

Fusion of Terrestrial LiDAR Point Clouds with Color Imagery

Senior Project: Colin Axel

Advisors: Dr. Jan van Aardt, Paul Romanczyk, David Kelbe

Committee: Dr. Jan van Aardt, Dr. Carl Salvaggio, Dr. Harvey Rhody

Chester F. Carlson Center for Imaging Science

Rochester Institute of Technology

May 16, 2013

Abstract

Display and representation of real world scenes in three dimensions (3D) is a useful tool for facilitating visualization and scientific measurements. Light detection and ranging (LiDAR) allows scenes to be captured and represented in laser intensity as a set of 3D points (point cloud) with high spatial accuracy. Taking pictures with even a modest camera allows scenes to be displayed as high-resolution two-dimensional (2D) images with accurate color information. Combining the spatial information from 3D LiDAR data with the spectral information from a 2D image provides a true-color, three dimensional replication of a scene. This paper presents a method for fusing a terrestrial (ground-based) LiDAR point cloud with a single color image of the same scene using manually selected control points. No *a priori* scene information or calibration between the two imaging systems is needed. The result is a point cloud for which each discrete 3D sample contains three location coordinates (x, y, z ; distance from laser scanner) as well as three color values (red, green, and blue; digital counts from image).

1 Introduction

Visualization of data is a powerful tool in many applications. Technology has evolved to the point that a three-dimensional scene can be measured and represented as a grayscale set of points through the use of LiDAR. LiDAR is an active remote sensing technique that uses a laser and internal mirrors to scan an area. The time it takes a laser to reach an object, reflect, and return to a detector, is used to find the location or range of the object. There are two types of LiDAR, namely discrete and waveform. Discrete LiDAR stores a point for each surface the laser reflects off, generating a 3D point cloud. Each point in the cloud contains an x, y, z coordinate (made possible through the knowledge of angular scans, or use of a GPS and inertial measurement unit, depending on whether the unit is airborne or ground-based) and an intensity value that represents the strength of the reflection of the laser off a given surface. An airborne LiDAR system is mounted on the bottom of an airplane and a terrestrial system is generally mounted on a tripod. The ability to quickly measure and represent scenes in 3D is extremely useful in many applications that require manual measurements (such as measuring diameter at breast height in a forest). However, it's often necessary to represent a point cloud in two dimensions. One way of doing this is to project the spherical point cloud (colored with the return intensity at each sample) onto a 2D image plane, creating an "intensity image." Figure 1 shows a terrestrial lidar system scanning a scene, the resulting point cloud, and the derived intensity image.

The one drawback of a LiDAR point cloud is the lack of color or even calibrated intensity. Images displayed in color are aesthetically pleasing and enable us to distinguish between objects, as well as separate the foreground from the background. Technology has reached the point where inexpensive, consumer-grade cameras can produce acceptable images and more expensive digital single-lens reflex (DSLR) cameras can generate high-resolution pictures. High-resolution color images are valuable for almost any field of research.



(a) Terrestrial laser scanner



(b) Sample point cloud



(c) Sample intensity image

Figure 1: Using a terrestrial LiDAR system is a fast, accurate way of measuring a 3D scene. (a) A terrestrial LiDAR system with labeled parts (b) a portion of the point cloud generated from the scene shown in (a), and (c) an intensity image created from the point cloud in (b).

For example, a color picture of adequate quality of a forest scene could be used to classify the type of vegetation or for extraction and identification of objects.

An ideal imaging system should be able to capture and combine both spatial and color information. A great deal of publications have shown successful fusion of color imagery with point clouds obtained from airborne LiDAR. Abedinia et al. (2008) successfully superpositioned aerial imagery on an overhead view of a LiDAR point cloud using a combination of image pre-processing, the Scale Invariant Feature Transform (SIFT) (Lowe, 2004) and the Random Sample Consensus (RANSAC) (Fischler and Bolles, 1981) outlier removal algorithms. Wang and You (2010) also performed the same task using region of interest extraction, region growing, and assistant information maps. In both of these publications the LiDAR data and imagery were captured from thousands of feet above ground. When viewing a point cloud from overhead, it appears very similar to an image captured from above, making the registration task much simpler than the ground-based case.

Recent publications have also confirmed the ability to accurately register a terrestrial LiDAR point cloud to a color image through mounted, hybrid LiDAR-camera systems or using complex geometric constraints. Haala et al. (2004) demonstrated the use of a calibrated, mounted system to accurately combine data from a terrestrial LiDAR scanner and a panoramic camera for the investigation of tree species recognition. Similarly, Aguilera et al. (2007) used a calibrated LiDAR-camera system along with epipolar geometry constraints, the Affine-Scale Invariant Feature Transform (A-SIFT) introduced by Morel and Yu (2009), collinearity equations, and RANSAC to fuse point clouds and imagery of urban scenes.

The publications discussed above verify the feasibility of automatically registering a point cloud to an image or set of images. In those investigations the data were either collected from an airborne system or a mounted/calibrated terrestrial system. The objective of this investigation was to assess our ability to register

a point cloud obtained from a discrete, ground-based LiDAR to a single RGB image (image captured with a red, green, and blue color filter array over the detector) without the need for any a priori knowledge or calibration between the two imaging systems. As a result, the registration technique presented in this paper is not fully automatic, but arguably it presents a flexible approach to image-to-lidar registration.

