Range Sensing for 3D Object Reconstruction and Visualization

by

Seha Kim

B.S., Department of Mineral and Petroleum Engineering
Seoul National University, 1998

M.S., Department of Mineral and Petroleum Engineering
Seoul National University, 2000

Submitted to the Department of Civil and Environmental Engineering
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Civil and Environmental Engineering

at the

Massachusetts Institute of Technology
September 2003

© 2003 Massachusetts Institute of Technology
All rights reserved

Signature of Author

Department of Civil and Environmental Engineering
August 15, 2003

Certified by

Ruaidhri M. O'Connor, Assistant Professor,
Department of Civil and Environmental Engineering
Thesis Supervisor

Accepted by

Chairman, Departmental Committee on Graduate Students
Range Sensing for 3D Object Reconstruction and Visualization
by

Seha Kim

Submitted to the Department of Civil and Environmental Engineering
on August 15, 2003 in Partial Fulfillment of the
Requirements for the Degree of Master of Science in
Civil and Environmental Engineering

ABSTRACT
3D photography is a technique to create detailed computer graphics models of real world objects. The most important step in 3D photography is range sensing: obtaining the 3D coordinate information of the points on the object.

We have proposed a range sensing method with an active source such that the scheme will result in a better accuracy but still maintain low cost. The scheme uses a line laser source to improve the accuracy compared to passive sources and a digital camera emerging as inexpensive tools for ubiquitous sensing reducing the cost of the overall system.

The range sensing system operates as follows. The active laser light source has used to project a line pattern on an object. The object has been placed on a linear stage that is connected to a stepper motor. A digital camera captures the projected laser stripe on the surface of the object. The stepper motor and the digital camera are controlled and operated using the ProRAST.NET window application.

We have applied the range sensing technology and visualized models from range images as the result of range sensing. A test object has scanned and the data is triangulated with the camera projection properties resulting from a camera calibration. The triangulation returns 3D coordinates of the points on the surface of the object with respect to the camera reference frame. This 3D information is integrated and constructed into a range image digitizing the 3D geometry of the object. The experiments have shown that this method and the range sensor give reliable results.

Thesis Supervisor: Ruaidhri M. O'Connor
Title: Assistant Professor, Department of Civil and Environmental Engineering
Acknowledgements

First of all, I would like to thank my advisor Ruaidhrí M. O'Connor who showed enormous patience and encouragement to me. I would not have come this far and finish my master’s thesis without him. Also, I want to express my sincere gratitude to Kevin Amaratunga who has been a good counselor.

Life in MIT was fun mainly because I had good friends with whom I can share my life, and I would like to, at least, mention their names: Jaehyuk, Jisoo, Saeyoon, Yeonjoo, Yoon-jung, and Junghoon.

I’d like to thank my friends since we met in high school, Mijung, Eunjoo.

Finally, I want to thank my parents, my sisters, Wonha and Jungha, and my brother, Jiha, who have supported me with love throughout my life. Thank you and I love you all.
Contents

1 Introduction .................................................. 10
  1.1 3D Photography and Range Sensing ....................... 10
  1.2 Related Work ............................................. 11
  1.3 Goals of Thesis ........................................... 12

2 Range Sensing ............................................... 14
  2.1 Range Image .............................................. 14
  2.2 Range Sensor ............................................. 14
  2.3 Optical Triangulation ..................................... 16
  2.4 Optical Triangulation Process ............................. 18

3 Range Sensor ............................................... 20
  3.1 Hardware .................................................. 20
  3.2 Software .................................................. 22

4 Camera Calibration ......................................... 27
  4.1 Projection and Intrinsic Parameters ........................ 27
  4.2 3D Rigid Transformation and Extrinsic Parameters .... 30
  4.3 Camera Calibration Method ................................ 32

5 Triangulation ............................................... 35
  5.1 Plane in B-dual-space Geometry .......................... 35
  5.2 Triangulation with B-dual-space Geometry ............... 36

6 Detection .................................................... 39
B.4 Triangulation ............................................................... 83
B.5 Reconstruction ............................................................. 84
B.6 Motor Control ............................................................... 87
B.7 PLY File ................................................................. 89
List of Figures

Figure 1.1: Flowchart for Implementation ................................................................. 11
Figure 2.1: Taxonomy of the Method to Acquire Shape of Object .................. 15
Figure 2.2: Taxonomy of the Optical Range Sensing ........................................ 16
Figure 2.3: Optical Triangulation Geometry ........................................................... 17
Figure 2.4: Optical Triangulation Sensor System ............................................... 18
Figure 3.1: Scanning Equipment ........................................................................... 21
Figure 3.2: Flowchart of ProRAST.NET Operation ............................................ 23
Figure 3.3: ProRAST.NET Login Window Form to Open Session ................... 24
Figure 3.4: ProRAST.NET Window Form for Connection to the Server ............ 24
Figure 3.5: Remote Controls Window Form .......................................................... 25
Figure 3.6: 2D Profile Image of the Scanned Object ........................................... 26
Figure 4.1: Intrinsic Camera Parameters ............................................................... 28
Figure 4.2: Image Plane ......................................................................................... 29
Figure 4.3: 3D Rigid Transformation and the Extrinsic Parameters ................. 31
Figure 4.4: Camera Calibration ............................................................................. 33
Figure 4.5: Normal Vector of the Checkerboard Plane ...................................... 34
Figure 5.1: Triangulation ....................................................................................... 37
Figure 6.1: Detection Process ................................................................................ 40
Figure 6.2: Laser Stripe on Image and Its RGB Components on the Indicated White Line 41
Figure 6.3: Laser Stripe Detection Process .......................................................... 41
Figure 7.1: Flowchart for System Calibration ...................................................... 43
Figure 7.2: Plane Calibration ................................................................................. 44
Figure 7.3: Calibration Process ............................................................................. 45
Figure 7.4: Reconstructed Calibration Object .................................................... 46
Figure 7.5: Laser Projection Plane Calibration .................................................... 47
Figure 7.6: Illuminant Stripe on 2D Image and Extracted Location of the Stripe ..........48
Figure 7.7: Coordinate Vector of the Plane in B-dual-space Geometry ..................49
Figure 8.1: Object and Image of Stripe Pattern ...............................................52
Figure 8.2: Extracted Stripe Pattern on the Surface of the Object .......................52
Figure 8.3: Reconstructed Location of the Stripe in 3D ..................................53
Figure 8.4: Scanned Images ............................................................................54
Figure 8.5: Range Image of a Cup with the Platform .........................................55
Figure 8.6: Range Image of a Cup Without the Platform ....................................56
Figure 8.7: Range Image of a Bar .................................................................57
Figure 8.8: Range Image of a Mouse ..............................................................58
Figure 8.9: Projection of Points ......................................................................59
Figure 8.10: 3D Geometry Model for a Cup and the Platform .........................62
Figure 8.11: 3D Geometry Model for a Cup Without the Platform .....................63
Figure 8.12: 3D Geometry Model for a Styrofoam Bar .....................................64
Figure 8.13: 3D Geometry Model for a Mouse ................................................65
List of Tables

Table 7.1: Intrinsic Parameters of Pictures Shown in Figure 7.4. ............................................. 46
Table 8.1: Structure of the PLY File Format............................................................................. 60
Table 8.2: Example of PLY File .................................................................................................. 61
Chapter 1

Introduction

The 3D geometry of an object has various applications ranging from visualization to making better design decisions. But this information cannot be obtained easily from the usual 2D images: the distance from all the points on the surface to some reference frame have to be calculated. One popular way to get the 3D information from images is using optical triangulation techniques [1]. In this thesis, we propose a low-cost active source based optical triangulation scheme, and show its usefulness with some test objects.

1.1 3D Photography and Range Sensing

3D photography is a technique to create detailed computer graphics models of real world objects. 3D photography measures surface characteristics like 3D geometry and reflectance, which are necessary to construct graphical models. Methods to digitize and reconstruct the shape of 3D objects have grown with the development of lasers, CCD’s and high-speed sampling.

This 3D photography is applicable to science or engineering research, manufacturing, entertainment, automation applications, and medical diagnosis. For instance, for civil and environment engineering applications, this technique can be used to reconstruct of the 3D structure of an old sewage pipe systems, subway tunnels, and building, such that the operator can work in a more friendly environment.
The most important step in 3D photography is range sensing: obtaining the 3D coordinate information of the points on the object. Range sensing is a sequence of processes involving system calibration, scanning, triangulation, and reconstruction of the surface. The general flow for range sensing is given in Figure 1.1. System calibration characterizes the equipment used for range sensing, which is called a range sensor, and triangulation relates a point in the image to the 3D location in space. Then, the 3D geometric information is reconstructed. The result of this process is a range image, which can be further processed to generate a 3D photography with surface properties. This thesis presents a method for range sensing that is accurate and low-cost.

1.2 Related Work

Various methods for 3D scanning have been attempted so far: stereo that mimics the human visual system, motion parallax, occlusion, perspective, shading, and focus are a few
One approach that is directly related to our work is the "3D photography on your desk" project [3]. A shadow is drawn from a general incandescent light source over an object by moving a stick, and the shadows are recorded using a camera. Using range sensing techniques with B-dual space geometry, they reconstructed the 3D model of the test object. This method is very simple and inexpensive, but the passivity of the source compromises the accuracy of the resulting model.

At the other end of range sensing techniques is the "The Digital Michelangelo Project" [12], which uses a commercial 3D scanner from Cyberware Inc. This project aims to construct a virtual museum based on 3D photography. Although their technique is very accurate, it is also very expensive.

1.3 Goals of Thesis

We would like to propose a method with an active source such that the scheme will result in a better accuracy but still maintain low cost. The scheme uses a line laser source and a digital camera. The active source is used to improve the accuracy compared to passive sources. Digital cameras are emerging as inexpensive tools for ubiquitous sensing reducing the cost of the overall system.

The remainder of this thesis is organized as follows. Chapter 2 explains the general process of acquiring the range image, and Chapter 3 covers the range sensing equipment, the range sensor. Then, the detailed steps involved in range sensing are presented from Chapter 4 to Chapter 8. Camera calibration, triangulation, and detection that are introduced in Chapter 4, 5, and 6, respectively, are the main procedures for the following Chapters. In Chapter 7, we show how the overall range sensor is characterized based on these algorithmic blocks. And then, the process of reconstructing the range image from a set of
location information is shown in Chapter 8 with some concrete examples. We conclude this thesis with some remarks and future work in Chapter 9.
Chapter 2

Range Sensing

Range sensing is a process to produce a range image. A range image is a type of digital image whose data at each pixel gives a distance value from a fixed frame [1]. Special sensors, called range sensors, acquire the range images. A device containing these sensors measures the 3D positions of the points on an object and returns a range image. Once the range image is produced, we can construct a 3D geometric model out of it.

