SOFTWARE METHOD FOR EMBEDDING SPATIAL DATA INTO A VISUAL IMAGE IN TAGGED IMAGE FILE FORMAT

by

Bhaskar Reddy Duggireddy Vijaya, B.Tech.

THESIS

Presented to the Faculty of

The University of Houston-Clear Lake

In Partial Fulfillment

Of the Requirements

For the Degree

MASTER OF SCIENCE

in Computer Engineering

THE UNIVERSITY OF HOUSTON-CLEAR LAKE

AUGUST, 2018

SOFTWARE METHOD FOR EMBEDDING SPATIAL DATA INTO A VISUAL IMAGE IN TAGGED IMAGE FILE FORMAT

by

Bhaskar Reddy Duggireddy Vijaya

APPROVED BY

_____

Ishaq Unwala, Ph.D., Chair

_____

Thomas L. Harman, Ph.D., Committee Member

_____

Jiang Lu, Ph.D., Committee Member

_____

Carol Fairchild, M.S., Committee Member

APPROVED/RECEIVED BY THE COLLEGE OF SCIENCE AND ENGINEERING:

_____

Said Bettayeb, Ph.D., Associate Dean

_____

Ju H. Kim, Ph.D., Dean

Acknowledgments

I express my sincere and deepest gratitude to my supervisor, Dr. Ishaq Unwala, Associate Professor Department of Computer Engineering. His expertise, invaluable guidance, constant encouragement, affectionate attitude, understanding, and patience added considerably to my experience. Without his continual inspiration, it would not have been possible to complete this study.

I owe my special thanks to Dr. Jiang Lu for being part of my thesis committee.

I take this opportunity to express my deep sense of gratitude and respectful regards to Prof. Carol Fairchild and Dr. Thomas L. Harman for providing the necessary equipment for my work and also for the guidance and encouragement.

Special thanks to Prof. Dr. Hakduran Koc, program chair and associate professor of Computer Engineering.

I extend my thanks to the Dean's Office, Librarians, Writing Centre and UHCL staff. The financial support from the Financial Aid Department in the form of a graduate/research assistantship is gratefully acknowledged

I am also very thankful to my friends Vijay, Anush, Chiranjeev, Shasidar, Travis and others in the Research lab at the UHCL for their assistance.

I thank Pranitha, Siddhu, Hari Krishna, Jitendra, Sindhu, my roommates specially Sunilnath, classmates and my best friends for the support and help.

Above all, I would like to thank my parents and my sisters, brothers who with their love and encouragement have made all this possible.

ABSTRACT

SOFTWARE METHOD FOR EMBEDDING SPATIAL DATA INTO A VISUAL IMAGE IN TAGGED IMAGE FILE FORMAT

Bhaskar Reddy Duggireddy Vijaya
University of Houston-Clear Lake, 2018

Thesis Chair: Dr. Ishaq Unwala

The digital image captured by the camera provides only visual data (illumination and colors) but has no relative size data and therefore is unable to be modeled and printed in 3D. To create a 3D image, the object's distance from the camera and visual data of objects are required. Distance information can be acquired by LIDAR and RGB-D. A digital camera only provides illumination and color information, but the image has a flat plane. i.e. there is no depth information for the object.

To get the full 3D effect, we need to combine both the picture and the distances of the objects from the device. To accomplish the goals of the research, it was necessary to merge the digital visual image with distance data from LIDAR or RGB-D sensor to get more complete information about the object and the 3D shape of an object.

As a proof of concept, a set of software routines are written to implement the algorithm.

TABLE OF CONTENTS

LIST OF FIGURES

Figure Page

CHAPTER 1: INTRODUCTION

Over the last few years, capturing both the digital still and video images have become very popular and cost-effective. Digital cameras are built into every laptop, smartphone, and tablet computer today. Gaming systems and robotics are also major users of the digital images. Gaming systems are merging digital imagery with real-time images to create virtual and augmented reality using VR (virtual reality) goggles. Robotic systems are using the digital imagery to map their surroundings and navigate safely. Digital images are also used in face recognition techniques for security and other uses.

Along with digital images, another area that is gaining prominence is 3D modeling and printing. The digital image provides visual data (illumination and colors), but has no relative size data and therefore is unable to be modeled and printed in 3D. To create a 3D image, the distance of the object from the camera is required. To acquire distance information, a couple of techniques are currently in use: LIDAR and RGB-D. These techniques are discussed in the following section.