2 Data Collection

2.1 Test Scenes

Two test scenes were chosen for this investigation: the lockers in the basement of the Chester F. Carlson Center for Imaging Science (CIS) and the loading dock located on the south wall of the CIS building, RIT campus. They were chosen because they are readily accessible, contain multiple objects of different colors, and contain several easily identifiable edges. Figure 2 contains images of each scene.

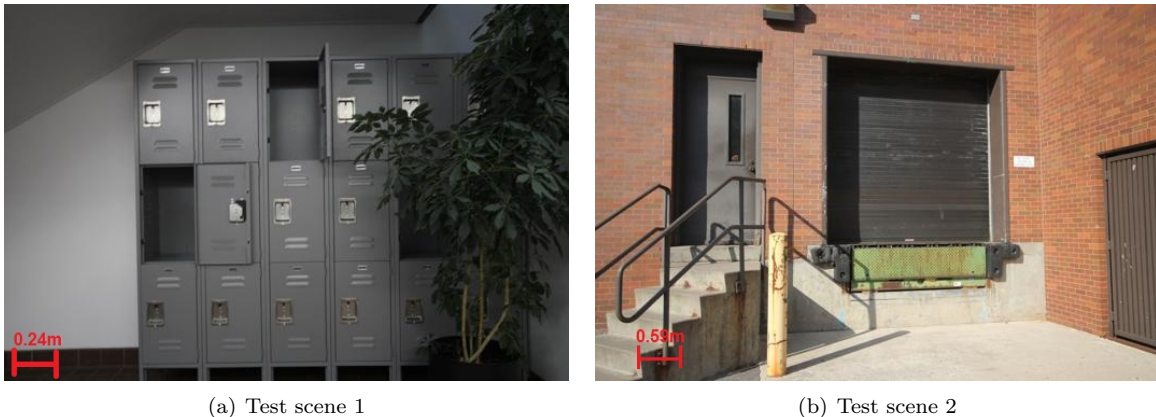


Figure 2: Two scenes were utilized for this investigation. (a) The lockers in the basement of CIS and (b) the loading dock located on the south side of the CIS building.

2.2 Data Collection

The two instruments needed for this investigation were a laser measurement sensor (LMS) for terrestrial LiDAR use and a DSLR camera for image capture. The LiDAR system used was the SICK LMS 151. The camera used was a Canon EOS Rebel T2i with a 10-20 mm lens. At each test location the same data collection procedure was followed. The LiDAR was mounted on the tripod and placed in front of the scene (centered). The scanner was controlled by an iOS application and captured a 360° scan. The tripod was then relocated a few feet to the side. Three or four images were captured with the Canon at approximately the same height, position, and viewing direction as the LiDAR. No measurements or calibration were performed to ensure that capture took place at the exact same location. The specifications of both the SICK LMS 151 and the Canon EOS T2i Rebel are listed in table 1.

Table 1: LiDAR and camera settings, specifications.

SICK LMS 151	Canon EOS T2i Rebel
905 nm laser	18 Megapixel CMOS sensor
0.25° angular resolution	10-20 mm lens
25 Hz scanning frequency	f/3.5 - f/22
0.5 to 50 m range	ISO100 - ISO6400

The data collected from the LiDAR provided a full point cloud, an intensity image, and the elevation and rotation angle of the LiDAR at each discrete sample. The data from the Canon was simply a set of

high-resolution RGB images.

3 Methodology and Results

The workflow starting with the collected data and ending with a colored point cloud consists of several image processing steps. The code for all of these steps was written in Matlab and utilized some of the built in functions found in the Image Processing Toolbox. Certain steps require user input. To demonstrate the entire algorithm, each step will be illustrated with actual examples from the loading dock scene. The actual code can be seen in appendix A.

3.1 Input

The input to the workflow is a single color image and a single intensity image of the same scene. An example of each input is shown in figure 3. The color image is taken directly from the Canon and the intensity image is a product of the LiDAR data processor.