2.1 Range Image

There are two types of digital images in computer graphics: intensity image and range image. Intensity images are photographic images encoding the light intensities acquired by cameras. They are the measurement of the amount of light impinging on photosensitive devices. Range images encode shape and distance, in particular, the data tied to a pixel is the distance measurement from a known reference frame. The camera reference frame can be used as a known coordinate system, which is the case for this thesis. From the range image, a 3D geometric model of an object or a scene can be created.

2.2 Range Sensor
There are quite a number of measurement methods to acquire the shape information of an object to make a range image (Figure 2.1). In this thesis, an optical method is considered. There is an extensive reference to various optical methods in Besl [1].

![Shape acquisition diagram]

**Figure 2.1: Taxonomy of the Method to Acquire Shape of Object**

Optical range sensors are classified into active and passive depending on the type of source used (Figure 2.2). An active optical range sensor uses energy projection to an object, or controls some parameters, for instance, focus. A passive range sensor uses intensity images to find depth, for example, stereopsis. Passive methods do not typically yield a dense or highly accurate digitizations required for our application. The method used in this thesis can be classified as active optical range sensing, to be more specific, optical triangulation.
2.3 Optical Triangulation

Optical triangulation is one of the commonly used methods in active optical range sensing (Figure 2.2). The fundamentals of this method are depicted in Figure 2.3. A beam of light from an illuminant strikes the surface. Some of the light bounces toward the sensor through the imaging lens. The position of the reflected part of the object is called a range point, which is determined by finding the intersection of the illuminant’s projected direction $\theta$ and the viewing direction of the sensor $\alpha$. The center of the imaged reflection is triangulated against the laser line of sight [8].

Figure 2.2: Taxonomy of the Optical Range Sensing
The pattern of the light source can be a spot, a circle, a line, or several lines at once. In this thesis, a line stripe from a line laser projector is used. The stripe that is the intersection of the plane of light and the surface of the object is observed through a lens (Figure 2.4). Adjacent profiles of the 3D depth data are acquired by either moving the object, the light source, or the sensor. In this thesis, the object is placed on a linear stage that is moved by equal distance steps.
The sensor used to acquire the digital image consists of three components: a sensor, a frame grabber, and a host computer. For this thesis, a digital camera is used for capturing the illuminated stripe pattern, which works as a sensor and a frame grabber as well. The input to the camera is the incoming light. The light enters the lens and hits the image plane, a charge coupled device (CCD) array. A frame grabber digitizes the intensity of light recorded by the photosensors of the CCD array into a 2D rectangular array. The resulting digital image is a 2D matrix whose entries are called pixels.

The details of the optical triangulation sensor system used in this thesis are given in Chapter 3.

2.4 Optical Triangulation Process
Camera calibration, the first step to range sensing, characterizes the projection transformation, which relates a point in 3D to its corresponding point in a 2D image. But projection is not invertible; 3D location cannot be found directly from 2D point. Triangulation utilizes a known plane as well as the projection to find the 3D location of a point on the known plane. The known plane can be a physical plane or it can be an imaginary plane generated by a light source. These two are the main algorithmic building blocks to get the range image.

System calibration finds the laser plane using camera calibration and triangulation with a known plane. Once the laser plane is identified, triangulation with the laser plane is repeatedly applied to get the 3D locations of the points on an object in a 2D image. The final step is to integrate the 3D locations to construct a range image.
Chapter 3

Range Sensor

A range sensor is an apparatus consisting of both hardware and software and used to generate range images. The hardware consists of a laser projector, a digital camera, a stepper motor, and a linear stage. The software is a window application based on MS.NET technologies, which is used to operate the hardware, perform image processing and present the result of scanning.

3.1 Hardware

The scanning equipment is shown in Figure 3.1. The equipment consists of a laser projector that generates a plane of light, a digital camera hooked to a computer for operation, and a stepper motor that continuously displaces the linear stage.
For high accuracy, an active light source such as a plane laser projector rather than a passive method is used. The CEMER pico laser emmitter projects a 5mWatt bright red laser line across sections of the object. This device emits a plane of monochromatic light from a single source. The laser projection plane makes an illuminated stripe on the surface of the object.

The object to be photographed is placed on a linear stage that is driven by a stepper motor controlled by a main computer. The stepper motor is coupled to the threaded shaft by a special joint on the linear stage. The rotation of the motor is translated to displacement of the platform along the stage. The stepper motor used here is a unipolar NEMA 23KSM motor rated at 7.8 volts and 0.9 amperes. This motor rotates in increments of 1.8° and has a holding torque of 82 oz-in. This stepper motor is interfaced with a controller allowing for accurate positioning and manipulation. The BiStep2A04 stepper controller is used to control the action of the stepper motor. The controller is interfaced to the server by the serial port. The controller draws from the same 7.8 volts power supply to the stepper
motors. Control of the stepper motor can be achieved using a console based application, called SerTest. SerTest is developed for the BiStep2A04 controller by the manufacturer.

A digital camera is placed at a fixed location. The digital camera used here is the Canon S30 PowerShot. The digital camera is also interfaced to the server. The digital camera takes a picture of the illuminated stripe. Next, the object is moved by precise distance and the stripe on the surface is again photographed by the digital camera.

This process continues repeatedly until the laser projection covers the extents of the object. The relative location between the digital camera and the laser has to be fixed during the scanning. We take multiple 2D images of adjacent profiles of the object surface through this scanning process. This results in the input for the image processing and triangulation procedures developed in this thesis.

### 3.2 Software

An integrated software system called ProPAST.NET was developed to operate the hardware and automatically process the captured images. This software is based on the Microsoft.NET framework. The software is used to synchronize the camera and the motor operation using the BiStep2A04 controller. This also provides user access to manipulate the scanning process. Figure 3.2 is the flowchart of the ProRAST.NET operation.
In a general sensing case, all sensors are not in the laboratory. When we are scanning some infrastructure with the hardware that has its operating software in it, we have to remotely access the hardware and operate the software in that sensing hardware if we cannot directly access the sensing field. We can operate the sensor remotely with the .NET web service architecture.

ProRAST.NET client applications connect to the lab server hosting Web Services, which includes several Web Methods. Those Web Method handle all client-server communications and hardware control. Remote login is provided to allow users access to
the experiment through the Internet. The user can control the hardware, reset the system and perform a complete scanning by Internet access (Figure 3.3).

![ProRAST.NET Login Window Form to Open Session](image1)

**Figure 3.3**: ProRAST.NET Login Window Form to Open Session.

The lab server keeps track of registered users. ProRAST.NET checks the login sessions to avoid interrupting multiple client access while a scanning is in progress and then checks the experiment hardware to verify it is ready to operate (Figure 3.4).

![ProRAST.NET Window Form for Connection to the Server](image2)

**Figure 3.4**: ProRAST.NET Window Form for Connection to the Server.
Hardware control functions are accessible through the Remote Controls Form shown in Figure 3.5.

![Remote Controls Window Form](image)

**Figure 3.5: Remote Controls Window Form.**

The 'Step & Shoot' frame in the Figure 3.5 allows the user to capture images at specified locations by incrementally moving the object. A slider bar represents the location of the platform of the linear stage. Captured images are loaded into the temporary folder with a sequential naming system. The Reset Motor command allows a user to adjust the platform location in case a connection failure occurred during the last session. The 'Batch Mode' frame provides controls that start an entire scanning process. The user can specify the number of scanning steps and the spacing interval between two consecutive scanning.

SerTest is a console application for the BiStep2A04 controller and receives all data in the form in Figure 3.5.

After each image is captured, simple image processing to find the laser stripe is performed. Then each 2D profile image is updated and shown in the ProRAST display. At this stage, we can see the 2D profile of the scanning result (Figure 3.6).
Figure 3.6: 2D Profile Image of the Scanned Object.
Chapter 4

Camera Calibration

In order to reconstruct a 3D location from a point on a 2D plane, we have to find the mapping between the two spaces. One building block that will be repeatedly used in this process is determining the projection transformation from a 3D space to the 2D plane. The transformation depends on the characteristics of the camera as well as what global coordinate we choose. The other building block, which relates a point on the 2D plane to a 3D space, is covered in the next Chapter.

From now on, we try to use the following terminology consistently. Image plane is the 2D plane where the image lies and pixel coordinate is the coordinate system chosen for it. Camera reference frame is a coordinate system for the 3D space and world frame is another 3D coordinate system that we know. All the coordinate systems are assumed to follow the right-hand rule.

The projection transformation from a 3D location in the world reference frame to a 2D image plane is described by camera parameters. The camera calibration is a process to find the camera parameters. The relationship between the 2D image and 3D location is derived by the camera calibration process.

4.1 Projection and Intrinsic Parameters

Suppose the camera reference frame has its origin at $O_c$ with three axes $x_c$, $y_c$, and
$z_c$ where $z_c$ is named as the optical axis (Figure 4.1). And the corresponding pixel coordinate has its origin at $O_p$ and axes $x_p$ and $y_p$ parallel to $x_c$ and $y_c$, respectively. Conventionally, the origin of the image plane is the upper left point. The intersection between the optical axis and the image plane is called the principal point and denoted as $\bar{c}$, and the distance from $O_c$ to the image plane is the focal length $f$.

![Camera frame and Image plane](image)

**Figure 4.1: Intrinsic Camera Parameters**

The intrinsic parameters are the characteristics of a camera that link a point in the camera reference frame to a point in the image plane or pixel coordinates. The intrinsic parameters define the projection transformation from the camera reference frame onto the image plane. The intrinsic parameters are $f$, $\bar{c}$, $k$, $dx$, and $dy$. $k$ is the distortion factor that we will describe shortly, and $dx$ and $dy$ are the dimensions of the pixel in the $x$ and $y$ direction respectively. To describe the projection, a simple pinhole camera model is used first, and then non-ideal effects like distortion are gradually added.

In the Figure 4.2, $\bar{P} = [X \ Y \ Z]^T$ is a point in the camera reference frame and $\bar{p} = [u \ v]^T$ is a point in pixel coordinate system from the projection of $\bar{P}$ onto the image plane.
Let the image frame be the translated coordinate system with origin at \( \bar{c} = [c_x, c_y]' \).

Then the projected point with respect to the image frame is

\[
\overrightarrow{p}_{\text{pinhole}} = \overrightarrow{p} - \overrightarrow{c}.
\]

Then, the projection from the camera reference frame to the image frame through a pinhole camera is given as

\[
\overrightarrow{p}_{\text{pinhole}} = \frac{f}{Z} \begin{bmatrix} X \\ Y \end{bmatrix},
\]

where \( f \) is again focal length which is the distance of the lens focal point to the image sensor. If the dimensions of the pixels in the image sensor in \( x \) and \( y \) directions are \( dx \) and \( dy \) respectively, then \( f_x = f / dx \) and \( f_y = f / dy \).