This research aims to explore the merging of the digital image and the distance data to create a 3D image storage and modeling system [1].

A large number of optical illusions, intentionally or unintentionally, are possible because the physical distance data is missing from the image. A few examples of these optical illusions that can be resolved by distance data are provided at the end of this section.

LIDAR: LIDAR is an acronym for Light Detection and Ranging. LIDAR is based on the use of a laser beam which is emitted and its reflection collected to calculate the distance. Laser stands for 'Light Amplification by Stimulated Emission of Radiation'. The laser is a device which generates a very narrow, highly concentrated beam of electromagnetic radiation. The laser has many applications, among its many uses are medical applications, barcode scanners, communication, laser printing, welding and cutting, laser nuclear fusion, laser surgery, skin treatments, measuring distances etc.

LIDAR measures the distance of an object. The measurement is done by projecting a laser beam on the target and then analyzing the reflected light. The LIDAR can estimate a distance with very high accuracy and without physically touching or harming the object. The LIDAR can also get intensity values of the reflected light, which can then be used for other analysis. The intensity values are used for matching with a camera image and has applications in agriculture, forest fire detection, river survey, mapping, mining, archeology, and gaming etc.

RGB-D Sensor: The RGB-D is an acronym for Red Green Blue Depth sensor. RGB-D cameras/sensors can be assigned to one of two distinct groups: Structured Light (SL) sensor or Time of Flight (ToF) sensor.

Structured Light (SL) Sensor: The IR (Infra-Red) camera emits an IR light pattern onto a scene and calculates the depth information based on the deformation of the projected pattern. The projected pattern is invisible to the human eye as the emitted light pattern wavelength corresponds to an infra-red color. First generation Kinect sensor

(Microsoft, 2015b) and Asus Xtion PRO Live (ASUSTek Computer Inc., 2015) used the structured light sensor RGB-D camera.

Time of Flight (ToF) Sensor: ToF sensors use a laser to emit a light pulses at the target and measure the distance between the sensor and the target for each point of the image based on the time span of light pulse seen by the detector and the speed of light. The second version of the Kinect sensor is based on ToF sensor. The devices based on ToF sensor have higher accuracy and responsiveness than devices based on structured light [24].

Resolution: The image resolution is described in terms of "pixels per inch" (PPI), which refers to the number of pixels displayed or captured per inch of an image. The higher resolution means that there are more pixels per inch, more pixels results in denser image points and creating a crisper and higher quality image. When scanning or photographing, it is recommended to try and capture the image at the largest possible resolution for the best quality reproduction.

It's better to have more information than not enough! It's much easier for the image editing applications, like Photoshop, to discard any unwanted image information (thus reducing the size of an image) than it is to create new pixel information (enlarge an image).

Here are some optical illusion photos possible due to lack of the distance information. Most of the illusions in these images are self-explanatory.

Figure 1.1 and Figure 1.2 are from https://www.quora.com/What-are-some-great-optical-illusions

Figure 1.1: Optical Illusion of Chair position

Figure 1.2: Optical Illusion of Ant and Helicopter

Figure 1.3 shows the concept of the Ames room, where things appear to be of different sizes even if they are similar. The images are from http://www.zmescience.com/other/feature-post/optical-illusion-ames-room