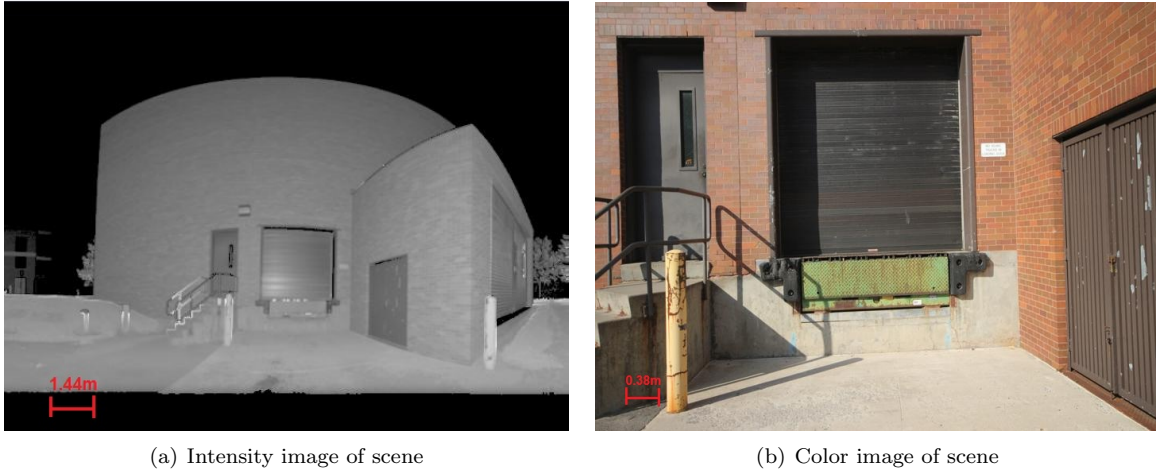


Figure 3: The input to the algorithm is an intensity and a color image of the same scene. (a) The intensity image covers a much broader range than the color image in(b).

3.2 Preprocessing

Figure 3 shows that the intensity image from the LiDAR covers a much larger scene than the color image. A 10-20 mm lens was used to take the rectilinear, color images. Some images were taken with fisheye lenses that cover an area similar in size to the intensity image, but the algorithm has yet to be tested with them. In order to ensure an accurate registration, the user is prompted to crop the intensity image to the rectangular region that is represented in the color image. The user is also prompted to click the location in the intensity image that most closely matches the center of the color image. This location will be used in the next step to locate a new origin. A screenshot of this process is shown in figure 4 and the resulting intensity image can be seen in figure 5.

3.3 Distortion Correction

As mentioned in the introduction, an intensity image is a projection of the spherical point cloud onto a 2D image plane. One consequence of representing a set of 3D points sampled across constant scan lines and angles as a 2D image, is radial distortion. Radial distortion increases as distance from the origin increases. It is assumed that there is no radial distortion present in the color image so the distortion in the LiDAR image must be corrected for an accurate registration. In order to correct for radial distortion, the spherical



Figure 4: The user crops the region of the intensity image that is displayed in the color image in order to eliminate extra data. The user also selects the point in the intensity image that represents the center of the color image. This will force a change of origin in the next step of the algorithm.

coordinates of the point cloud (r, ϕ, θ) are converted to cartesian coordinates (x, y, z) . For this paper, r represents the radial distance to the point from the LiDAR (in millimeters), ϕ is the angle of the rotation stage (in degrees), and θ is the elevation angle of the laser scanner (in degrees).

The coordinates of the center point in the intensity image (that the user selected in the previous step), i_0 & j_0 , are used to force the corrected image to have an origin at the height and direction the camera was facing. Equations 1 and 2 accomplish this task by subtracting 90° from the rotation stage angle, because it starts all the way to the right (0°), but the camera is facing forwards (90°). The elevation should already be at approximately 0° , because the LiDAR scan and picture were taken from roughly the same position.

$$\alpha(i, j) = 90 - \phi(i_0, j_0) \quad (1)$$

$$\beta(i, j) = 0 - \theta(i_0, j_0) \quad (2)$$

Here (and for the rest of the paper) i, j refer to the row and column coordinates of a given pixel. The terms α and β represent the shift of the rotation stage and elevation angles, respectively. The shifted angles are used in the standard spherical to cartesian conversion in equations 3–5.

$$x_{pc}(i, j) = r \cos(\phi(i, j) + \alpha(i, j)) \cos(\theta(i, j) + \beta(i, j)) \quad (3)$$

$$y_{pc}(i, j) = r \cos(\phi(i, j) + \alpha(i, j)) \sin(\theta(i, j) + \beta(i, j)) \quad (4)$$

$$z_{pc}(i, j) = r \sin(\phi(i, j) + \alpha(i, j)) \quad (5)$$

The result of the conversion is a set of x, y, z coordinates for each sample of the point cloud. The x dimension is the horizontal plane in line with the camera (parallel to the image plane), y is in the direction the camera is pointing (perpendicular to image plane), and z is the vertical plane in line with the camera (also parallel to the image plane). Here the subscript, pc , stands for point cloud. The next step is to project the homogeneous point cloud coordinates through a camera matrix (equation 6). A camera matrix is a 3×4 projection matrix that maps a set of 3D coordinates into homogeneous image coordinates using a Euclidean transformation (rotation and translation). The matrix used in this algorithm simply rearranges the coordinates for conversion to inhomogeneous coordinates. The rotation and translation have already been



Figure 5: The cropped intensity image covers approximately the same region as the color image.

performed in equations 1 and 2.

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_{pc} \\ y_{pc} \\ z_{pc} \\ 1 \end{bmatrix} \quad (6)$$

The coordinates at this point are homogeneous. They are converted to inhomogeneous coordinates so that they correspond to the intensity image and so points at infinity can be represented. This is shown in equations 7 and 8.