For the real camera, a more comprehensive camera model is required to get higher accuracy. The more realistic model extends the pinhole camera model with some correction for the systematically distorted image coordinates.

The most commonly used correction is for the radial distortion of a lens, which causes a radial displacement of the image points. As a special case, radial displacement due to symmetric distortion only depends on the radial distance from the principal point \( \bar{c} \), which
can be written as

\[ \overline{p}_{\text{distortion}} = \overline{p}_{\text{pinhole}} (1 + k_1 \| \overline{p}_{\text{pinhole}} \|^2 + k_2 \| \overline{p}_{\text{pinhole}} \|^4 + \ldots) . \]

Based on the literature [14], it is well known that the displacement is strongly dependent on the first term. Therefore, displacement due to symmetric radial distortion can be approximated as

\[ \overline{p}_{\text{distortion}} = \overline{p}_{\text{pinhole}} (1 + k \| \overline{p}_{\text{pinhole}} \|^2) , \]

where \( k \) is again the distortion factor.

Another non-ideal effect is skew where it happens when two axes of the image sensor are not orthogonal and tangential distortion due to the lens. For simplicity we do not consider this effect in the formulation of the camera model used here.

Considering all the effects, projection from \( \overline{P} = [X \ Y \ Z]^T \) in the camera reference frame to \( \overline{p} = [u \ v]^T \) in the image frame is obtained from the parameters \( f_x, f_y, \overline{c} \) and \( k \), if we consider symmetric radial distortion.

### 4.2 3D Rigid Transformation and Extrinsic Parameters

To explain the perspective projection, the camera reference frame is used as a known reference frame. In fact, a coordinate with respect the camera reference frame of a point in 3D is often unknown. Rather a coordinate with respect to some world coordinate to be calculated in most case. Therefore, the location and orientation of the camera reference frame with respect to some known reference frame has to be determined. The extrinsic parameters are a set of geometric parameters that relates a global frame to the camera reference frame by translation and rotation.
The 3D rigid transformation is described in Figure 4.3. An arbitrary point \( \mathbf{P}_w \) in the global frame can be expressed as \( \mathbf{P}_c \) in the camera reference frame by translation and rotation. Using matrix equations,

\[
\mathbf{P}_c = \mathbf{R}\mathbf{P}_w + \mathbf{i},
\]

where \( \mathbf{R} \) is a rotation matrix and \( \mathbf{i} \) is a 3x1 translation vector that describes the relative positions of the origins of the two reference frames.

\( \mathbf{R} \) can be defined by a rotation vector \( \mathbf{\beta} = [\beta_x \quad \beta_y \quad \beta_z]^T \) such that \( \mathbf{R} = e^{\mathbf{\beta} \mathbf{\Lambda}} \),

where \( \mathbf{\beta} \mathbf{\Lambda} \) is a skew-symmetric matrix given as
\[ \bar{\beta} = \begin{bmatrix} 0 & -\beta_z & \beta_y \\ \beta_z & 0 & -\beta_x \\ -\beta_y & \beta_x & 0 \end{bmatrix}. \]

The rotation matrix may also be written in a compact form using Rodrigues' formula, from [10]:

\[
R = I_{3 \times 3} \cos(\theta) + \bar{\beta} \Lambda \frac{\sin(\theta)}{\theta} + (\bar{\beta} \bar{\beta}^T) \frac{1 - \cos(\theta)}{\theta^2},
\]

where \( \theta = \| \bar{\beta} \| = \sqrt{\beta_x^2 + \beta_y^2 + \beta_z^2} \).

The parameters \( R, \bar{t} \) are called the extrinsic parameters, and they are dependant on the location of the calibration object with respect to the camera.

4.3 Camera Calibration Method

Camera calibration is the process to estimate the intrinsic and extrinsic parameters and define the projection transformation. A standard method is to get an image from a 3D calibration object with known dimensions and then find the parameters that minimize the error between the points on the image and the projection from the points on the calibration object in the world frame.

The overall projection \( \Pi \) from \( \bar{P}_w \) in the real world to \( \bar{p} \) in the image plane is

\[
\bar{p} = \Pi_{f,\bar{c},k,\bar{\beta},\bar{I}}(\bar{P}_w),
\]

and the unknowns are focal lengths \( f_x \) and \( f_y \) (2 degrees of freedom (DOF)), principal point coordinates \( c_x \) and \( c_y \) (2 DOF), radial distortion factor \( k \) (1 DOF), the translation
vector $\vec{t}$ (3 DOF), and the rotation vector $\vec{\beta}$ (3 DOF).

The usual way of doing calibration is to use a 3D object; however, it is not easy to fabricate an accurate 3D calibration object. Therefore, we turned to a planar calibration object, for example a checkerboard object as shown in Figure 4.4. One problem with a planar calibration object is that the principal point cannot be recovered directly. However, it turns out that assuming the principal point is at the center of the image is sufficient in most cases [3].

\[ \begin{align*}
\text{3D rigid transformation} & \\
R & \rightarrow \vec{t} \\
& \\
\text{Camera frame} & \rightarrow \text{Image plane} \\
O_c & \rightarrow \vec{p}_i \\
& \rightarrow \vec{P}_i \\
\text{World frame} & \rightarrow \\
O_w & \\
\end{align*} \]

Figure 4.4: Camera Calibration

Let $\vec{p}_i$ be the pixel coordinate of the $i$-th observed point in the image plane and $\vec{P}_i$ be the corresponding point in the world frame. The camera calibration process finds the set of camera parameters that minimize the projection error between the model and $N$ observations $\vec{p}_i$, which can be expressed as
\{ f, \mu, k, \beta, \alpha \} = \text{Argmin} \sum_{i=1}^{N} \left\| \mathbf{p}_i - \Pi_{f,\mu,k,\beta,\alpha}(\mathbf{P}_i) \right\|^2.

This is a non-linear optimization, which can be solved by a standard gradient descent method with iteration and a good initial guess. The initial guess for iterating is from the linear calibration that excludes the camera lens distortion as documented elsewhere in the literature [2,5,11,14].

The location of the plane in the world frame is derived from the camera calibration result. When calibrated, the world frame is on the checkerboard shown in Figure 4.4. \( x_w \) and \( y_w \) axes are on the checkerboard plane and \( z_w \) axis is normal to it. Then the normal vector \( \mathbf{n} \) of the checkerboard plane (Figure 4.5), which describes the plane, is the third column of the rotation matrix. The distance from the camera reference frame is \( \mathbf{n}^T \mathbf{r} \). The normal vector and the distance derived from the extrinsic parameters, as the result of the camera calibration define the checkerboard plane location with respect to the camera reference frame.

Figure 4.5: Normal Vector of the Checkerboard Plane
Chapter 5

Triangulation

Usually, the projection transformation is not invertible since we lose information when we move from 3D to 2D. So finding a 3D location from 2D requires additional information. Triangulation is the procedure used to find the 3D location of a point that lies on a known plane in the camera reference frame, from a corresponding image in the image frame. The known frame can be a physical one, such as the checkerboard, or an imaginary plane created by the line laser. Triangulation is an important operation that is repeatedly applied in the reconstruction process. B-dual-space geometry is used to implement the triangulation [5].

5.1 Plane in B-dual-space Geometry

All definitions are given in Euclidean space as well as in projective geometry [5]. B-dual-space geometry is derived from projective geometry. A plane in space can be represented by a homogeneous 4-vector in a reference frame. If the plane does not contain the origin of the reference frame, it can be represented by a 3-vector.

In Euclidean geometry, let a point \( P \) in the camera reference frame be a vector \( \vec{X} = [X \ Y \ Z]^T \). The point \( P \) also can be represented by the homogeneous 4-vector \( \vec{X} \sim [X \ Y \ Z \ 1]^T \). The \( \sim \) sign means that any scaled vector of \( [X \ Y \ Z \ 1]^T \)
represents the same point in the space.

A plane is defined as the set of points \( P \) with homogeneous coordinate vector \( \overline{X} \) that satisfies:
\[
\langle \overline{\pi}, \overline{X} \rangle = 0,
\]
where \( \overline{\pi} = [\pi_x \, \pi_y \, \pi_z \, \pi_i]^T \). If \( \pi_x^2 + \pi_y^2 + \pi_z^2 = 1 \), then \( [\pi_x \, \pi_y \, \pi_z]^T \) is the normal vector of the plane and \( -\pi_i \) is the distance of the plane from the origin of the camera reference frame.

Analogously, suppose \( \overline{X} = [X \, Y \, Z]^T \) in the camera reference frame is given and we want to define a plane that does not have the origin. The plane is given by
\[
\langle \overline{\omega}, \overline{X} \rangle = 1,
\]
where \( \overline{\omega} \) is the coordinate vector of the plane in B-dual-space geometry.

The relationship between the homogeneous plane vector \( \overline{\pi} \) and the B-dual-space plane vector
\[
\overline{\omega} = -\frac{1}{\pi_i} \begin{bmatrix} \pi_x \\ \pi_y \\ \pi_z \end{bmatrix}.
\]

## 5.2 Triangulation with B-Dual-space Geometry

Triangulation is a procedure to find a 3D location from its projected 2D data and a known plane.

Imagine a plane in space with a coordinate vector \( \overline{\omega} \) (Figure 5.1). Let a point \( P \) on a
known plane correspond to the coordinate vector \( \vec{X} = [X \ Y \ Z]^T \) in the camera reference frame, and let the point \( p \) on the image plane be the projection of the point \( P \) onto the image plane. \( p \) is a vector \( \vec{x} = [x \ y]^T \) in the image frame. Then the corresponding homogeneous coordinate vector for \( p \) is \( \vec{x} \sim [x \ y \ 1]^T \). The triangulation problem is then to find the point \( P \) from its projection \( p \) and the known plane or, alternatively, finding \( \vec{X} \) from \( \vec{x} \) and \( \vec{w} \).

![Known plane](image)

**Figure 5.1: Triangulation**

Since \( P \) lies on the optical ray \((O, p)\) in the camera reference frame, its coordinate vector satisfies \( \vec{X} \sim \vec{x} \), or \( \vec{X} = Z \vec{x} \). Relating this with the face that \( P \) lies on the plane, or \( \langle \vec{w}, \vec{X} \rangle = 1 \), we find

\[
Z = \frac{1}{\langle \vec{w}, \vec{x} \rangle},
\]

and consequently, \( \vec{X} = \frac{\vec{x}}{\langle \vec{w}, \vec{x} \rangle} \).
With the above triangulation method, the 3D location can be found from the pixel coordinate and the coordinate vector of the plane. The pixel coordinate is extracted from the 2D image captured by the digital camera and the coordinate vector of the plane is calculated by the system calibration step.
Chapter 6

Detection

Before finding the location of the projected laser stripe on the object to be used, the location of stripe in the image has to be identified. This chapter explains how to detect the location of the illuminant stripe in a 2D image, where the subject is strongly related to image processing.