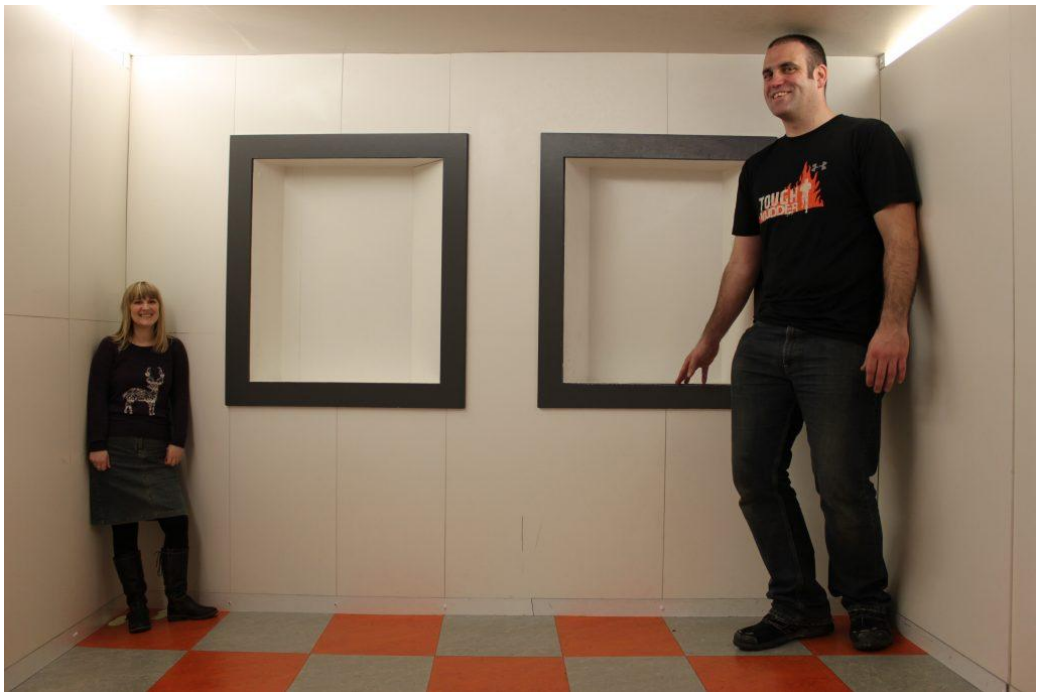

Figure 1.3: Optical Illusion of Ames Room

Due to lack of distance data, mapping images from Google and other mapping suppliers are distorted, giving completely incorrect visual information. Here are a couple of such images (Figure 1.4.1, 1.4.2), you may find hundreds more if you search on the Internet.

Figure 1.4.1 is from http://twentytwowords.com/14-bridges-that-google-earth-fantastically-screwed-up-the-golden-gate-bridge-looks-ridiculous/

Figure 1.4.1: CatSkills

Figure 1.4.2 is from

http://blogs.discovermagazine.com/discoblog/files/2011/04/60_deception-pass-e1302722247650.jpg.


Figure 1.4.2: Deception Pass

Below are some of the many 3D art images (Figure 1.5.1, 1.5.2) that depend on illusion, more such images at

http://designhey.com/amazing-examples-of-3d-street-art

Figure 1.5.1: 3D Street Art

Figure 1.5.2: 3D Street Art

The soccer player wearing a blue jersey in Figure 1.6 is a normal size human being, but he looks short due to a well-known issue called foreshortening.

Figure 1.6: Short Illusion due to foreshortening issue

Below a park in Paris Figure 1.7 (http://www.weirdoptics.com/globe-illusion-paris/) is completely flat but looks like a globe.

Figure 1.7: Globe Illusion Park in Paris

CHAPTER 2: PROBLEM STATEMENT

Normally we use only one type of sensor, either the LIDAR sensor or the RGB-D, to capture the scene. However, each sensor, LIDAR or digital camera, only provides partial information about the scene, either it has the distance information or the visual information in terms of illumination and color.

The LIDAR sensor can only provide a list of 3D points without any visual information about the object, so it is harder to recognize the object shape and color.

Figure 2.1: LIDAR 3D point cloud [1]

The image from a digital camera provides the illumination and color information, but the image has a flat plane. i.e. there is no depth information for the object.