$$X_{im}(i, j) = \frac{x_{pc}(i, j)}{y_{pc}(i, j)} = \frac{x}{w} \quad (7)$$

$$Y_{im}(i, j) = \frac{z_{pc}(i, j)}{y_{pc}(i, j)} = \frac{y}{w} \quad (8)$$

Here the subscript, *im*, stands for inhomogeneous, image coordinate. With a set of established image coordinates, the distortion-corrected image can be constructed. The transformed bounds of the cropped region are used to determine what coordinates will actually fall in the corrected intensity image. The image is reconstructed one pixel at a time using the actual inhomogeneous coordinates (X_{im} and Y_{im}), a matrix of row-only coordinates (which will be referred to as *rowMat*), and a matrix of column-only coordinates (*colMat*). For each pixel location in the empty corrected image (*correctedIm*), the pixel location whose inhomogeneous coordinates most closely match what the row and column coordinates should be, is chosen. The pixel value from the intensity image (*I*) at that pixel location is then used as the pixel value in the corrected image. The high-level algorithm is displayed below:

```

for  $i = 0 : \text{to number of rows}$  do
  for  $j = 0 : \text{number of columns}$  do
    index = min ( $(\text{rowMat}(i, j) - X_{im}(i, j))^2 + (\text{colMat}(i, j) - Y_{im}(i, j))^2$ );
    correctedIm(i,j) = I(index);
  end
end

```

Algorithm 1: Pseudocode for finding the location of the closest matching inhomogeneous coordinate to the computed row and column coordinates. The chosen location is used as the index of the pixel whose digital count will be used in the corrected image.

The result is a reconstructed intensity image with significantly less radial distortion. An example of a straight edge, before and after distortion correction, can be seen in figure 6.

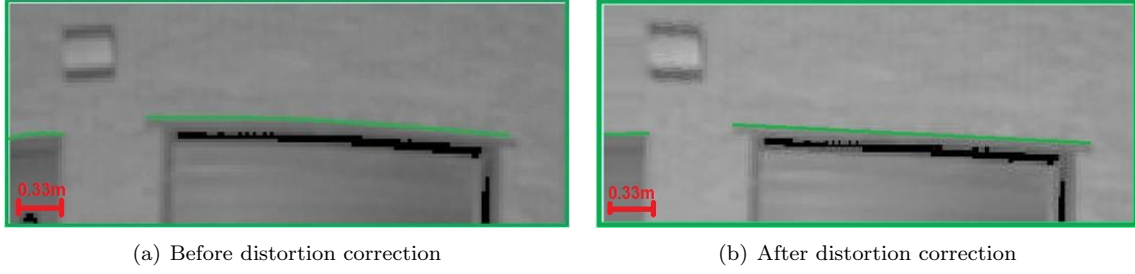


Figure 6: The intensity image suffers from radial distortion and must be corrected. (a) Curved lines are corrected so that they are accurately displayed as (b) straight lines.

3.4 Transform Generation with Control Points

At this point in the algorithm the distortion-corrected and cropped intensity image looks very similar to the color image. In order to build a transformation matrix to register the color image to the intensity image, a set of coordinates of matching pixels must be obtained. This is accomplished by having the user select matching points in both images. Figure 7 shows the interface for control point selection.

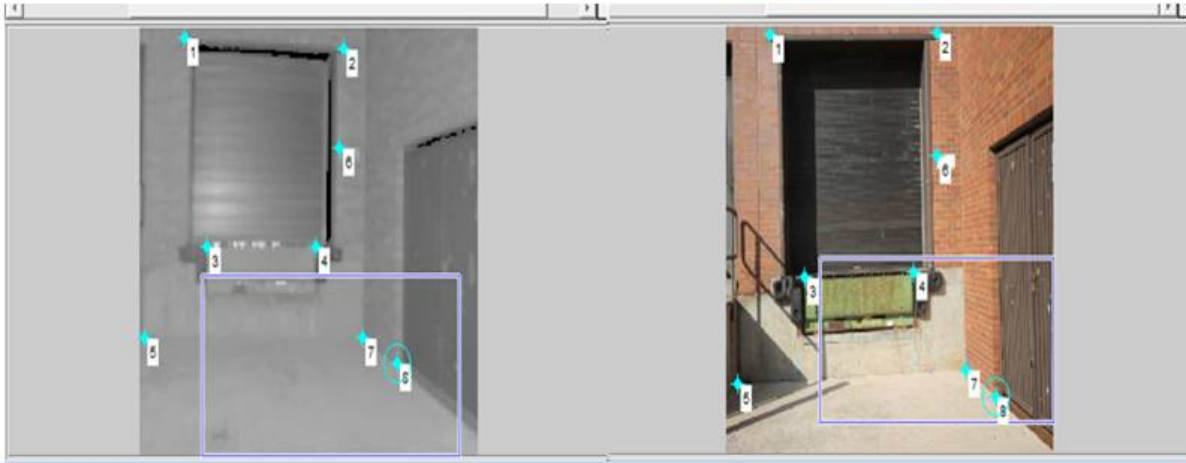


Figure 7: The user manually selects matching points in both images. The best points tend to be corners or unique identifiers, such as the sign to the right of the loading dock.

The set of matching points is used to generate a 3×3 projective transform matrix using Matlab's control points-to-transform function (`cp2tform`). The matrix is applied to the color image in order to make it as spatially similar to the intensity image as possible. The resulting image, shown in figure 8, is scaled to the same size as the intensity image.



