image to gray scale and `cvCanny()` to perform the Canny Edge Detection [9] before PPHT. The two hough line equations on either side of the object were used to learn the laser line points, while the points not identified as part of the hough line were taken as target object points.

3.4 Generation of Point Cloud

The first step is to use the calculated camera extrinsics $(R \mid T)$ for each side of the target object along with the intrinsic parameters (A) to transform each laser pixel (P_c) into 3D laser surface points (P_w) using equation 3. The laser pixel points are expressed in the homogenous coordinate system.

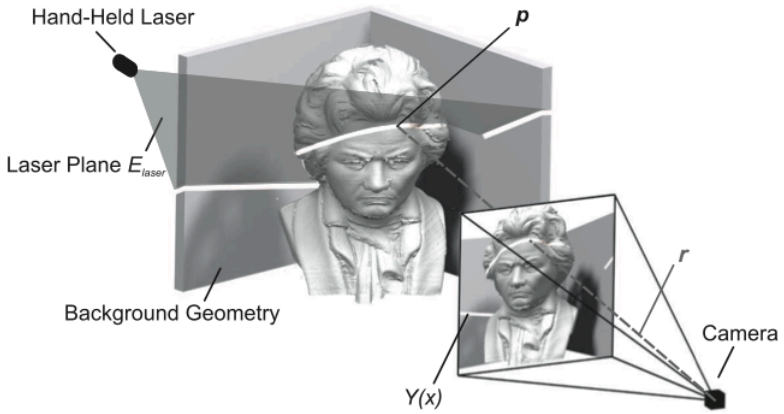


Fig. 9. Laser Triangulation [3]

$$P_w = \underbrace{s \times R^{-1} \times A^{-1} \times P_c}_{\overrightarrow{b}} - \underbrace{R^{-1} \times T}_{\overrightarrow{a}} \tag{3}$$

$$\text{where } P_c = \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}, \overrightarrow{a} = \begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix}, \overrightarrow{b} = \begin{pmatrix} b_x \\ b_y \\ b_z \end{pmatrix}$$

$$\text{and } s = \frac{a_z}{b_z}$$

Next in order to bring all the laser surface points into a common coordinate system, we transformed all the 3D laser points from the right side of the target object to the coordinate system of the left side using equation 4

$$P_l = R_1^{-1} \times P_r - R_1^{-1} T_1 \tag{4}$$

$$\text{where } P_r = \left[R_2 \mid T_2\right] \times P_w$$

With all the 3D laser surface points in a common coordinate system, we randomly choose 3 points to generate the laser plane equation. Using the coefficients of this equation, we could define the normal to the plane ($\overrightarrow{N}$) using equation 5.

$$A_x + B_y + C_z + D = 0 \tag{5}$$

$$\text{where } \overrightarrow{N} = \begin{pmatrix} A \\ B \\ C \end{pmatrix}$$

The last step is to use the target object pixels (P_c) and intersect them with the laser plane equation represented by the normal ($\overrightarrow{N}$) to obtain the 3D surface points of the target object (P_w) as shown in figure 9 using equation 6

$$P_w = s \times R^{-1} \times A^{-1} \times P_c - R^{-1} \times T \tag{6}$$
$$\text{where } s = \frac{\overrightarrow{N} \times \overrightarrow{a} - D}{\overrightarrow{b} \times \overrightarrow{N}} \text{ and } P_c = \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}$$

In addition, we used OpenCV routine `cvGet2D()` to retrieve the RGB color information for each object pixel and mapped it to the calculated corresponding 3D object surface point. We used the reference image as the source for retrieving the original color information since that image does not have a laser line sweeping the target object.

3.5 Point Cloud Processing and Registration

The generated point cloud was saved as a text file in the `UOS_RGB` file format. The first line represented the number of 3D object surface points and the following lines had (x, y, z) coordinates of each point respectively with their (R, G, B) information defined in a left-handed coordinate system. Each complete scan of the target object was in this way stored as a new `scanXXX.3d` file where `XXX` determines the index of the scan.