6.1 Detection

The laser illuminates the imaged object with a stripe. As the laser light is scattered on the surface of the object, the illuminated area is not clearly distinct but appears as a sort of light intensity distribution corresponding to the approximate location of the laser light reflection.

In order to refine the laser stripe illumination from the image, a simple heuristic algorithm is used. The first assumption is that the object is opaque and not shiny. Furthermore, we assume that the vertical line of the image is not parallel to the laser projection plane. The scanning to find the illuminant stripe on each image is done from left to right along the vertical lines (Figure 6.1).
The points where the laser hits are characterized by the difference between the red component and the green or blue component (Figure 6.2), and a threshold is used to decide which pixel is illuminated by the laser. The image is scanned vertically, line by line. Then the indices of the pixel array, where the differences are greater than the threshold, are averaged along the vertical line to locate the laser projection stripe. The average value is taken to be the location of illumination (Figure 6.3). For a more complete result we would apply an image-processing filter to eliminate noise around the center of the illuminated stripe in image. This operation is beyond the scope of this work. Curless uses a spacetime analysis [8] to set the good result even when the reflectance on the surface of the object varies.
Figure 6.2: Laser Stripe on Image and Its RGB Components on the Indicated White Line

Figure 6.3: Laser Stripe Detection Process
Chapter 7

System Calibration

Before scanning the object, the system needs to be calibrated. System calibration uses projection, detection, and triangulation to identify the characteristics of a range sensor. As a result, the location of the laser projection plane with respect to the camera reference frame is found.

Figure 7.1 shows the overall steps involved in system calibration. The first step, plane calibration, is a process to find the locations of planes with respect to the camera reference frame. Plane calibration performed using the camera calibration algorithm, and the intrinsic camera parameters, the extrinsic parameters, and the location information of the planes are obtained. Each plane in plane calibration has a laser stripe on it. With a known plane location, the illuminant laser stripe on the plane is detected, and the points are triangulated. This gives the location of the points in 3D with respect to the camera reference frame, and the laser plane can be located from these points.
7.1 Plane Calibration

The first step finding the location of the laser projection plane is to find the locations of some planes, which are attached a calibration object. These planes do not need to be perpendicular; not being parallel is enough (Figure 7.2).

For these planes, the camera calibration procedure is executed to find the camera parameters. The intrinsic camera parameters characterize the camera and projection from 3D to 2D. The extrinsic camera parameters for each plane give us the location information of the planes with respect to the camera reference frame (Chapter 4). As the result of the
plane calibration, the location of the planes in the camera reference frame is found.

![Diagram of camera frame, image plane, and laser source.](image)

Figure 7.2: Plane Calibration

For camera calibration, we have used the “Camera Calibration Toolbox for MATLAB” written by Bouguet (http://www.vision.caltech.edu/bouguetj/calib_doc/). Throughout the thesis, no skew is assumed, and the principal points is fixed at the center of the image. Symmetric radial distortion is approximated by the leading term (section 3.1); to use only first distortion factor, the “est_dist” parameter is set to \([1; 0; 0; 0; 0]\).

Figure 7.3 shows the various images of calibration object with different angle. From these images, the intrinsic camera parameters and extrinsic camera parameters for each location of the checkerboard are calculated. The reconstructed locations of the checkerboard from the extrinsic camera parameters for each plane are shown in Figure 7.4. The grid points are the corners of the checkerboard are used for the calibration object. Table 7.1 shows the intrinsic parameters from those images of the calibration object. Then, the locations of planes are found from the rotation matrix and translation vector, as explained in section 4.3.
Figure 7.3: Calibration Process
Figure 7.4: Reconstructed Calibration Object

<table>
<thead>
<tr>
<th>Intrinsic Parameter</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Focal Length</td>
<td>(697.70833, 690.43328)</td>
</tr>
<tr>
<td>Principal Point</td>
<td>(310.29604, 228.14934)</td>
</tr>
<tr>
<td>Distortion Factor</td>
<td>-0.13396</td>
</tr>
</tbody>
</table>

Table 7.1: Intrinsic Parameters of Pictures Shown in Figure 7.4.

7.2 Extraction of the Laser Projection Plane

The next step is to derive the location of the laser projection plane with respect to the
camera reference frame. The location information of this laser projection plane will be used to triangulate the 3D coordinates of points on the surface of a target object.

To locate the laser projection plane, two or more lines that are projected on different planes, are necessary. The plane should be calibrated and have a known location with respect to the camera reference frame. To maintain a fixed location, we have taken pictures of reference planes with a laser stripe in the plane calibration step. The emission of the laser makes line on the plane and this line is the intersection between the laser projection plane and the plane that is calibrated already (Figure 7.5). Figure 7.6 shows the process of identifying the B-dual-space representation of the laser plane. $\omega_v$ and $\omega_h$ are plane vectors corresponding to the vertical and horizontal plane, respectively, and $\lambda_v$ and $\lambda_h$ are line vectors of the lines on the vertical plane and horizontal plane, respectively, due to the laser.

![Figure 7.5: Laser Projection Plane Calibration](image)

The simple image-processing method explained in section 6.1 is applied to extract a stripe from the image, and Figure 7.6 shows an example of an image where the algorithm is applied and the result. Then, points on the stripe are triangulated with the location
information of the checkerboard plane using the technique explained in section 5.2.

Figure 7.6: Illuminant Stripe on 2D Image and Extracted Location of the Stripe

With the B-dual-space geometry, the coordinate vector of the line $\vec{\lambda}$ can be expressed as:

$$\vec{\lambda} = \vec{x}_1 \times \vec{x}_2,$$

where $\vec{x}_1$ and $\vec{x}_2$ are the coordinate vectors of the extracted points [5]. Then the coordinate vector of the line, contains the points with respect to the camera reference frame, can be calculated from the above method. For implementation, not only two points but also, all points extracted from images are used.

The coordinate vector of the laser plane can be derived from at least two lines’ location. The B-dual-space geometry is used to find the plane coordinate vector $\vec{\omega}_p$ that describes the location of the laser projection plane. Figure 7.7 depicts the coordinate vectors of the B-dual-space geometry. The symbols used for vectors in B-dual-space geometry in Figure 7.7 correspond to the symbols in Figure 7.5 without the bar.
The coordinate vector of the laser projection plane is the intersection of the two lines in the B-dual-space geometry and this is calculated as follows [5]:

$$\overline{\omega}_p = \frac{1}{2}(\overline{\omega}_1 + \overline{\omega}_2),$$

where $\overline{\omega}_1 = \overline{\omega}_h + \alpha_h \overline{\omega}_h$ and $\overline{\omega}_2 = \overline{\omega}_v + \alpha_v \overline{\omega}_v$. The $\alpha_1$ and $\alpha_2$ are calculated from the following equation.

$$\begin{bmatrix} \overline{\omega}_h \\ \overline{\omega}_v \end{bmatrix} = \begin{bmatrix} \overline{\omega}_1 \\ \overline{\omega}_2 \end{bmatrix} = \begin{bmatrix} \overline{\omega}_v \\ \overline{\omega}_h \end{bmatrix}.$$

Therefore,

$$\begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} = \left(\overline{\omega}_h - \overline{\omega}_v\right)^T \left[\overline{\omega}_h - \overline{\omega}_v\right]^{-1} \left[\overline{\omega}_h - \overline{\omega}_v\right]^T \left(\overline{\omega}_v - \overline{\omega}_h\right).$$

The location of the laser projection plane with respect to the camera reference frame is
calculated as above.
Chapter 8

Reconstruction of 3D Object

After system calibration is performed, we are ready to construct a range image of a target object. The object is scanned and the points on the object are triangulated using the laser plane that is identified by system calibration in Chapter 7. Then, the location information from each scanning is aligned properly, resulting in a range image. From this range image, a geometric description file can be fabricated to visualize the surface of the object.

8.1 Reconstruction of 3D Location of the Point

An object is scanned and the scene is recorded as a digital image at each time step. The 3D locations of the points on the stripe on the image can be triangulated with known laser projection plane location. The reconstructed 3D coordinates with respect to the camera reference frame are incorporated.

The digital camera in the scanning equipment captures the illuminant stripe on the surface of the object (Figure 8.1). By the simple image-processing method in Chapter 6, the points on the illuminant stripe are extracted as pixel coordinate values (Figure 8.2).

The coordinate vector of the laser projection plane is already known as a result of the system calibration in Chapter 7; therefore, the 3D location of the points can be reconstructed by the triangulation explained in Chapter 5. Figure 8.3 is the set of images of the reconstructed 3D locations of the stripe from different viewpoints after triangulation of the stripe exposed in Figure 8.2.
Figure 8.1: Object and Image of Stripe Pattern

Figure 8.2: Extracted Stripe Pattern on the Surface of the Object
8.2 Construction of Range Image

In the scanning process, different parts of the object are taken into consideration by successive scanning and translation of the linear stage, where the object is placed upon. (Figure 8.4). However, the relative location of the laser projection plane with respect to the camera reference frame is unchanged even though the object moves, so the 3D location of the extracted stripe is always on the same plane with respect to the camera reference frame. Therefore, to construct the range image for the object, it is necessary to combine multiple depth data that came from the multiple scanning steps. In other words, the points from each
scanning and triangulation step need to be translated according to the movement of the linear stage.

![Scanned Images](image)

**Figure 8.4: Scanned Images**

First of all, the movement of the linear stage with respect to the camera reference frame should be quantified. The linear stage with a calibration object is translated with the same amount as the case with an object, and the translation vector for each step is estimated. To get the measurement of the relative transformation of the stage, the calibration object is attached to the stage and the extrinsic parameters have to be calculated for the calibration object at each time step. Then, the difference between two translation vectors shows how much the object is moved between two successive scanning.

After the alignment of all scanning stripes with translation modification, the range image is constructed. The range image is 3D geometry data structured as a set of (x, y, z) triplets. The range image of the object shown in Figure 8.4 is displayed in Figure 8.5 and Figure 8.6. Figure 8.7 and Figure 8.8 show range images from a Styrofoam bar and a mouse, respectively.
Figure 8.5: Range Image of a Cup with the Platform
Figure 8.6: Range Image of a Cup Without the Platform
Figure 8.7: Range Image of a Bar
8.3 Visualization

8.3.1 Tessellation

The range image constructed until section 8.3 is a kind of cloud of data. The range image can be visualized as a 3D geometric computer graphics model.

To visualize the 3D geometric model in graphics, the polygon faces connected the 3D data is fabricated from the cloud of 3D data as shown in Figure 8.5 to Figure 8.8. In this thesis, the “Delaunay.m” module of MATLAB is utilized to tessellate the points of
unstructured form. Given a set of data points, the Delaunay module makes a set of lines connecting each point to its natural neighbors. First, all points are projected on a plane that is perpendicular to a laser projection plane (Figure 8.9). Then, the data from one scan has form of line on the plane. Next, the Delaunay tessellates the 2D projected data and gets the indices for each polygon.