(a) Final intensity image

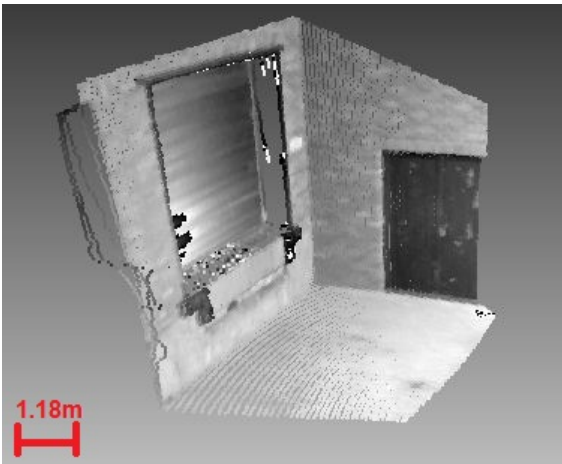


(b) Transformed color image

Figure 8: The 3×3 projective transformation matrix is applied to the color image to align it with the intensity image as closely as possible. The match isn't perfect, but it's very close. There are some areas in the color information with no information due to the transformation.

3.5 Output

With the intensity and color images aligned, the two can be registered and output as a point cloud. Each pixel in the intensity image, for which there is an associated x, y, z point cloud coordinate, is assumed to match to the pixel in the color image at the same location. The red, green, and blue pixel values from the color image are merged with the x, y, z values from the intensity image to create a six element vector for each pixel. These vectors are then written out to a Stanford polygon file (.ply format) so that they can be viewed as point clouds. Figures 9 and 10 show two point clouds for each scene, one colored with intensity values from the LiDAR, and the other colored with values from the color image.



(a) Point cloud using intensity values



(b) Point cloud using color values

Figure 9: A side-by-side comparison of the point cloud colored with histogram equalized intensity values and the same point cloud using the RGB pixel values from the color image.

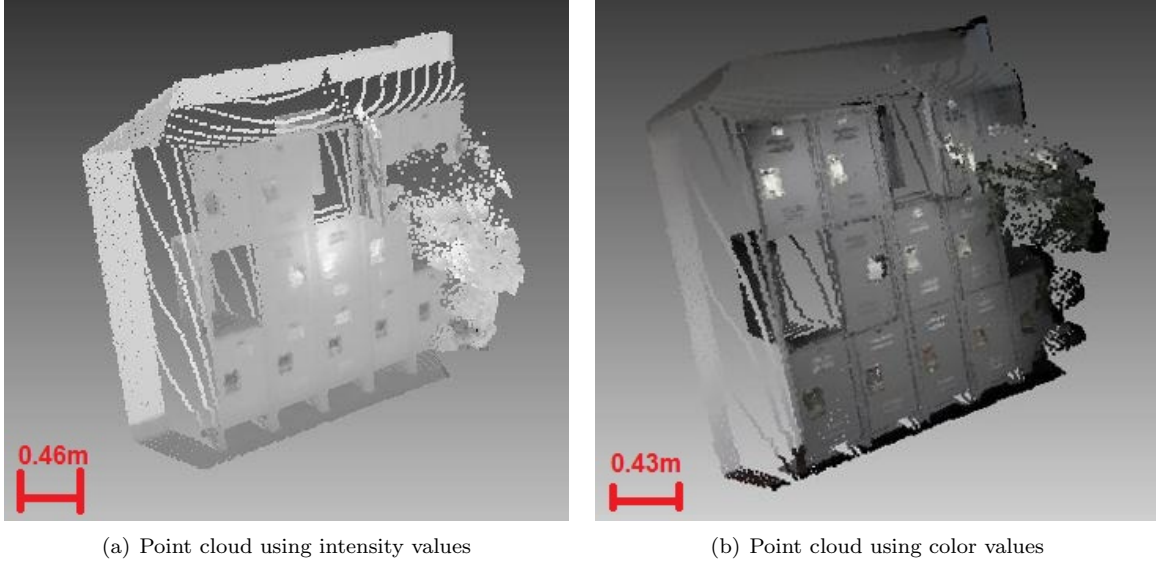


Figure 10: Another side-by-side comparison. This one is slightly harder to see because the lockers are already gray.

4 Discussion

4.1 Results

The results demonstrate that the picture was able to be successfully registered to the point cloud for the two test scenes (Figures 9 and 10). The resulting point clouds are visually appealing and they contain much more contextual information. Unfortunately, a metric hasn't been created to quantify the accuracy of the results yet, leaving visual inspection as the main assessment.

The registrations aren't perfect, as can be seen by the white sign to the right of the loading dock in figure 9(b). When the point clouds are viewed straight on they appear almost identical to the image (Figure 11). Errors (even small ones) along the periphery of the camera's field-of-view are the most noticeable, because even a misregistration of less than one degree is being propagated through the entire depth of the scene. Another issue is the effect of beam divergence. The tree in figure 10(b) looks very choppy and colored gray, rather than green. This is because when the laser beam from the scanner travels through space, it diverges and if a point of reflectance isn't large enough (such as the edge of a leaf), the beam has a chance to hit another object behind it, e.g., the locker behind it.