We used ICP [1] to register the two points from different scans into a common coordinate system using equation 7. The algorithm requires an initial starting guess of relative poses $(x, y, z, \theta_x, \theta_y, \theta_z)$ to compute the rotation and translation to fit the 3D geometrical models together. Since our system did not have an odometer, we set the initial pose to 0 and let it extrapolate further.

$$E(R,t) = \sum_{i=1}^{N_m} \sum_{j=1}^{N_d} w_{i,j} \|m_i - (Rd_j + t)\|^2$$
$$\begin{aligned} \text{where } N_m &= \text{number of points in model set } M \\ N_d &= \text{number of points in data set } D \\ w_{i,j} &= 1, \text{ if } m_i \text{ is closest to } d_j \\ w_{i,j} &= 0, \text{ otherwise} \end{aligned} \tag{7}$$

We used 6D Simultaneous Localization and Mapping (SLAM) from 3DTK [2] for automatic scan registration that uses cached kd-trees for fast iterative ICP match [10]. The `slam6D` component produced a `frames` file that was used by the `show` component to visualize the 3D model of the target object along with its color information.

Fig. 10. Results

4 Some Experimental Results

The imaging system in addition to the source directory, expects a reference image without the laser stripe and two images of the individual background patterns used for calibration as input. The program saves the point cloud thus obtained in a destination directory which is used by the 3DTK components for processing and visualization as shown below.

```
$ bin/projectionlaserscanner $SOURCE_DIR \
                             $REFERENCE_IMG \
                             $LEFT_CHECKERBOARD_PATTERN \
                             $RIGHT_CHECKERBOARD_PATTERN \
                             $DESTINATION_DIR \
$ bin/slam6D $DESTINATION_DIR
$ bin/show $DESTINATION_DIR
```

The results thus obtained from the `show` program are shown in figure 4.

5 Conclusion

conclusion

References

1. Besl, P., McKay, H.: A Method for Registration of 3-D Shapes. Pattern Analysis and Machine Intelligence, IEEE Transactions on **14**(2) (feb 1992) 239 –256
2. Automation Group (Jacobs University Bremen) and Knowledge-Based Systems Group (University of Osnabrück): 3DTK - The 3D Toolkit. http://slam6d.sourceforge.net/ [Online; accessed 6-May-2012].

3. Winkelbach, S., Molkenstruck, S., Wahl, F.M.: Low-Cost Laser Range Scanner and Fast Surface Registration Approach. In: Proceedings of the 28th conference on Pattern Recognition. DAGM'06, Berlin, Heidelberg, Springer-Verlag (2006) 718–728
4. Bradski, G., Kaehler, A.: Learning OpenCV: Computer Vision with the OpenCV Library. Software that sees. O'Reilly (2008)
5. Zhang, Z.: A Flexible New Technique for Camera Calibration. Pattern Analysis and Machine Intelligence, IEEE Transactions on **22**(11) (nov 2000) 1330 – 1334
6. Brown, D.C.: Close-Range Camera Calibration. Photogrammetric Engineering **37**(8) (1971) 855–866
7. Kiryati, N., Eldar, Y., Bruckstein, A.M.: A Probabilistic Hough Transform. Pattern Recogn. **24**(4) (February 1991) 303–316
8. Matas, J., Galambos, C., Kittler, J.: Robust Detection of Lines using the Progressive Probabilistic Hough Transform. Comput. Vis. Image Underst. **78**(1) (April 2000) 119–137
9. Canny, J.: A Computational Approach to Edge Detection. IEEE Trans. Pattern Anal. Mach. Intell. **8**(6) (June 1986) 679–698
10. Nuchter, A., Lingemann, K., Hertzberg, J.: Cached k-d Tree Search for ICP Algorithms. In: Proceedings of the Sixth International Conference on 3-D Digital Imaging and Modeling. 3DIM '07, Washington, DC, USA, IEEE Computer Society (2007) 419–426