![Diagram](image)

**Figure 8.9: Projection of Points**

### 8.3.2 PLY File Format

To describe the 3D shape we selected the PLY geometric file format. The PLY file format is a simple object description for research with polygon models. It was developed at Stanford University.

The structure of a PLY file is composed of a header and the list of elements (Table 8.1). The header is a series of carriage-return terminated lines of text that describe the remainder of the file. The first entry of the header tells whether the file is binary or ASCII. The
remainder of the header describes each element type, the element’s name, how many such
elements are in the object, and a list of the various properties associated with the element.
The list of elements can be a vertex list that is a set of triples for each vertex, face list that
identifies which vertices make up the face, or other element lists.

<table>
<thead>
<tr>
<th>Header</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertex</td>
</tr>
<tr>
<td>List</td>
</tr>
<tr>
<td>Face List</td>
</tr>
<tr>
<td>(list of other elements)</td>
</tr>
</tbody>
</table>

Table 8.1: Structure of the PLY File Format

Table 8.2 demonstrates the basic components of the PLY file. The contents that are in
the braces are not the file format for just explaining. More explanation can be found at
http://www-graphics.stanford.edu/data/3Dscanrep

```plaintext
ply
format ascii 1.0  { ascii/binary, format version number }
comment made by anonymous  { comments keyword specified, like all lines }
comment this file is a cube
element vertex 8  { define "vertex" element, 8 of them in file }
property float32 x  { vertex contains float "x" coordinate }
property float32 y  { y coordinate is also a vertex property }
property float32 z  { z coordinate, too }
element face 6  { there are 6 "face" elements in the file }
property list uint8 int32 vertex_index  { "vertex_indices" is a list of ints }
end_header  { delimits the end of the header }
0 0 0  { start of vertex list }
```
A program called "PLYview" is used to view this file format and visualize 3D geometric model. The "PLYview" program is a software for PLY file format and developed and distributed by Cyberware Inc.(http://www.cyberware.com/). Figure 8.10 to Figure 8.13 show some reconstructed 3D geometric models viewed by PLYview.

Table 8.2: Example of PLY File

```
0 0 1
0 1 1
0 1 0
1 0 0
1 0 1
1 1 1
1 1 0
4 0 1 2 3
4 7 6 5 4
4 0 4 5 1
4 1 5 6 2
4 2 6 7 3
4 3 7 4 0
{ start of face list }
```

61
Figure 8.10: 3D Geometry Model for a Cup and the Platform
Figure 8.11: 3D Geometry Model for a Cup Without the Platform
Figure 8.12: 3D Geometry Model for a Styrofoam Bar
Figure 8.13: 3D Geometry Model for a Mouse
Chapter 9

Conclusions

In this thesis, we have applied the range sensing techniques and visualization methods to form a range image and a 3D geometry model from an object.

We have described and developed a range sensing system, a system for digitizing the 3D geometry of an object. The range sensor uses an active laser light source to project a line pattern on an object and a linear stage with a stepper motor to translate the object. A digital camera is used to capture the projected laser stripe on the surface of the object. The whole range sensing system is controlled by the ProRAST.NET window application.

The overall process of range sensing consists of system calibration and range image reconstruction, which are built on camera calibration, detection, and triangulation. A test object is scanned and the data is triangulated based on the range sensor characteristics from system calibration. Once triangulation returns 3D coordinates of points on the surface of the object with respect to the camera reference frame, the points are integrated to constructed a range image. Then the points in the range image are connected with polygons to make 3D surfaces of the geometry model. The overall process seems to be quite stable, and some interesting results are shown in Chapter 8.

For future work, more accurate image-processing method may be proposed to detect the illuminant location of the laser stripe in each image. We have found that faster, more reliable, and more accurate detection algorithm is crucial for acquiring a better range image, although it is not the main objective of this thesis. For instance, a spatial-temporal coherence concept similar to MPEG encoding can be used to detect the location of the laser strip, which takes the difference of successive images and localize the search zone where
the stripe is expected to be. This may reduce the computation time to detect the laser stripe from images, which may make the whole process real time.

More reliable polygon generation or tessellation technique to generate 3D geometry model from the range image should be developed. This is because the visualization result depends on the quality of the polygon faces extracted from the cloud of unstructured points. Also, we need a method to integrate range images from different point of view in order to get a 360-degree model of an object. For example, combining both views from the top of an object and from the bottom of it will give more complete 3D geometric information of the object.
Appendix A

MATLAB Source Code List

A.1 Important Script

SCRIPT_INCVEC Script for calculating increment of stage translation.
SCRIPT_RECONSTRUCTION Script for object reconstruction.
SCRIPT_SAVEAS_PLY_FILE Script to convert range image to PLY.
SCRIPT.SYSTEM_CALIBRATION Script for system calibration.

A.2 Basic Tools

DISPLAY_IMAGE Read and display the 640*480 image.
DRAW_IMAGE_WITH_SIZE Display the image with real size.
DRAW_IN_CAMERA_IMAGE Display the points in the camera reference frame.
EXTRACT_PARAMETER Extract camera parameters from "param" variable.
EXTRACT_STRIPE Extract stripe from image.

A.3 System Calibration
LAMBDALASERStripe Find lambda vector of the laser stripe.

LAMBDAVECTOR Homogeneous coordinate vector of the line.

OMEGA_LASER_PROJECTION Omega vector of the laser projection plane.

OMEGA_LASER_PROJECTION2 Omega vector of the multiple laser projection planes.

OMEGA_VECTOR Homogeneous coordinate vector of the plane.

### A.4 Triangulation

IMAGE_TRIANGULATION Triangulate from an image.

TRIANGULATION Triangulation of the point.

### A.5 Reconstruction

MAKE_FACE Make faces from the triangulated points.

PROJECT_PERPEND_PLANE Project points to the plane perpendicular to project plane.

REMOVE_FLOOR Remove the floor from the given data points.

### A.6 Motor Control

SCRIPT_STEPPER_MOTOR Script for serial I/O for stepper motor and control
A.7 PLY File

**DISP_PLY_HEADER** Print out the header of PLY format file.

**FIND_SCAN_FORMAT** Find scanf format corresponding to the precision.

**PARSE_HEADER_LINE** Get the first word for each line of header.

**READ_PLY** Read PLY format file (only ascii case).

**READ_PLY_ELEMENT** Read PLY element.

**READ_PLY_FACES** Read PLY faces.

**READ_PLY_HEADER** Read the header of PLY.

**READ_PLY_LIST** Read PLY list.

**READ_VERTICES** Read vertices in PLY.

Structure used PLY modules.

**WRITE_PLY** Write PLY file from the data (vertex and face).

**WRITE_PLY2** Write PLY file from the data (vertex and face).
Appendix B

MATLAB Source Code

B.1 Important Script

SCRIPT_INCVEC Script for calculating increment of stage translation.

% SCRIPT_INCVEC Script for calculating increment of stage translation.
% seh / Aug 11, 2003

% need camera calibration result for the translation iamges
load CalibResults_translation

nTrans = 30;
sumDifTc = [0;0;0];
for i = 1 : ( nTrans - 1 )
    eval( sprintf( 'difTc = Tc_%d - Tc_%d;', i + 1, i ) );
    sumDifTc = sumDifTc + difTc;
end

incVec = sumDifTc ./ nTrans;
incVec = incVec;

save incVec incVec
% result: 3*1 vector, increment of the platform of the linear stage in one step

SCRIPT_RECONSTRUCTION Script for object reconstruction.

% SCRIPT_RECONSTRUCTION Script for object reconstruction.
% seh / Aug 11, 2003

% camera calibration result
load CalibResults
% system calibration result
load omegaLaserVec

% translation of the linear stage information
load incVec

nFirst = 1; % first image number
nEnd = 37; % end image number
x3D = []; % first image number
i = 1;
for j = nFirst : nEnd
    % triangulation
    imageFileName = sprintf( 'Capture_000%d', j );
    eval( sprintf( 'xTri = image_triangulation( ''%s'', ''jpg'', 200, fc, cc, kc, alpha_c, omegaLaserVec );', imageFileName ) );

    % add incMat: considered the linear stage translation
    incMat = [];
    for k = 1 : size( xTri, 2 )
        incMat( :, k ) = incVec;
    end
    xTri = xTri + ( i - 1 ) * incMat;
    disp( sprintf( '%d: %s', i, imageFileName ) )

    % make range data
    x3D = [ x3D, xTri ];
    i = i + 1;
end

save x3D x3D
% result: x3D, 3*(# of point) array
% Each column is the (x,y,z) of a point w.r.t. camera reference frame.

SCRIPT_SAVEAS_PLY_FILE Script to convert range image to PLY.

% SCRIPT_SAVEAS_PLY_FILE Script to convert range image to PLY.
%
% seha / Aug 13, 2003

load x3D

figure
plot3( x3D( 1, : )', x3D( 2, : )', -x3D( 3, : )', '.k', 'MarkerSize', 1 );
SCRIPT_SYSTEM_CALIBRATION Script for system calibration.

% SCRIPT_SYSTEM_CALIBRATION Script for system calibration.
% seha / Aug 11, 2003

% load camera calibration result
load Calib_Results

% threshold for detection
diffThresh = 150 *2; % can be adjusted

for i = 1 : 11
    % read image file
    imageName = sprintf( 'Capture_000%d', 83 + i );
    command = sprintf( 'I_%d = imread( ''%s'' , ''jpg'' );', i, imageName );
    eval( command )

    % rotation matrix from the camera calibration result
    command = sprintf( 'Rc = Rodrigues( omc_%d );', i );
    eval( command )
end
eval( command )

% plane calibration
command = sprintf( 'omega_%d = omega_vector( Rc_%d, Tc_%d );', i, i, i);
eval( command )

% find 3D information of the laser illuminant stripe
command = sprintf( 'lambda_%d = lambda_laserStripe( I_%d, fc, cc, kc, alpha_c, Tc_%d, Rc_%d, diffThresh );', i, i, i, i);
eval( command )
end

% find the laser projection plane from the lines
% multiple command case: take average the plane vector
omega_command = ' [ ';
lambdacommand = ' [ ';
for i = 1 : 11
    omega_command = strcat( omega_command, sprintf( ' omega_%d ', i ));
    lambdacommand = strcat( lambdacommand, sprintf( ' lambda_%d ', i ));
end
omega_command = strcat( omega_command, ' ]' );
lambdacommand = strcat( lambdacommand, ' ]' );

eval( sprintf( 'omega = %s;', omega_command ) );
eval( sprintf( 'lambda = %s;', lambdacommand ) );

omegaLaserVec = omega_laser_projection2( omega, lambda );
save omegaLaserVec omegaLaserVec
% result: omegaLaserVec, 3*1 vector
% Coordinate vector of the laser projection plane.