4.2 Future Work

This investigation reached a point deemed acceptable for the completion of a senior project, but there is still much work that could be done. The original proposal of this project called for a fully automated algorithm that would only require the user to input the point cloud and the image without any other interaction needed to produce a point cloud. The idea was to use SIFT to perform the selection and matching of control points. SIFT is a computer vision algorithm that uses a difference-of-gaussians function applied to a series of sampled and smoothed images in scale space to extract key locations that can be used as control points. The control points can then be matched from one image to another by finding the point with the smallest Euclidean distance from a feature vector. Unfortunately, this algorithm failed to find adequate matches between a LiDAR intensity image and a color image when converted to grayscale (it was tested using the luminance channel as well as the red, green, and blue channels). The main reason for this is most likely the severe difference in pixel values for the same object between images. The laser scanner is an active imaging system, producing it's own 905 nm source of illumination. The camera is a passive imaging system and relies on the illumination from the sun. After several unsuccessful attempts with SIFT, combined with image processing

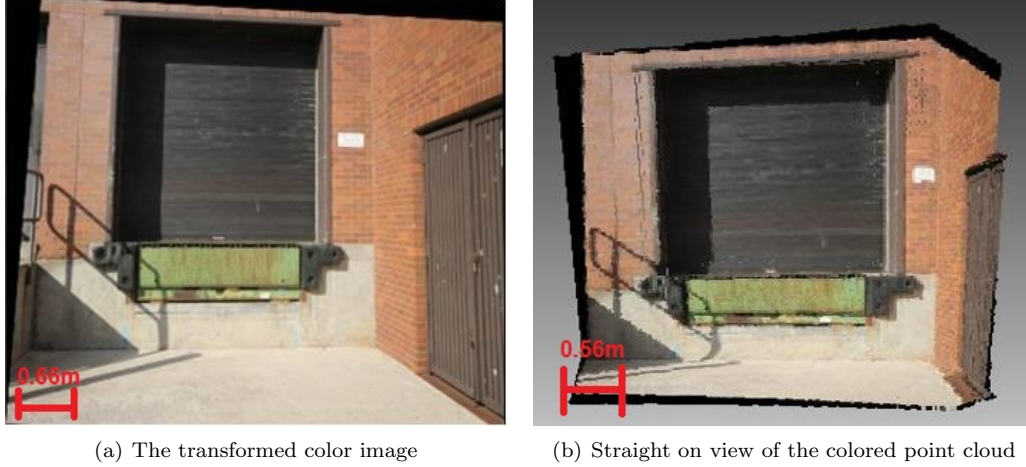


Figure 11: This comparison is to show the visual similarity point cloud and the 2D image when viewed straight on, i.e., how the camera would see the scene.

techniques, it was decided that having the user spend less than a minute to select control points was an acceptable option. One way to overcome this problem in future studies would be to construct a rig of two cameras that could be mounted so that each camera is located on either side of the LiDAR head. The system could be calibrated in the laboratory and then binocular stereo techniques could be used to trace any ray of light in the scene and improve the point cloud-to-image registration.

Another (potential) improvement to this project would be to implement the 3D discrete linear transformation (DLT) Abdel-Aziz and Karara (1971). After the user selects control points in both images, the x, y, z coordinates corresponding to the points selected in the intensity image and the row, column coordinates of the points selected in the color image, would become the input to the DLT to estimate a 3×4 camera matrix. Each point cloud coordinate could then be projected through the camera matrix and the resulting coordinates would be interpolated to find which pixel location in the color image is the closest match. Theoretically, the results should be very similar to those obtained with spatial registration, but I feel this technique is worth testing.

An evaluation metric needs to be created for quantifying the results. By creating a test target with known geometry and placing it in the scene, one could measure the registration error. The algorithm also should be tested on scenes that don't contain many straight lines and that have a variety of colors (unlike the gray lockers).

5 Conclusions

This investigation used matching control points between a LiDAR intensity image and a color image to transform the color image and spatially register it to a 3D point cloud. The result is a true color point cloud that makes contextualization and visualization of a 3D scene much easier. No knowledge of the scene or calibration of the two imaging systems is necessary. The ability to generate colored point clouds could be especially useful for applications such as the measurement and classification of forest scenes. The algorithm presented in this paper requires the user to both crop and select points. In the future, this algorithm could be modified to be fully automatic after a point cloud and image are provided as input.

6 Acknowledgements

I would like to thank Dr. Jan van Aardt, Dr. Carl Salvaggio, Dr. Harvey Rhody, Mr. Paul Romanczyk, Mr. David Kelbe, Dr. Kerry Cawse-Nicholson, Mr. David Nilosek, and Dr. Noah Snavelly for their insights on this project. Without the help of these individuals this project would not have been possible.

7 References

- Abdel-Aziz, Y. I. and Karara, H. M. (1971). Direct linear transformation from comparator coordinates into object space coordinates in close-range photogrammetry. In *Proceedings of the Symposium on Close-Range photogrammetry*, volume 1, pages 1–18.
- Abedinia, A., Hahn, M., and Samadzadegan, F. (2008). An investigation into the registration of LiDAR intensity data and aerial images using the SIFT approach. In *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, volume XXXVII, pages 169–176.
- Aguilera, D. G., González, P. R., and Lahoz, J. G. (2007). Automatic co-registration of terrestrial laser scanner and digital camera for the generation of hybrids models. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 36:162–168.
- Fischler, M. A. and Bolles, R. C. (1981). Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395.
- Haala, N., Reulke, R., Thies, M., and Aschoff, T. (2004). Combination of terrestrial laser scanning with high resolution panoramic images for investigations in forest applications and tree species recognition. *International Society for Photogrammetry and Remote Sensing (ISPRS)*, 1.
- Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision (ICJV)*, 60(2):91–110.
- Morel, J.-M. and Yu, G. (2009). ASIFT: A new framework for fully affine invariant image comparison. *SIAM Journal on Imaging Sciences*, 2(2):438–469.
- Wang, Q. and You, S. (2010). Automatic registration of large-scale multi-sensor datasets. In *11th European Conference on Computer Vision (ECCV)*.