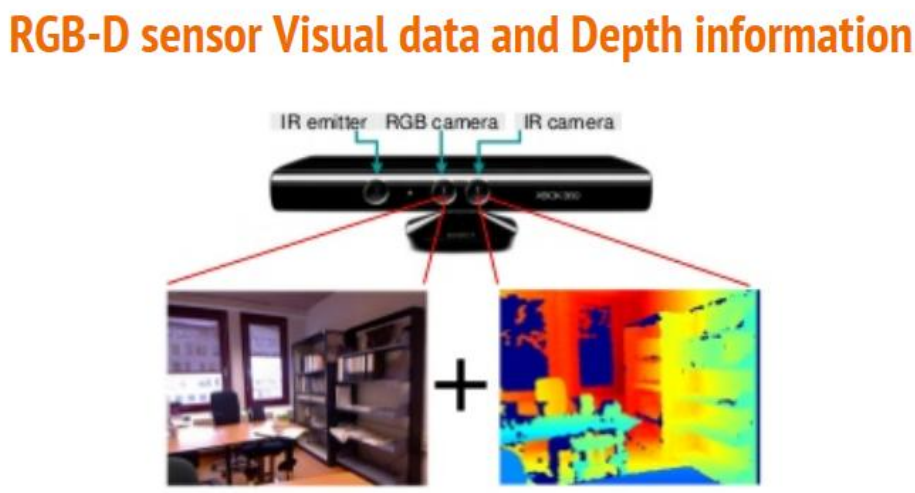

Figure 2.2: RGB-D Visual Data and Depth Information [15]

To get the full 3D effect, both the actual size of objects in the picture and the distances of the object from the device need to be combined and presented to the viewer.

This research develops formats and algorithms to combine these two different datasets into one homogenous data set.

CHAPTER 3: PROPOSED SOLUTION

To accomplish the goals of the research, it is necessary to merge the digital image with distance data from the RGB-D camera to get complete visual and spatial information about the object. There are many ways to merge digital image with distance to obtain a 3D color model. Based on convenience, the speed of merging two types of data, and the volume of the procedural data, we will develop a technique to merge digital image with distance data from Kinect v2 sensor and camera.

Kinect RGB-D sensor is more reliable in indoor scenes and has a range of about meters. The algorithm we create will be able to merge the digital camera image with distance data from Kinect v2 sensor [24].

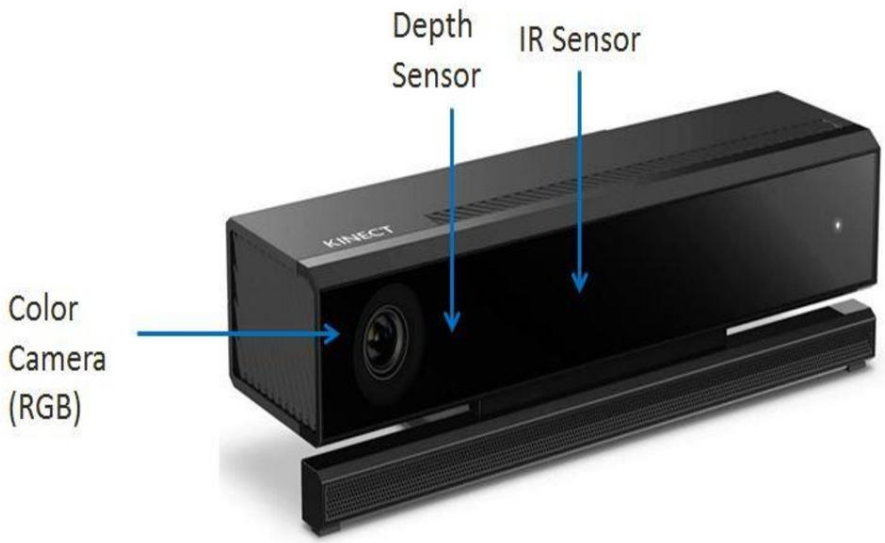

Figure 3.1: Microsoft Kinect v2 sensor

For the depth values, from Kinect v2 sensor, we will check the encoding [31] field. If saved distance data is big_endian or not, is encoded in the field of the header of

/kinect2/sd/image_depth collected values. If encoding is 16UC1, this means 16-bits unsigned short with 1 channel, so each depth is represented with 2 bytes. We can convert these bytes to depth, with consideration of the is_bigendian field.

If the byte order is little-endian (is_bigendian:0), then depth = data[i] + (data[i + 1] << 8). If the byte order is big-endian (is_bigendian:1), then depth = (data[i] << 8) + data[i+1].

Tagged Image File Format (TIFF) is used, as TIFF file uses lossless compression, unlike standard JPEG files which use lossy compression. TIFF file may be edited and saved again without losing image quality. TIFF format is widely accepted versatile raster data format in the world today. TIFF is a suitable format for storage, transfer, display, and printing of raster images such as clipart, logotypes, and scanned documents. The TIFF used to store and transfer digital satellite imagery, scanned aerial photos, elevation models, and scanned maps. To save the TIFF image with new distance tag, we will use an open source application in Ubuntu which supports TIFF image to update the LibTIFF source code.