B.2 Basic Tools

DISPLAY_IMAGE Read and display the 640*480 image.

function [ I, h ] = display_image( imageFileName, imageFileFormat )
% DISPLAY_IMAGE Read and display the 640*480 image.
% [ I, h ] = display_image( imageFileName, imageFileFormat )
% I : image
% h : image object handler
%
% ex.
% imageFileName = '108_0807';
% imageFileFormat = 'jpg';
% h = display_image( imageFileName, imageFileFormat )
%
% seha, June 17, 2003.

[ I, map ] = imread( imageFileName, imageFileFormat );
imageWidth = size( I, 2 );
imageHeight = size( I, 1 );
figure;
h = imagesc( I );
set( gca, 'Unit','pixels' );
currentAxisPosition = get( gca,'Position' );
set( gca, 'Position', [ currentAxisPosition(1),
currentAxisPosition(2), imageWidth, 480 ] );
set( gcf, 'Position', [ 100, 100, ...
   imageWidth + 2 * currentAxisPosition(1) , imageHeight + 2 *
currentAxisPosition(2) ] );


DRAW_IMAGE_WITH_SIZE Display the image with real size.

function [ h ] = draw_image_with_size( I )
% DRAW_IMAGE_WITH_SIZE Display the image with real size.
% [ h ] = draw_image_with_size( I )
% I : image
% h : image handle
%

imageWidth = size( I, 2 );
imageHeight = size( I, 1 );
figure;
h = imagesc( I );
set( gca, 'Unit','pixels' );
currentAxisPosition = get( gca,'Position' );
set( gca, 'Position', [ currentAxisPosition(1),
currentAxisPosition(2), imageWidth, imageHeight ] );
set( gcf, 'Position', [ 10, 10, ...
   imageWidth + 2 * currentAxisPosition(1) , imageHeight + 2 *
currentAxisPosition(2) ] );


DRAW_IN_CAMERA_IMAGE Display the points in the camera reference frame.
function [ h ] = draw_in_camera_frame( Y )
% DRAW_IN_CAMERA_IMAGE Display the points in the camera reference frame.
% draw 3D points with respect to the camera reference frame
% [ h ] = draw_in_camera_frame( Y )
% Y: 3*n array
% h: image handle
%
% seha / June 28, 2003

h = figure;
plot3( Y( 1, : ), Y( 3, : ), Y( 2, : ), '.' );

%maxY = max( max( max( Y ) ) );
%lengthY = Y(1);
axisLength = abs( Y(1) ) * 1;

hold on
hx = line( [ 0 axisLength ], [ 0 0 ], [ 0 0 ] );
set( hx, 'Linewidth', 2 )
set( hx, 'Color', [ 0.5, 0, 0 ] )
tx = text( axisLength, 0, 0, 'X' );
set( tx, 'Color', [ 0.5, 0, 0 ] )
xlabel('X')

hy = line( [ 0 0 ], [ 0 axisLength ], [ 0 0 ] );
set( hy, 'Linewidth', 2 )
set( hy, 'Color', [ 0, 0.5, 0 ] )
ty = text( 0, axisLength, 0, 'Z' );
set( ty, 'Color', [ 0, 0.5, 0 ] )
ylabel('Z')

hz = line( [ 0 0 ], [ 0 0 ], [ 0 axisLength ] );
set( hz, 'Linewidth', 2 )
set( hz, 'Color', [ 0, 0, 0.5 ] )
tz = text( 0, 0, axisLength, 'Y' );
set( tz, 'Color', [ 0, 0, 0.5 ] )
zLabel('Y')

set( gca, 'ZDir', 'reverse' )
set( gca, 'DataAspectRatio', [ 1 1 1 ] )
set( gca, 'XTickLabel', [] )
set( gca, 'YTickLabel', [] )
set( gca, 'ZTickLabel', [] )
grid
\%h = figure
\%plot3( Y( 1, : ), Y( 2, : ), Y( 3, : ), '.' );

\%axisLength = Y(1) * 0.5;

\%hold on
\%hx = line([ 0 axisLength ], [ 0 0 ], [ 0 0 ]);\n\%set( hx, 'Linewidth', 2 );\n\%set( hx, 'Color', [ 0.5, 0, 0 ]);\n\%tx = text( axisLength, 0, 0, 'X' );\n\%set( tx, 'Color', [ 0.5, 0, 0 ]);\n
\%hy = line([ 0 0 ], [ 0 axisLength ], [ 0 0 ]);\n\%set( hy, 'Linewidth', 2 );\n\%set( hy, 'Color', [ 0, 0.5, 0 ]);\n\%ty = text( 0, axisLength, 0, 'Y' );\n\%set( ty, 'Color', [ 0, 0.5, 0 ]);\n
\%hz = line([ 0 0 ], [ 0 0 ], [ 0 axisLength ]);\n\%set( hz, 'Linewidth', 2 );\n\%set( hz, 'Color', [ 0, 0, 0.5 ]);\n\%tz = text( 0, 0, axisLength, 'Z' );\n\%set( tz, 'Color', [ 0, 0, 0.5 ]);\n
\%set( gca, 'DataAspectRatio', [ 1 1 1 ]);\n\%grid;

**EXTRACT_PARAMETER** Extract camera parameters from "param" variable.

function [ fc, cc, alpha_c, kc, Rc, Tc, omc ] =
extractparamter( param )
\% EXTRACT_PARAMETER Extract camera parameters from "param" variable.
\% extract camera parameters from the result of
calibration_iteration.m
\% [ fc, cc, alpha_c, kc, Rc, Tc ] = extractparamter( param )
\%
\% seha / June 25, 2003

fc = param( 1 : 2 );
cc = param( 3 : 4 );
alpha_c = param( 5 );
kc = param( 6 : 10 );
omic = param( 16 : 18 );
Tc = param( 19 : 21 );
Rc = Rodrigues( omc );

**EXTRACT_STRIPE** Extract stripe from image.

```matlab
function [ xStripe ] = extract_stripe( I, diffThresh )
% EXTRACT_STRIPE Extract stripe from image.
% [ xStripe ] = extract_stripe( I )
% I: image
% xStripe: pixel coordinate of the points on stripe
%
% seha / July 5, 2003

laserI = extract_stripe2( I, diffThresh );
[i, j] = find( laserI );
if isempty( i )
    disp('error...no extracted stripe')
end
xPixel = j;
yPixel = -i;
xStripe = [ j', i' ];
```

```matlab
function [ laserI ] = extract_stripe1( I )
% EXTRACT_STRIPE1
% [ laserI ] = extract_stripe1( I )
% I: image
% laserI: image(double data array) which has pixel data of laser
% stripe is 1, o.w. zero
%
% seha / June 26, 2003

r = I( :, :, 1 );
g = I( :, :, 2 );
b = I( :, :, 3 );

rDiff = ( double( r ) - double( g ) ) + ( double( r ) -
double( b ) );

imageWidth = size( I, 2 );

% first method
laserI = zeros( size( r ) );
for i = 1 : imageWidth
    [ maxValue, maxIndex ] = max( rDiff(:, i ) );
    aveMaxI = round( sum( maxIndex ) / length( maxIndex ) );
    laserI( maxIndex, i ) = 1;
end
```
function [ laserI ] = extract_stripe2( I, diffThresh )
% EXTRACT_STRIPEl
% [ laserI ] = extract_stripe1( I )
% I: image
% laserI: image(double data array) which has pixel data of laser
stripe is 1, o.w. zero
%
% seha / June 26, 2003

r = I( :, :, 1 );
g = I( :, :, 2 );
b = I( :, :, 3 );

rDiff = ( double( r ) - double( g ) ) + ( double( r ) -
double( b ) );

imageWidth = size( I, 2 );

rDiffIndex = find( rDiff >= diffThresh );
rDiffFlagMat = zeros( size( r ) );
rDiffFlagMat( rDiffIndex ) = 1;

laserI = zeros( size( r ) );
for i = 1 : imageWidth
    maxIndex = find( rDiffFlagMat( :, i ) == 1 );
    if ~isnan( maxIndex )
        aveMaxI = sum( maxIndex ) / length( maxIndex );
        laserI( round( aveMaxI ), i ) = 1;
    end
end

% DISPLAY OPTION
%figure
%imagesc( laserI );
%colormap( gray )

B.3 System Calibration

LAMBDA_LASER_STRIPE Find lambda vector of the laser stripe.

function [ lambdaVec ] = lambda_laser_stripe( I, fc, cc, kc, alpha_c, Tc, Rc , diffThresh )
% LAMBDA_LASER_STRIPE Find lambda vector of the laser stripe.
% \[ \lambdaVec = \text{lambdalaser\_stripe}(I, \text{fc}, \text{cc}, \text{kc}, \text{alpha}_c, \text{Tc}, \text{Rc}) \]
% I: image
% others: camera parameters
% \lambdaVec: coordinate vector of the stripe
%
% seha / July 5, 2003

% plane vector
omegaVec = \text{omega\_vector}(\text{Rc}, \text{Tc});

% extract the line
xStripe = \text{extract\_stripe}(I, \text{diffThresh});

% normalize some pixel data
xN = \text{normalize}(xStripe, \text{fc}, \text{cc}, \text{kc}, \text{alpha}_c);

% make homogeneous vector
xStripeNormal = [xN; \text{ones}(1, \text{size}(xN,2))];

% triangulation
xStripeTri = \text{triangulation}(xStripeNormal, \text{omegaVec});

% show result
draw\_in\_camera\_frame(xStripeTri);
title('xStripe Reconstructed')

% find the \lambda (line vector)
lambdaVec = \text{lambda\_vector}(xStripeTri);

\textbf{LAMBDA\_VECTOR} Homogeneous coordinate vector of the line.

function [ \lambdaVec ] = \text{lambda\_vector}(xLine)
% LAMBDA\_VECTOR Homogeneous coordinate vector of the line.
% [ \lambdaVec ] = \text{lambda\_vector}(xLine)
% xLine: triangulated 3*n vector in the line
% \lambdaVec: result line vector
%
% seha / June 30, 2003

nPoint = \text{size}(xLine,2);
lambdaVec = \text{zeros}(3,1);
counter = 1;
for i = 1 : nPoint - 2
    for j = i + 1 : nPoint
        temp = \text{cross}(xLine(:,i),xLine(:,j));
        temp = temp./\text{norm}(temp);
        lambdaVec = lambdaVec + temp;
    end
end
crossAll( :, counter ) = temp;
counter = counter + 1;
end
end
lambdaVec = lambdaVec ./ ( counter - 1 );