A Code

A.1 pc_rgb_register.m

```
% Name: pc_rgb_register
% Description: process and register a lidar intensity image with rgb image
% Output: a .ply file containing x,y,z,r,g,b information for point cloud
% Required functions: enviread, write2ply
% History: written in fall/winter 2012-2013 by Colin Axel for senior project

%%
close all
clear all
clc
%% setup/loading of files
%location of the point cloud
pc_path = 'C:\My Matlab\senior project\lidar collect\';
pc_file = 'SICK_lockers2_2012-10-04_153152';
angle_file = 'SICK_lockers2_2012-10-04_153152_pos';

%location of the color image
im_path = 'C:\My Matlab\senior project\lidar collect\';
im_file = 'lockers2photo.jpg';

%read in position/angle data from lidar
position_data = enviread([pc_path,pc_file]);
angle_data = enviread([pc_path,angle_file]);

%extract the intensity data
intensity_im = rot90(position_data.z(:,...
                    floor(size(position_data.z,2)/2)+1:end,2),3);

%extract the range data
r = rot90(position_data.z(:,...
                    floor(size(position_data.z,2)/2)+1:end,1),3);

%extract the rotation stage angle
theta = rot90(angle_data.z(:,...
                    floor(size(angle_data.z,2)/2)+1:end,1),3);

%extract the elevation angle
phi = rot90(angle_data.z(:,...
                    floor(size(angle_data.z,2)/2)+1:end,2),3);

%read in the color image
color_image = imread([im_path, im_file]);

%get size (this will be the same for every array)
[rows cols] = size(phi);

%resize the color image to the size of the intensity image
color_image = imresize(color_image, [rows cols]);

%% origin selection and cropping
```

```

%here the color and intensity image are displayed so that the user can
%click the point in the intensity image that most closely matches the
%center of the color image (this will become the new origin)

figure('Name', 'RGB image'), imshow(color_image);
figure('Name','Select point that matches RGB center and then crop'),...
    imshow(intensity_im,[])

%code for the point selection and to get the pixel location
rgb_center = impoint();
rgb_centerPos = round(getPosition(rgb_center));
rgb_center_row = rgb_centerPos(2);
rgb_center_col = rgb_centerPos(1);

%in the same window the user crops the section of the intensity image
%that is contained in the color image
orig_bounds = imrect();
orig_bounds_pos = round(getPosition(orig_bounds));
orig_TL= [orig_bounds_pos(2),orig_bounds_pos(1)]; %top left
orig_TR = [orig_bounds_pos(2),orig_bounds_pos(1)+orig_bounds_pos(3)];%top right
orig_BL = [orig_bounds_pos(2)+orig_bounds_pos(4),orig_bounds_pos(1)]; %bottom left
orig_BR = [orig_bounds_pos(2)+orig_bounds_pos(4),...
    orig_bounds_pos(1)+orig_bounds_pos(3)]; % bottom right
close all;

%% convert to cartesian coordinates

%we have to force the coordinates to the reference frame of the new origin
%the center should already be at approximately 0 degrees of elevation and
%90 degrees of rotation
alpha = double(0 - phi(rgb_center_row,rgb_center_col));
beta = double(90 - theta(rgb_center_row,rgb_center_col));

%compute x,y,z of the point cloud(pc) for later use
x_pc = double(r).* cos(pi/180 * (phi + alpha)) .* cos(pi/180 * (theta + beta));
y_pc = double(r).* cos(pi/180 * (phi + alpha)) .* sin(pi/180 * (theta + beta));
z_pc = double(r).* sin(pi/180 * (phi + alpha));

%equations to convert from spherical to inhomogeneous cartesian
%image coordinates (see manual for math)
imgI_x = cotd(theta+beta);
imgI_y = tand(phi+alpha).*cscd(theta+beta);

%% correct for radial distortion

%compute the transformed bounding coordinates
TL_transformed = [imgI_x(orig_TL(1),orig_TL(2)),imgI_y(orig_TL(1),orig_TL(2))];
TR_transformed = [imgI_x(orig_TR(1),orig_TR(2)),imgI_y(orig_TR(1),orig_TR(2))];
BL_transformed = [imgI_x(orig_BL(1),orig_BL(2)),imgI_y(orig_BL(1),orig_BL(2))];
BR_transformed = [imgI_x(orig_BR(1),orig_BR(2)),imgI_y(orig_BR(1),orig_BR(2))];

%compute the lengths and the ratio of lengths to determine scale factor
orig_length = max(max(orig_BL-orig_TL,orig_TR-orig_BR));

