# A Cross-Platform Open Source 3D Object Reconstruction System using a Laser Line Projector

IEEE GSC 2012, Passau


Vaibhav Bajpai and Vladislav Perelman

School of Engineering and Sciences
Jacobs University Bremen
Bremen, Germany


November 2012

# Overview

- Motivation and Goals
- Approach
  - Data Acquisition
  - Camera Calibration
  - Identification of 2D Laser Lines and Object Points
  - Point Cloud Generation
  - Point Cloud Processing and Registration
- Experimental Results
- Future Work, Conclusion

# Overview

- Motivation and Goals
- Approach
  - Data Acquisition
  - Camera Calibration
  - Identification of 2D Laser Lines and Object Points
  - Point Cloud Generation
  - Point Cloud Processing and Registration
- Experimental Results
- Future Work, Conclusion

# Motivation

- active contact-free triangulation-based 3D object reconstruction techniques have been known for more than a decade
    - structured light method used by Microsoft Kinect
    - Stereophotogrammetry used in Google Maps
    - time-of-flight method used in engineering industry
- rely on high-precision expensive actuators to move the laser, depend on external sensors to track the scanner
- there is a need for a low-cost solution.

# Motivation

- David Laser Scanner initially started to solve this issue.
  - it uses self-calibration to eliminate the need of external sensors
  - the concept has been published as a research paper [1]
  - the package is no longer free, runs only on Windows


- A need for a free alternative to the David Laser Scanner

# Goals

- Cross-platform
    - written in standard C++
    - uses OpenCV and 3DTK (available on Mac, GNU/Linux, Windows)
- Open-source
    - Fork us on Github*
- Free
    - utilized by a low-cost inexpensive hardware

This paper presents how we used these programming tools as basic building blocks to bring the David Laser Scanner concept into reality.

* https://github.com/vbajpai/projectionlaserscanner

# Overview

- Motivation and Goals
- Approach
  - Data Acquisition
  - Camera Calibration
  - Identification of 2D Laser Lines and Object Points
  - Point Cloud Generation
  - Point Cloud Processing and Registration
- Experimental Results
- Future Work, Conclusion

# Data Acquisition

- a hand-held laser sweeps across the object, while an inexpensive web camera captures these multiple runs
- mplayer to extract frames

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

- read frames using OpenCV

```
IplImage *img =
cvLoadImage (
             filename.c_str(),
             CV_LOAD_IMAGE_UNCHANGED
           );
```

# Camera Calibration

to establish a mathematical relationship between the natural units of the camera with the physical units of the 3D world

- intrinsic calibration
- extrinsic calibration

```
vector<CvMat*> cameraParameters =
camera->calibrate(imageList);
```

| | |
|---|---|
| 0 | cameraMatrix |
| 1 | rotationVector (R1 and R2) |
| 2 | translationVector (T1 and T2) |

camera extrinsics

points in image plane

$$s \times \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = A \cdot [R \,|\, T] \cdot \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}$$

points in the world coordinate system

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

camera intrinsics

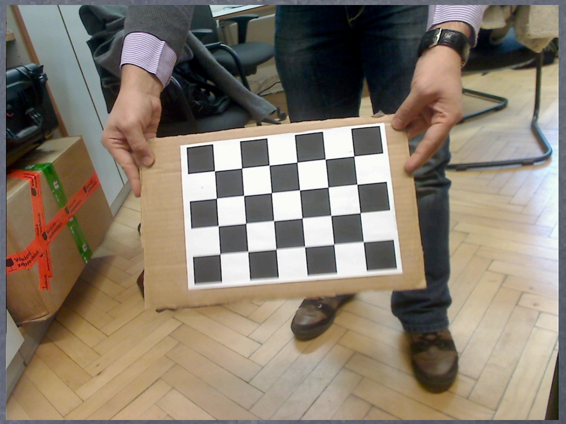
# Intrinsic Calibration

- calibration object: planar chessboard pattern
- use OpenCV to locate corners

```
int ifFound =
cvFindChessboardCorners (
                          img,
                          cvSize(WIDTH, HEIGHT),
                          corners,
                          &cvFindCornerCount,
                        );
```

- rotate/translate the pattern to provide multiple views
- use OpenCV to calculate intrinsic matrix

```
cvCalibrateCamera2 (
                    objectPoints, imagePoints
                    pointCounts, cvGetSize(img),
                    cameraMatrix, distCoeffs,
                    rvecs, tvecs
                  );
```