By writing new code and updating the functions in libraries to set the TIFF tag and save the image with tag, we plan on adding distance data to a tiff image. To have backward compatibility, we will use SXIV tool and Imlib2, LibTIFF libraries.

Figure 3.2: Proposed solution for merging distance and image data

As proofs of concept, a set of software functions will be written in libraries LibTIFF, Imlib2 and also in SXIV application to write Kinect sensor distance data to tiff image.

CHAPTER 4: PREVIOUS WORK

There are many methods to combine distance data of objects in an image. We will see GeoTIFF and some other research papers that inspired us to merge the distance data to the image using Tiff library.

4.1 TIFF

The Tagged Image File Format [2] filename extensions are TIF and TIFF. TIFF is largely used for image-manipulation applications.TIFF specification [3], published by Aldus Corporation/ The TIFF specification describes TIFF as a tag-based file format for storing and interchanging raster images. Raster image represents a generally rectangular grid of pixels or points of color.

TIFF allows for a wide range of different compression schemes and color spaces. TIFF supports multiple images in a single file. TIFF may be edited and re-saved without losing image quality as TIFF uses lossless compression (or no compression), unlike standard JPEG (Joint Photographic Experts Group) files.

The TIFF file basic structure is as follows:

TIFF files are arranged into three sections: the Image File Header (IFH), the Image File Directory (IFD), and the bitmap (image) data. One IFD and one bitmap per image stored if a TIFF file contains multiple images [10].

Figure 4.1: Physical arrangements of data in a TIFF file[10].

Figure 4.1 show some of the possible arrangements of TIFF file sections .The IFH (Header) appears first in each of the TIFF file example in Figure 4.1. In the first example, all of the IFDs are first written to the file followed by the bitmaps (image data). This is the most efficient for reading IFD data quickly. In the second example, each IFD is written, followed by its bitmapped data. This is the common internal format of a multi-image TIFF file. In the last example, the bitmapped data has been written first, followed by the IFDs.

A TIFF file begins with an 8-byte image file header (IFH). The first two bytes are either "II" for little-endian byte ordering or "MM" for big-endian byte ordering. 0 and 42 (2A.H) are the next two bytes representing TIFF version. The remaining 4 bytes are the first "Image File Directory" (IFD) offset.

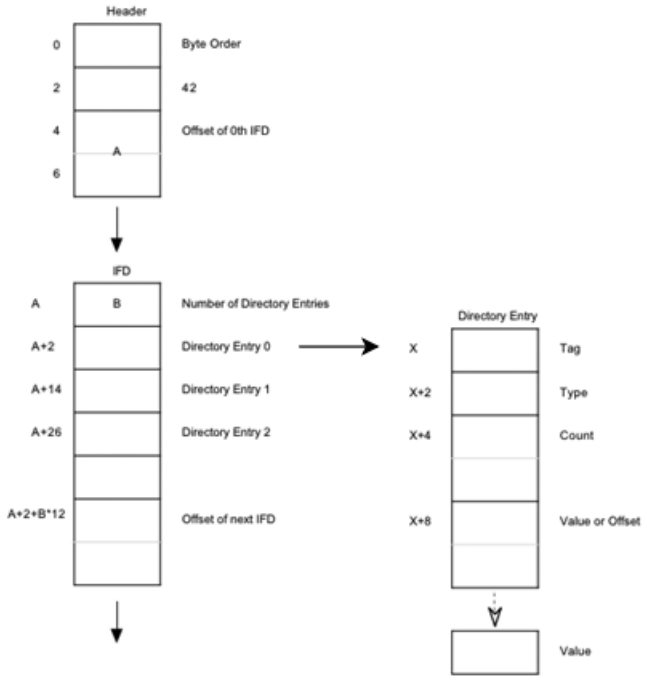

Figure 4.2: Image File Header and IFD in detail [2]

An Image File Directory (IFD) [3] consists of 2-bytes indicating the number of directory/field entries followed by a sequence of 12-byte field entries, and the IFD is terminated with a 4-byte offset of the next IFD or 0 if none. Every TIFF file has at least one IFD.

Each (IFD) directory entry consists of the tag identified in the first two bytes. The field type (byte, ASCII, short int, long int, ...) is the next two bytes. The next four bytes indicate the count. The last four bytes are either the value itself if it fits within 4 bytes or an offset to the values.