**OMEGA_LASER_PROJECTION** Omega vector of the laser projection plane.

```matlab
function omegaLaser = omega_laser_projection( omegaH, omegaV, lambdaH, lambdaV )
% OMEGA_LASER_PROJECTION Omega vector of the laser projection plane.
% omegaLaser = omega_laser_projection( omegaH, omegaV, lambdaH, lambdaV )
% omegaH: horizontal plane b-dual-space coordinate vector
% omegaV: vertical plane b-dual-space coordinate vector
% lambdaH: line vector on the horizontal plane
% lambdaV: line vector on the vertical plane
% omegaLaser: b-dual-space vector for the laser projection plane

% seha / June 30, 2003

%A = [ dot( lambdaH, lambdaH ), -1 * dot( lambdaH, lambdaV )]; ...
% -1 * dot( lambdaH, lambdaV ), dot( lambdaV, lambdaV ) ];
A = [ lambdaH, -lambdaV ]' * [ lambdaH, -lambdaV ];

%b = [ dot( lambdaH, omegaV - omegaH )]; ...
% dot( lambdaV, omegaH - omegaV ) ];
b = [ lambdaH, -lambdaV ]' * [ omegaV - omegaH ];

alphaR = inv(A) * b;
alphaH = alphaR( 1 );
alphaV = alphaR( 2 );

omega1 = omegaH + alphaH * lambdaH;
omega2 = omegaV + alphaV * lambdaV;

omegaLaser = 1/2 * ( omega1 + omega2 );
```

**OMEGA_LASER_PROJECTION2** Omega vector of the multiple laser projection planes.

```matlab
function omegaLaser = omega_laser_projection2( omega, lambda )
```
% OMEGA_LASER_PROJECTION2 Omega vector of the multiple laser projection planes.
% omegaLaser = omega_laser_projection( omega, lambda )
% omega: array of plane's b-dual-space coordinate vectors
% = [ omega_1, omega_2, omega_3, omega_4 ];
% lambda: array of line's vector on each plane
% = [lambda_1, lambda_2, lambda_3, lambda_4 ];
% omegaLaser: b-dual-space vector for the laser projection plane
%
%nPics = size( omega, 2 );
omegaLaser = zeros( 3, 1 );
counter = 1;
for i = 1 : nPics - 2
    for j = i + 1 : nPics
        temp = omega_laser_projection( ... 
                                          omega(:, i ), omega(:, j ), ... 
                                          lambda(:, i ), lambda(:, j ) );
        omegaLaser = omegaLaser + temp;
        counter = counter + 1;
    end
end
omegaLaser = omegaLaser ./ ( counter - 1 );

OMEGA VECTOR Homogeneous coordinate vector of the plane.

function [ omegaVec ] = omega_vector( Rc, Tc )
% OMEGA_VECTOR Homogeneous coordinate vector of the plane.
% [ omegaVec ] = omega_vector( Rc, Tc )
% Rc:
% Tc:
% omegaVec:
%
%nVec = Rc(:, 3 );
dis = abs( nVec' * Tc );
piVec = [ nVec; dis ];
omegaVec = - piVec( 1:3, 1 ) ./ piVec( 4, 1 );
B.4 Triangulation

**IMAGE_TRIANGULATION** Triangulate from an image.

```matlab
function xTri = image_triangulation( imageFileName, imageFileType, diffThresh, fc, cc, kc, alpha_c, omegaLaserVec )

% IMAGE_TRIANGULATION Triangulate from an image.
% xTri = image_triangulation( imageFileName, diffThresh, fc, cc, kc, alpha_c, omegaLaserVec )
% imageFileName: name of image file
% imageFileType: type of image file
% diffThresh: threshold for line extraction from the image
% fc: focal length
% cc: principal point
% kc: distortion factor
% alpha_c: skewness of the axis
% omegaLaserVec: laser projection plane homogeneous coordinate vector
%
% seha / Aug 8, 2003

I = imread( imageFileName, imageFileType );

% extract stripe
xStripe = extract_stripe( I, diffThresh );

% normalize some pixel data
xN = normalize( xStripe, fc, cc, kc, alpha_c );

% make homogeneous vector
xNormal = [ xN; ones( 1, size( xN, 2 ) ) ];

% omegaLaserVec <- script_laser_plane_calibration
[ xTri ] = triangulation( xNormal, omegaLaserVec );
```

**TRIANGULATION** Triangulation of the point.

```matlab
function [ xTri ] = triangulation( xNormal, omegaVec )

% TRIANGULATION Triangulation of the point.
% triangulation( xN, omegaVec )
% xNormal: normalized homogeneous vector of pixel data
% omegaVec: plane coordinate vector in d-dual-space geometry
% xTri: the coordinates in the camera reference frame
%```

83
% seh / June 30, 2003

for i = 1 : size( xNormal, 2 )
    xTri( :, i ) = xNormal( :, i ) ./ dot( omegaVec, xNormal( :, i ) );
end

B.5 Reconstruction

MAKE_FACE Make faces from the triangulated points.

function M = make_face( xTris, i )
% MAKE_FACE Make faces from the triangulated points.
% M = make_face( xTris, i )
% xTris: triangulated point coordinates, 3 * n
% i: 2(delaunay) or 3(delaunay3)
% M: index of the face from this points, 4 * #of face
% seh / Aug 8, 2003
x = xTris( 1, : );
y = xTris( 2, : );
z = xTris( 3, : );
if (i == 2)
    M = delaunay( x', y' );
elseif (i == 3)
    M = delaunay3( x', y', z' );
end

PROJECT_PERPEND_PLANE Project points to the plane perpendicular to project plane.

function xPro = project_perpend_plane( omegaLaserVec, xxx )
% PROJECT_PERPEND_PLANE Project points to the plane perpendicular to project plane.
% xPro = project_perpend_plane( omegaLaserVec, xxx )
% omegaLaserVec: laser projection plane vector from the system calibration, 3*1
% xxx: 3D data of the object, 3*N
% xPro : projected points, 3*N
%
% Script for projection before tessellation

% laser projection plane normal vector
laserNormal = omegaLaserVec ./ sqrt( sum( omegaLaserVec.^2 ) );

% x-axis normal vector of the camera reference frame
xx = [ 1; 0; 0; ];

% projection plane normal vector
nVec = cross( laserNormal, xx );

% projection matrix
P = eye( 3 ) - nVec * nVec';
xPro = P * xxx;

**REMOVE_FLOOR** Remove the floor from the given data points.

```matlab
function [pts_mod,n_hat]=removefloor(pts,tol)
% REMOVE_FLOOR Function that removes the floor from the given data points.
% Usage: pts_mod=removefloor(pts,tol)
% Input - pts: 3-by-n matrix of points, where n is the number of them.
% tol: tolerance used to decide whether a noisy point is on the plane.
% Version: 0.1 beta
% Written by Junghoon Lee. =)

% Number of reference points.
num_ref=10;

% Number of points in the data.
um_pts=size(pts,2);

% --- Pick some reference points on the floor. --- %
curr_figure=gcf;
figure(curr_figure), clf
plot(pts(1,:), pts(2,:), '.k', 'MarkerSize',1), hold on
title('Pick 10 points that correspond to the floor:')

ref_pts=zeros(3,num_ref);
for iref=1:num_ref
    [x_in,y_in]=ginput(1);
    % Find the closest point to what is given.
```

```matlab
```
min_dist=inf;
min_index=0;
for ipts=1:num_pts
    dist=sqrt((pts(1,ipts)-x_in)^2+(pts(2,ipts)-y_in)^2);
    if (dist<min_dist)
        min_dist=dist;
        min_index=ipts;
    end
end

% Remember the point.
ref_pts(:,iref)=pts(:,min_index);
plot(ref_pts(1,iref), ref_pts(2,iref), 'ob')
end
hold off

% --- Find the normal vector. --- %

% Solve the least square problem assuming that
% 1) The point that is first picked is on the plane.
% 2) The z component of the normal vector is not zero.
% So, YOU HAVE TO BE A LITTLE LUCKY FOR THIS TO WORK WELL!

% Displacement from the first vector to the others.
dist=ref_pts(:,2:num_ref);
for idist=1:num_ref-1
    dist(:,idist)=dist(:,idist)-ref_pts(:,1);
end

% Solve the least squares problem.
n_hat=dist(1:2,:).'
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '
    '$nhat$[1];
n_hat=n_hat/norm(n_hat);

% --- Construct a new set of points that does not contain the
% floor. --- %
count=1;
pts_mod=[];
for ipts=1:num_pts
    % Compute the height of the plane at the location of the point.
    z=
        n_hat(1)*(pts(1,ipts)-ref_pts(1,1))
        +
        n_hat(2)*(pts(2,ipts)-ref_pts(2,1))
    /(-n_hat(3)+ref_pts(3,1));
% If the difference in the height is sufficient, keep the point.  
if ( abs(pts(3,ipts)-z)>tol )  
    pts_mod(:,count)=pts(:,ipts);  
    count=count+1;  
end  
end

if ( curr_figure ) clf  
plot3(pts(1,:), pts(2,:), pts(3,:), '.k', 'MarkerSize',1), hold on  
plot3(pts_mod(1,:), pts_mod(2,:), pts_mod(3,:), 'ob', 'MarkerSize',2), hold off  
title('Black: original points, Blue: remaining points')

B.6 Motor Control

SCRIPT_STEPPER_MOTOR Script for serial I/O for stepper motor and control movement.

% SCRIPT_STEPPER_MOTOR Script for serial I/O for stepper motor and control movement.

% construct a serial port object for stepper motor  
s = serial('COM2');  
set( s, 'BaudRate', 9600, ...  
    'Parity', 'none', ...  
    'StopBits', 1 );

if isvalid( s ) == 0  
disp( 'stepper motor error: serial port object are not valid.' );  
end

% connect the serial port object to the serial port  
try  
fopen( s );  
catch  
    errorMessage = { 'Connecting Error'; 'Serial port object are not valid.' };  
    msgbox(errorMessage,'Error','error')  
end
% object scanning

% set zero position
fprintf( s, 'x-100000g' )
disp('set ZERO')
pause

% query the device
location = -100000;

while(location<6000)
    serialMessage = sprintf( 'x%dgi', location );
    fprintf( s, serialMessage );

    fprintf( 1, 'STEP %d\n', i );
    disp('start to take pictures!')
    pause(7.02)
    % take picture
    location = location + 2000;
end

disp('plane...')
pause

% linear transformation scanning

location = -100000
fprintf( s, 'x%dg',location );
pause

% query the device
for i = 1 : 30
    serialMessage = sprintf( 'x%dgi', location );
    fprintf( s, serialMessage );

    fprintf( 1, 'STEP %d\n', i );
    disp('start to take pictures!')
    pause(7.1)
    % take picture
    location = location + 2000;
end

% disconnect the serial port object from the serial port
fclose( s );
B.7 PLY File

**DISP_PLY_HEADER** Print out the header of PLY format file.