# Extrinsic Calibration


- patterns are masked to allow individual calculation
- use OpenCV to calculate camera extrinsics

```
cvFindExtrinsicCameraParams2 (
                               objectPoints, imagePoints,
                               cameraMatrix, distCoeffs,
                               rvecs, tvecs
                             );
```

# Identification of 2D Laser Lines

apply image processing methods to discern 2D laser points and 2D object points

- image difference to find the laser line
- smoothen the difference image to reduce noise
- color threshold the smoothed difference image to remove outliers
- hough transform to calculate a laser line
- identify the object points

```
vector <vector <CvPoint>> pointWrapper =
scanner->findLaser(image);
```

| | |
|---|---|
| 0 | Left Laser Points |
| 1 | Object Points |
| 2 | Right Laser Points |

# Difference Image

–

- use OpenCV to calculate difference image

```
cvAbsDiff (
            src,
            referenceImage,
            differenceImage
          );
return differenceImage;
```

# Smoothen and Color Threshold

- use OpenCV to smoothen the difference image

```
dst = cvCloneImage(src);
cvSmooth(src, dst, CV_GAUSSIAN, 5, 5, 0, 0);
return dst;
```

- remove camera artifacts
- reduce information content

- use OpenCV to color threshold the smoothed image

```
cvSplit(smoothedImage, srcB, srcG, srcR, NULL);
for (int i=0; i<(differenceImage->width); i++) {
 for (int j=0; j<(differenceImage->height); j++) {
 ...
   if (cvGetReal2D(srcR, j, i) < 50) {
     /* darken every non-laser pixel */
   } else {
     /* color laser pixel as RED */
   }
```

- removes all outliers

# Hough Transform

- use OpenCV to convert thresholded difference image to gray scale

```
cvCvtColor(src, dst, CV_RGB2GRAY);
```

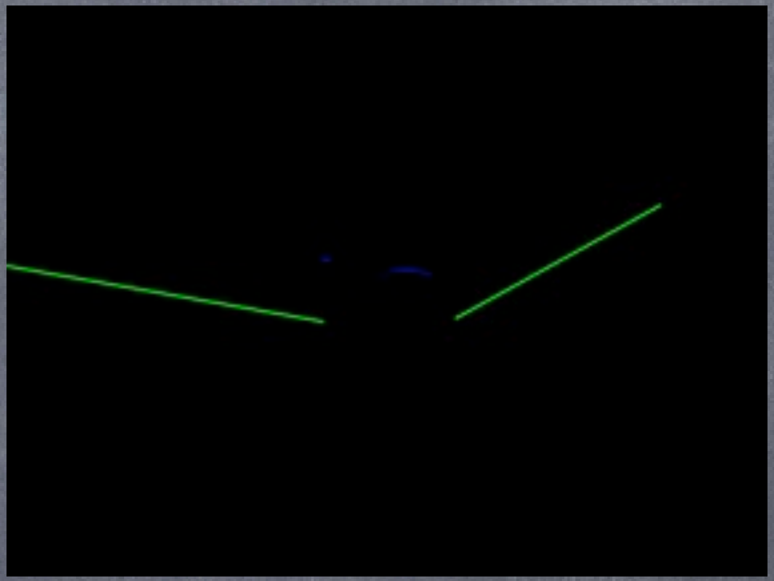
  - Canny edge detection expects a gray scale image.

- use OpenCV to perform Canny edge detection

```
cvCanny(dst, cannyImage, lowThresh, highThresh, 3);
```

  - Hough transform expects a binary image.
  - Non-zero points of the input image should be edge points.

# Hough Transform



- use OpenCV to perform Hough transform

```
CvSeq* line = cvHoughLines2(
                            cannyImage, storage,
                            CV_HOUGH_PROBABILISTIC, 1, CV_PI/180,
                            houghThresh, houghParam1, houghParam2
                           );
for (int i=0; i<line->total; i++) {
   CvPoint* lineEndPoints = (CvPoint*) cvGetSeqElem(line,i);
   cvLine(src, lineEndPoints[0], lineEndPoints[1], CV_RGB(255, 0, 0), 5);
}
return src;
```