```



```

%calculate the min and max transformed coordinates
minRow = min([TL_transformed(1),TR_transformed(1),BL_transformed(1),BR_transformed(1)]);
maxRow = max([TL_transformed(1),TR_transformed(1),BL_transformed(1),BR_transformed(1)]);
minCol = min([TL_transformed(2),TR_transformed(2),BL_transformed(2),BR_transformed(2)]);
maxCol = max([TL_transformed(2),TR_transformed(2),BL_transformed(2),BR_transformed(2)]);
new_length = max(max(maxRow-minRow,maxCol-minCol));

%use the ratio as the scale factor
scale_factor = round(orig_length/new_length);
%scale_factor = 300;

%scale the extremes
minRow = minRow * scale_factor;
maxRow = maxRow * scale_factor;
minCol = minCol * scale_factor;
maxCol = maxCol * scale_factor;

%scale the image coordinates to pixel size
imgI_x_scaled = imgI_x*scale_factor;
imgI_y_scaled = imgI_y*scale_factor;

%rows, columns for the corrected images
correctedIm_rows = round(maxRow - minRow);
correctedIm_cols = round(maxCol - minCol);

%allocate matrices for the corrected image
correctedIm = zeros(correctedIm_rows,correctedIm_cols,4);
[correctedIm_x,correctedIm_y] = meshgrid(minCol:maxCol,minRow:maxRow);
regError = zeros(correctedIm_rows,correctedIm_cols);

%create a buffer region of 10% around the bounded box just to make
%sure any points cut off by bounding box get included
bufferStartX = round(correctedIm_x(1,1)-correctedIm_x(1,1)/10);
bufferStartY = round(correctedIm_y(1,1)-correctedIm_y(1,1)/10);
bufferEndX = round(correctedIm_x(end,end)-correctedIm_x(end,end)/10);
bufferEndY = round(correctedIm_y(end,end)-correctedIm_y(end,end)/10);

%determine where the buffer regions starts and stops
[~, ind1] = min((bufferStartX - imgI_x_scaled(:)).^2 ...
               + (bufferStartY - imgI_y_scaled(:)).^2);

[~, ind2] = min((bufferEndX - imgI_x_scaled(:)).^2 ...
               + (bufferEndY - imgI_y_scaled(:)).^2);

%search within the bounded area + buffer zone for the closest matching
%value. the last ~50 columns of pixels were all the same along each row
% so i added the +round(rows*correctedIm_cols/10) to make it work.. needs fixing
for i = 1:correctedIm_rows
    for j = 1:correctedIm_cols
        [regError(i,j),ind] = min((correctedIm_x(i,j) - ...
            imgI_x_scaled(ind1:ind2+round(rows*correctedIm_cols/10))).^2 ...
            + (correctedIm_y(i,j) - imgI_y_scaled(ind1:ind2+round(rows*correctedIm_cols/10))).^2);
    end
end

```

```

        %corrected image
        correctedIm(i,j,1) = intensity_im(ind+ind1);
        %corrected coordinates
        correctedIm(i,j,2) = x_pc(ind+ind1);
        correctedIm(i,j,3) = y_pc(ind+ind1);
        correctedIm(i,j,4) = z_pc(ind+ind1);

    end
end
%% crop color image

%make the image a uint8
mx = max(max(max(correctedIm(:,:,1))));
mn = min(min(min(correctedIm(:,:,1))));
corrected_normalized = uint8(255*(correctedIm(:,:,1) - mn)/(mx - mn));

%show the intensity image to make cropping easier
figure, imshow(corrected_normalized);
figure

%crop image to best match the distortion free intensity image
croppedRGB = imcrop(color_image);
croppedRGB = imresize(croppedRGB, [correctedIm_rows correctedIm_cols]);

%% optional cropping of the distorted image for best match
% figure, imshow(croppedRGB);
% figure, imshow(corrected_normalized);
% bounds = imrect();
% boundsPos = round(getPosition(bounds));
% TL= [boundsPos(2),boundsPos(1)]; %top left
% BR = [boundsPos(2)+boundsPos(4),boundsPos(1)+boundsPos(3)]; % bottom right
% correctedIm = correctedIm(TL(1):BR(1),TL(2):BR(2),:);
% corrected_normalized = corrected_normalized(TL(1):BR(1),TL(2):BR(2));

%% point selection, transformation, and registration

%control point selection and transform generation
[input_points,base_points] = cpselect(croppedRGB,corrected_normalized,'Wait',true);
tform = cp2tform(input_points,base_points,'projective');

%registration
registered = imtransform(croppedRGB,tform);

%size changes during transform, resize to make the same size
registered = imresize(registered,[size(corrected_normalized,1)...
                                size(corrected_normalized,2)]);

%% creating the polygon file

%reshape into a Nx3 array
pointCloud = reshape(correctedIm(:,:,2:4),correctedIm_rows*correctedIm_cols,3);

%not sure why but the z axis comes out backwards.. this corrects it
pointCloud(:,3) = pointCloud(:,3)*-1;

```

```
%reshape color into an Nx3 array
color = reshape(registered,correctedIm_rows*correctedIm_cols,3);

%write out to point cloud
isComplete = write2ply('test7.ply',pointCloud,color);
```