```matlab
function disp_ply_header(filename)
    % DISP_PLY_HEADER Print out the header of PLY format file.
    % n = disp_ply_header(filename)
    % filename: PLY file name to be displayed
    % n: number of line
    %
    % Seha Kim
    % 03/27/03

    fid = fopen(filename,'r');
    if (fid==-1)
        error('failure open file')
    end

    % check for validation of PLY file
    tline = fgetl(fid);
    if (strcmp(tline,'ply')~=1)
        error('not valid PLY file')
    else
        disp(tline);
    end

    n = 1; % counter
    while (1)
        tline = fgetl(fid);
        disp(tline);

        n = n + 1;
        if (n>100)
            disp('header seems to be too long, so we''re quitting.');
            break;
        end

        if (strcmp(tline,'end_header')==1)
            break;
        end
    end

    fclose(fid);
```

89
**FIND_SCAN_FORMAT** Find scanf format corresponding to the precision.

```matlab
function type_format = find_scan_format(type_name)
% FIND_SCAN_FORMAT Find scanf format corresponding to the precision.
% type_format = find_scan_format(type_name)
%
% seha

switch type_name
    case 'int8'
        type_format = '%d';
    case 'int16'
        type_format = '%d';
    case 'int32'
        type_format = '%d';
    case 'int'
        type_format = '%d';
    case 'uint8'
        type_format = '%u';
    case 'uint16'
        type_format = '%u';
    case 'uint32'
        type_format = '%u';
    case 'float32'
        type_format = '%f';
    case 'float64'
        type_format = '%e';
    case 'float'
        type_format = '%f';
end
```

**PARSE_HEADER LINE** Get the first word for each line of header.

```matlab
function hline = parse_header_line(tline)
% PARSE_HEADER_LINE Get the first word for each line of header.
% parse_header_line(tline)
% tline: a line of header
% hline: cell which each cell has each word
%
% Seha Kim
% 03/27/03

isspace = findstr(tline,' ');
nword = length(isspace) + 1;
istart = 1;
```
for i=1:nword
    if i=nword
        iend = ispace(i)-1;
    else
        iend = length(tline);
    end
    hline(i) = tline(istart:iend);
    istart = iend + 2;
end

**READ_PLY** Read PLY format file (only ascii case).

function [ply,element,list] = readply(filename)
% READPLY Read PLY format file (only ascii case).
% [ply,element,list] = readply(filename)
% filename: PLY file name to open
% ply:
% element:
% list:
% seha

element = [];
list = [];

fid = fopen(filename,'r');
if (fid==-1)
    error('failure open file')
end

ply = readply_header(fid);

if (strcmp(ply.filetype, 'ascii')==1)
    celem = 0;
    clist = 0;

    for i=1:ply.num_elem
        fprintf( 1, 'reading element %d\n', i )
        switch ply.elem(i).type
            case 'element' % general element
                celem = celem + 1;
                element{celem} = readply_element(fid,ply.elem(i));
            case 'list' % list element
                clist = clist + 1;
                list{clist} = readply_list(fid,ply.elem(i));
        end
    end
else
disp('module for binary data is not available...')
end
fclose(fid);

READPLYELEMENT Read PLY element.
function e = readply_element( fid, elem )
% READPLYELEMENT Read PLY element.
% e = read_element(fid,elem)
% fid: file identifier
% elem : element
% e : elements list (elem.size*elem.num_prop)
%
% Seha Kim
% 03/27/03

e = [ ];
for i=1:str2num(elem.size)
    for j=1:elem.num_prop
        scanf_format = find.scan_format(elem.prop{j}.precision);
        e(i,j) = fscanf(fid,scanf_format,1);
    end
end

READPLYFACES Read PLY faces.
function f = readply_faces(fid,nface)
% READPLYFACES Read PLY faces.
% f = read ply_faces(fid,nface)
% nface : number of faces
% f : faces list cell(nface)(nvertex)
%
% Seha Kim
% 03/27/03

f = [ ];
for i=1:nface
    nvertex = fscanf(fid,'%d',1);
    for j=1:nvertex
        f(i)(j) = fscanf(fid,'%d',1);
    end
end
function [ply] = read_ply_header(fid)

% READ_PLY_HEADER Read the header of PLY.
% [ply] = read_ply_header(fid)
% fid : file id
% ply : ply structure
%
% Seha Kim
% 03/27/03

% check for validation of PLY file
if (strcmp(tline, 'ply') ~= 1)
    error('not valid PLY file')
end

ply.num_comment = 0;
ply.num_elem = 0;

while (1)
    tline = fgetl(fid);

    hline = parse_header_line(tline);

    switch hline(1)
    case 'format'
        ply.file_type = hline(2);
        ply.version = hline(3);
    case 'comment'
        ply.num_comment = ply.num_comment + 1;
        comment = [];
        for i=2:length(hline)
            comment = strcat(comment, hline(i), ' ');
        end
        ply.comment{ply.num_comment} = comment;
    case 'element'
        ply.num_elem = ply.num_elem + 1;
        ply.elem{ply.num_elem}.name = hline(2);
        ply.elem{ply.num_elem}.size = hline(3);
        ply.elem{ply.num_elem}.numprop = 0;
    case 'property'
        if (strcmp(hline(2), 'list') == 1)
            ply.elem{ply.num_elem}.type = 'list';

            ply.elem{ply.num_elem}.num_prop = ply.elem{ply.num_elem}.num_prop + 1;
        end
    end
end
nprop = ply.elem(ply.num_elem).num_prop;
ply.elem(ply.num_elem).prop(nprop).list_precision = hline(3);
ply.elem(ply.num_elem).prop(nprop).precision = hline(4);
ply.elem(ply.num_elem).prop(nprop).name = hline(5);
else
  ply.elem(ply.num_elem).type = 'element';
  ply.elem(ply.num_elem).num_prop = ply.elem(ply.num_elem).num_prop + 1;
  nprop = ply.elem(ply.num_elem).num_prop;
  if (find_user_defined_element(hline(2))==1)
    ply.elem(ply.num_elem).prop(nprop).precision = hline(2);
    ply.elem(ply.num_elem).prop(nprop).name = hline(3);
  else
    ply.elem(ply.num_elem).prop(nprop).precision = hline(2);
    ply.elem(ply.num_elem).prop(nprop).name = hline(3);
  end
end

function i = find_user_defined_element(hline)
if strcmp(hline,'int8')...
  +strcmp(hline,'int16')...
  +strcmp(hline,'int32')...
  +strcmp(hline,'uint8')...
  +strcmp(hline,'uint16')...
  +strcmp(hline,'uint32')...
  +strcmp(hline,'float32')...
  +strcmp(hline,'float64')==0
  i=1; % user-defined element
else
  i=0;
end

READ_PLY_LIST Read PLY list.
function l = read_ply_list(fid, elem)
% READ_PLY_LIST Read PLY list.
% l = read_ply_list(fid, elem)
% elem : elem
% l : list cell(nlist)(nindex)

% Seha Kim
% 03/27/03

l = []; for i = 1: str2num(elem.size)
    scanf_format = find_scan_format(elem.prop{1}.list_precision);
    nindex = fscanf(fid, scanf_format, l);
    for j = 1: nindex
        scanf_format = find_scan_format(elem.prop{1}.precision);
        l{i}(j) = fscanf(fid, scanf_format, l);
    end
end

READ_VERTICES Read vertices in PLY.

function v = read_vertices(fid, nvert, nprop)
% READ_VERTICES Read vertices in PLY.
% v = read_vertices(fid, nvert, nprop)
% nvert : number of vertices
% nprop : number of properties
% v : vertices list (nelem*3)

% Seha Kim
% 03/27/03

v = []; for i = 1: nvert
    for j = 1: nprop
        v(i, j) = fscanf(fid, '%f', 1);
    end
end

Structure used PLY modules.
% Structure used PLY modules.
% Seha
ply.file_type % ascii/binary
ply.version
ply.num_comment
ply.comment
ply.num_elem
ply.elem

elem.type % general element/ list element
elem.name
elem.size
elem.num_prop
elem.prop

prop.precision
prop.name
prop.list_precision

WRITE_PLY Write PLY file from the data (vertex and face).

function writeply( fileName, vertex, face )
% WRITEPLY Write PLY file from the data (vertex and face).
% writeply( fileName, vertex, face )
% fileName: file name
% vertex: vertex( nVertex * 3 )
% face: face( nFace * 3 ), triangle faces.
%
% seha / Aug 8, 2003

fid = fopen( fileName, 'w' );

if fid == -1
    error( 'cannot open file.' )
end

% write header
fprintf( fid, 'ply
' );
fprintf( fid, 'format ascii 1.0
' );
fprintf( fid, 'comment generated by seha
' );

nVertex = size( vertex, 1 );
fprintf( fid, 'element vertex %d
', nVertex );
fprintf( fid, 'property float x
' );
fprintf( fid, 'property float y
' );
fprintf( fid, 'property float z
' );

nFace = size( face, 1 );
% write vertex
for i = 1 : nVertex
    fprintf( fid, '%f %f %f\n', vertex( i, 1 ), vertex( i, 2 ), vertex( i, 3 ) );
end

% write face
for i = 1 : nFace
    fprintf( fid, '%d %d %d %d\n', 3, face( i, 1 ), face( i, 2 ), face( i, 3 ) );
end
fclose( fid );

WRITEPLY2 Write PLY file from the data (vertex and face).

function writeply2( fileName, vertex, face )
% WRITEPLY2 Write PLY file from the data (vertex and rectangle face).
% writeply( fileName, vertex, face )
% fileName: file name
% vertex: rectangle vertices (nVertex * 3)
% face: rectangle faces (nFace * 4), rectangle faces.
% seha / Aug 8, 2003
fid = fopen( fileName, 'w' );
if fid == -1
    error( 'cannot open file.' )
end

% write header
fprintf( fid, 'ply\n' );
fprintf( fid, 'format ascii 1.0\n' );
fprintf( fid, 'comment generated by seha\n' );

nVertex = size( vertex, 1 );
fprintf( fid, 'element vertex %d\n', nVertex );
fprintf( fid, 'property float x\n' );
fprintf( fid, 'property float y\n' );
fprintf( fid, 'property float z\n' );

nFace = size( face, 1 );
fprintf( fid, 'element face \%d\n', nFace );
fprintf( fid, 'property list uchar int vertex_indices\n' );
fprintf( fid, 'end_header\n' );

% write vertex
for i = 1 : nVertex
    fprintf( fid, '%f %f %f\n', vertex( i, 1 ), vertex( i, 2 ),
            vertex( i, 3 ) );
end

% write face
for i = 1 : nFace
    fprintf( fid, '%d %d %d %d %d\n', 4, face( i, 1 ), face( i, 2 ),
            face( i, 3 ), face( i, 4 ) );
end

fclose( fid );
Bibliography