# Discern Laser and Object Points

- Pack pixels and return

```
cvSplit(src, srcB, srcG, srcR, NULL);
for(int i=0; i<finalImage->width; i++){
  for(int j=0; j<finalImage->height; j++){

    if(cvGetReal2D(srcB, j, i) > OBJECT_THRESH){
      object.push_back(cvPoint(j,i));
      ifLeftLaser = false;
    }

    if(cvGetReal2D(srcR, j, i) > LASER_THRESH){
      if(ifLeftLaser == true) leftLaser.push_back(cvPoint(j,i));
      else rightLaser.push_back(cvPoint(j,i));

...
pointWrapper.push_back(leftLaser);
pointWrapper.push_back(object);
pointWrapper.push_back(rightLaser);
return pointWrapper;
```

# Point Cloud Generation

use camera parameters to calculate 3D laser and 3D object points

- calculate 3D laser points using camera extrinsics
- transform right 3D laser points to left coordinate system
- calculate laser plane equation using 3 laser points
- calculate 3D object points by intersecting laser plane and light ray
- append the pixel color information from the reference image

```
vector <Point3DRGB*> pointCloud =
pointCloud->generate(cameraParameters, pointWrapper);
```

| |
|---|
| CvPoint3D32f point3D |
| int RED |
| int GREEN |
| int BLUE |

# Laser Plane Equation

- calculate 3D laser points using camera extrinsics

$$P_w = \underbrace{s \cdot R^{-1} \cdot A^{-1} \cdot P_c}_{\vec{b}} - \underbrace{R^{-1} \cdot T}_{\vec{a}}$$

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

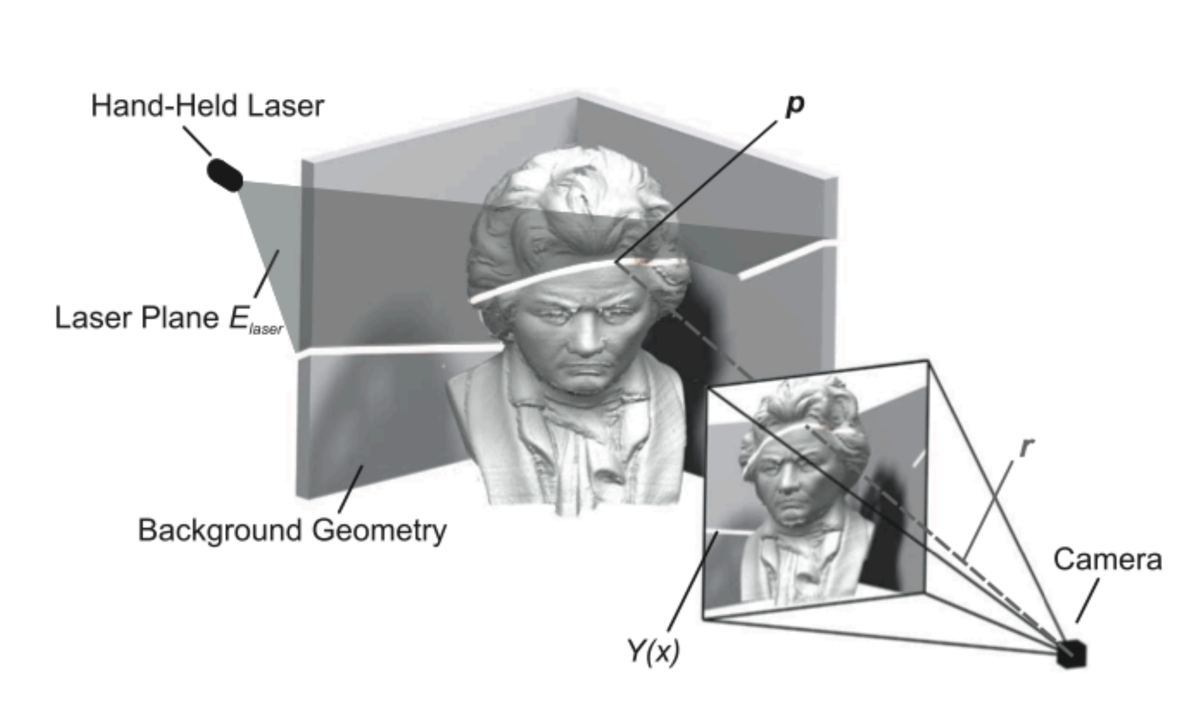
$$\text{and } s = \frac{a_z}{b_z}$$

- transform right 3D laser points to left coordinate system

$$P_l = R_1^{-1} \times P_r - R_1^{-1} T_1$$
$$\text{where } P_r = [R_2 \mid T_2] \times P_w$$