The TIFF file has offset values in three locations [10]. The first offset value is in the first IFD position located at the last four bytes of the header. The second offset value

is in the last four bytes of each directory tag, which contains data itself or an offset value to the data. The third offset value is in the last four bytes of the IFD indicating a start of next IFD.

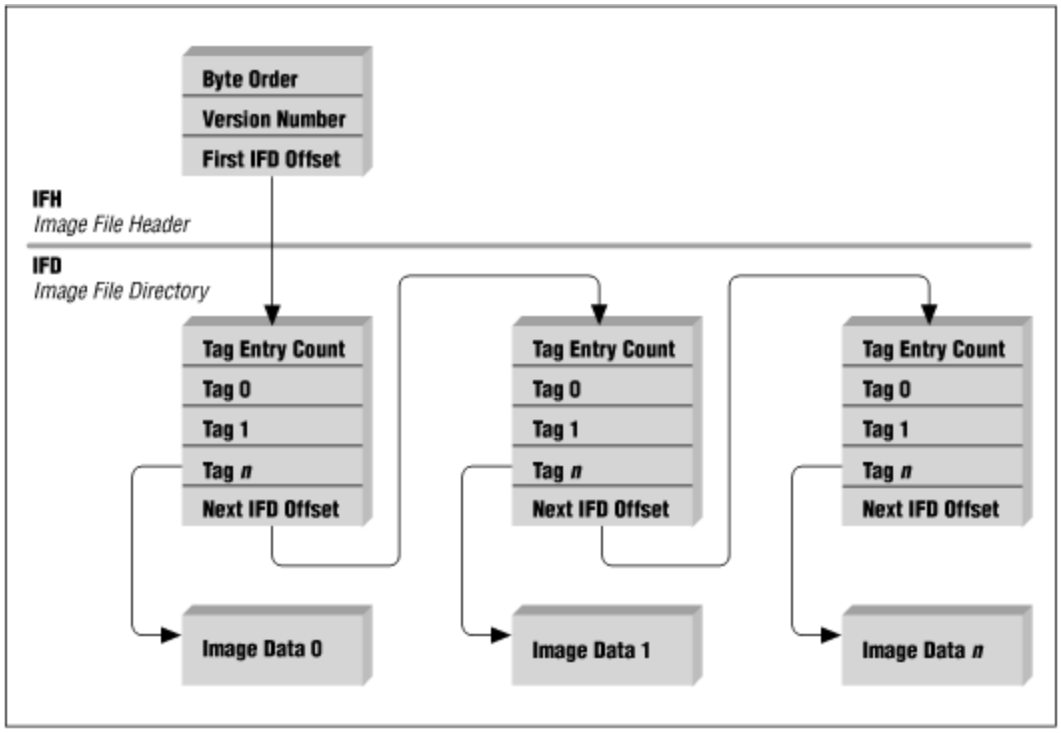

Figure 4.3: Logical organization of TIFF file [10]

TIFF tags sorted by tag number can be found in TIFF-6.0 specification [2] Appendix –A, page 117.

4.2 GeoTIFF

GeoTIFF [5] is the TIFF format with georeferencing information embedded to the TIFF image. Georeferencing means to 'define its existence in physical space'[4]. GeoTIFF establishes the relation between the image and a set of coordinates (map projection, latitude, longitude). The geographic data of a GeoTIFF file is used to position the image in the correct location.

The Keys in GeoTIFF called “GeoKeys”, which are virtually identical in function to a “Tag” in TIFF. All the Geokeys are referenced from the GeoKeyDirectoryTag. So GeoKeyDirectoryTag store the GeoKeys and GeoKey directory header information. A GeoTIFF file begins with GeoKey directory header information.

Header={KeyDirectoryVersion, KeyRevision, MinorRevision, NumberOfKeys}[4]

Where "KeyDirectoryVersion" shows the current version of Key. "KeyRevision" points to what rewriting of Key-Sets is used. "MinorRevision" shows the set of Key-codes is used. "NumberOfKeys" points to how many Keys are defined by the rest of this Tag.