- calculate laser plane equation using 3 laser points

$$E_x + F_y + G_z + H = 0$$
$$\text{where } \vec{N} = \begin{pmatrix} E \\ F \\ G \end{pmatrix}$$




Laser Triangulation [1]

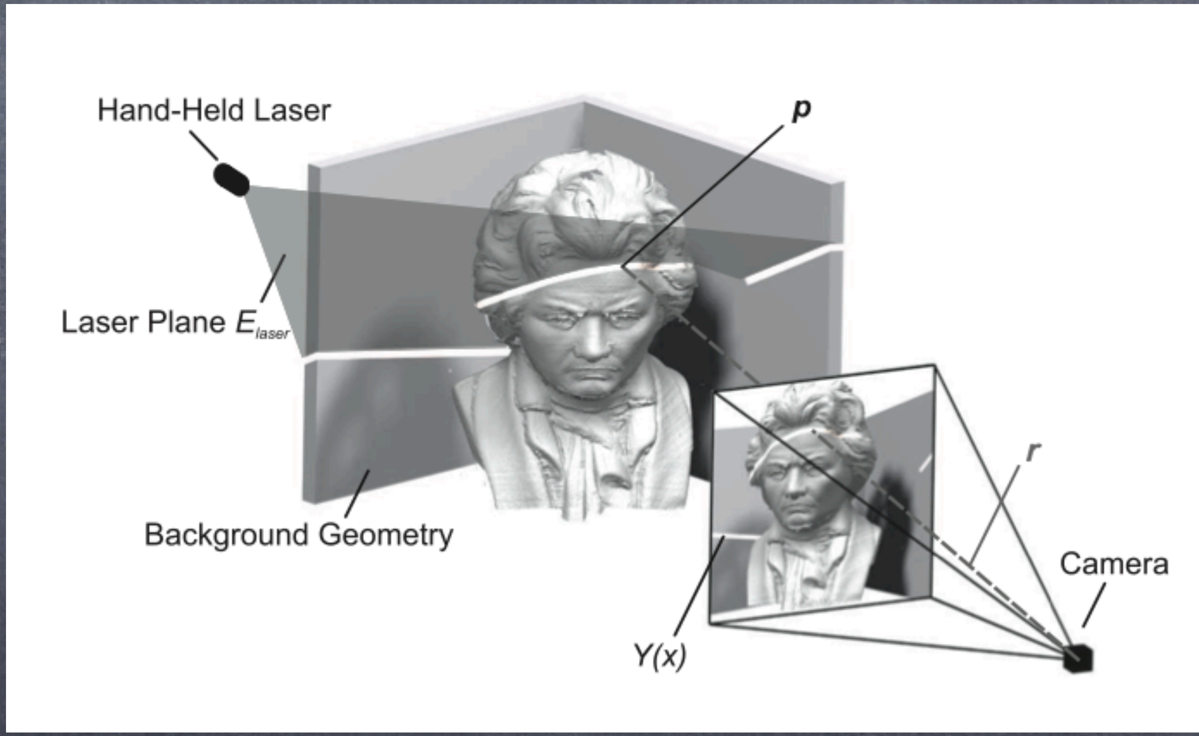
```
vector <double>
PointCloud::getPlaneEquation(CvMat* p1, CvMat* p2, CvMat* p3);
```

# Laser Triangulation

- intersect the object pixels with the laser plane to obtain 3D surface points of the target object

$$P_w = s \times R^{-1} \times A^{-1} \times P_c - R^{-1} \times T$$

$$\text{where } s = \frac{\overrightarrow{N} \times \overrightarrow{a} - D}{\overrightarrow{b} \times \overrightarrow{N}} \text{ and } P_c = \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}$$




Laser Triangulation [1]

```
vector <double>
PointCloud::get3DPoint(CvPoint point2D, vector<double> *plane);
```

# Colorize the Point Cloud

- use the information from the reference image to add color to the target object pixels.

```
for (unsigned int i=0; i < object.size(); i++) {
  ...
  CvScalar s =
  cvGet2D(referenceImage, object[i].x, object[i].y);

  pointCloud.push_back(
                      new Point3DRGB(cvPoint3D32f(x, y, z),
                      s.val[0], s.val[1], s.val[2])
                    );
}

return pointCloud;
```

# Registration

apply ICP [2] to register two points clouds from different scans into a common coordinate system.

$$E(R,t) = \sum_{i=1}^{N_m} \sum_{j=1}^{N_d} w_{i,j} \| m_i - (Rd_j + t) \|^2$$

$$\begin{aligned}
\text{where } N_m &= \text{number of points in model set } M \\
N_d &= \text{number of points in data set } D \\
w_{i,j} &= 1, \text{ if } m_i \text{ is closest to } d_j \\
w_{i,j} &= 0, \text{ otherwise}
\end{aligned}$$

- requires initial starting guess of relative poses
- the system lacks an odometer, we set initial pose to 0
- use 6D SLAM from 3DTK [3] for fast ICP match and visualization.

# Overview

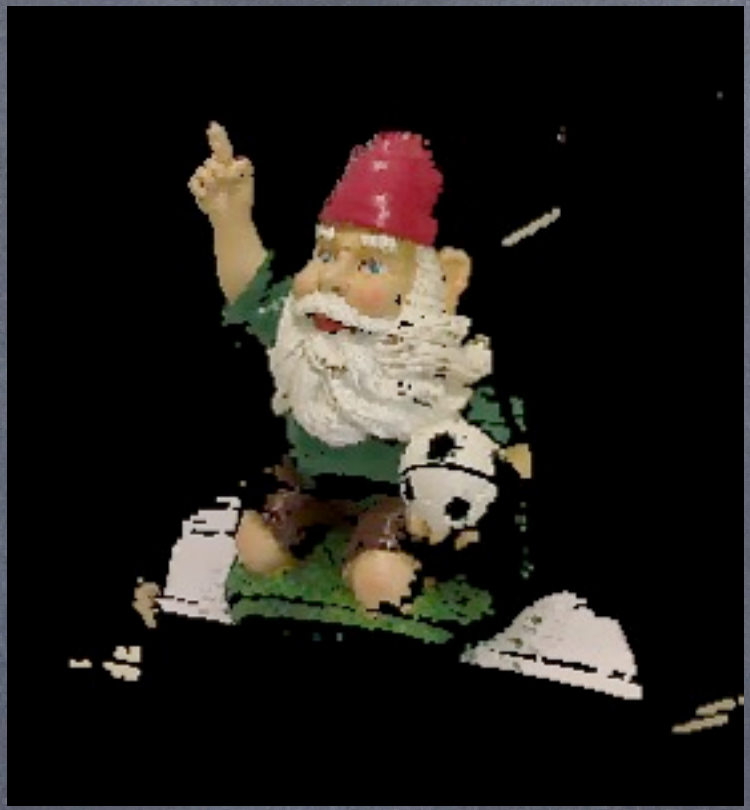
- Motivation and Goals
- Approach
  - Data Acquisition
  - Camera Calibration
  - Identification of 2D Laser Lines and Object Points
  - Point Cloud Generation
  - Point Cloud Processing and Registration
- Experimental Results
- Future Work, Conclusion

# Experimental Results



- discernible amount of noise is evident
- need to put the quality/price of hardware under consideration
- fast swipe of the laser creates gaps in the point cloud

# Experimental Results



- scan registration using ICP did not yield good results
- rotation angle between two scans maybe was too large.
- SLAM ICP could not converge the results.

# Overview

- Motivation and Goals
- Approach
  - Data Acquisition
  - Camera Calibration
  - Identification of 2D Laser Lines and Object Points
  - Point Cloud Generation
  - Point Cloud Processing and Registration
- Experimental Results
- Future Work, Conclusion

# Future Work

- real-time process of data acquisition, point-cloud generation and scan registration
    - get immediate feedback
    - help adjust the speed of laser sweep
    - help ascertain the required frequency of swipes
- performance evaluation with a larger dataset
- one-to-one comparison with the David Laser Scanner

# Conclusion

- 3D object reconstruction using a laser line projector and a web camera
  - free
  - cross-platform
  - open-source
- alternative to David Laser Scanner
- point clouds obtained are registered using SLAM from 3DTK [3] and viewed using its fast viewer

# References


(1) Low-Cost Laser Range Scanner and Fast Surface Registration Approach
Winkelbach, S., Molkenstruck, S,. Wahl, F.M.
DAGM 2006, Berlin, Heidelberg.

(2) A Method for Registration of 3D Shapes.
Besl, P., McKay, H.
IEEE Transactions on Pattern Analysis and Machine Intelligence 1992

(3) Automation Group (Jacobs University Bremen) and Knowledge-based Systems Group (University of Osnabrück)
3DTK - The 3D Toolkit.
http://slam6d.sourceforge.net/