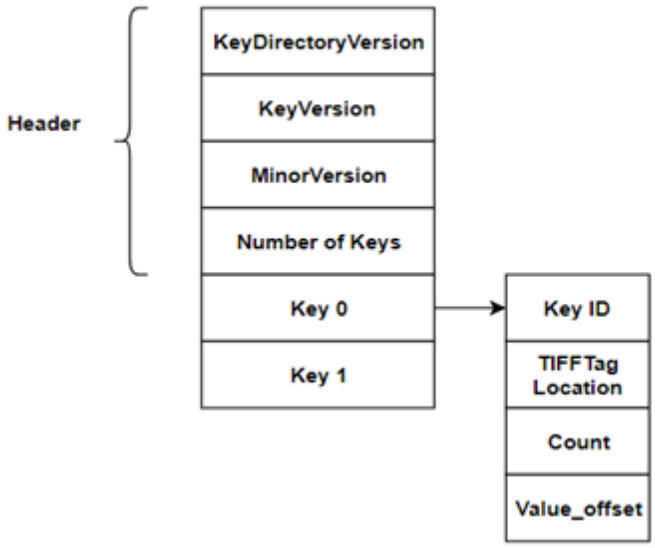

Figure 4.4: GeoTIFF structure

Geokey directory header is followed by each KeyEntry set.

KeyEntry = {KeyID, TIFFTagLocation, Count, Value_Offset }[4]

where “keyID” value is just like TIFF tag ID, but completely independent of TIFF tag-space. “TIFFTagLocation” points to which TIFF tag contains the value(s) of the Key. "Count" shows the number of values in this key. "Value_Offset" points to the index-offset into the TagArray pointed showed by TIFFTagLocation, if it is nonzero. If TIFFTagLocation=0, then Value_Offset contains the actual (SHORT) value of the Key with Count=1.

4.3 Research papers

Juan Li et al.[1] merged 2D Digital camera color image with 3D LIDAR set of data points called point cloud. He used a robotic servo by mounting 2 devices and moving vertically to cover the third dimension as the camera was the entry-level digital camera. He captured the 2D color image from the digital camera and 3D points cloud from LIDAR. He converted the 3D points cloud to 2D point cloud using pinhole camera model. Applied a learning technique to estimate the data points by random sampling technique called Random Sample Consensus (RANSAC) algorithm on checkerboard pattern camera image and LIDAR intensity image to detect corners. Juan Li than calculated the transformation matrices to match image intensity with color images using geometric constraints. Finally, the 2D points with color information were projected back to the 3D points.

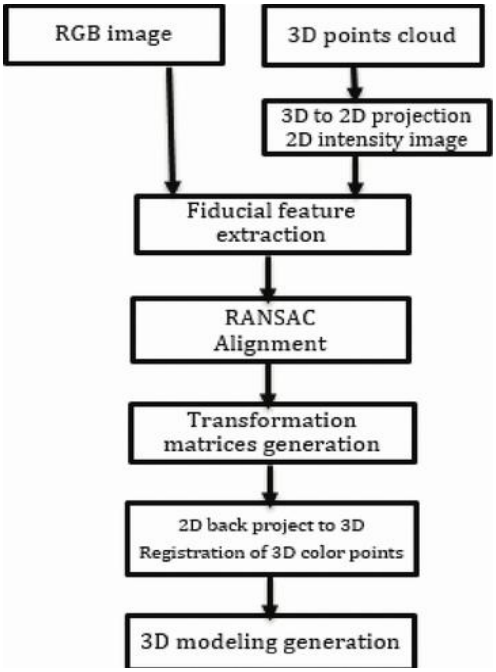

Figure 4.4: Juan Li's [1] Fusion of camera image and LIDAR point cloud

In another paper, Y. Cao et al.[11] showed that object detection accuracy is improved by augmenting RGB images with estimated depth. Depth information plays a major role in image interpretation in a significant amount of work [12], [13], [14]. All algorithms require depth data captured by depth sensors to exploit the information of depth as well as camera images to relate pixels to point clouds.

CNN (convolutional neural networks) based depth estimation method is used as RGB images are interrelated with estimated depth images of CNN models [11].

The author Y. Cao et al. designed the RGB-D detection system [11] based on how pixels within a similar object have similar depth information. The depth sensor and the camera has the same viewing angles in generating RGB-D datasets. So the depth image has depth information except RGB information replaced with intensity values.

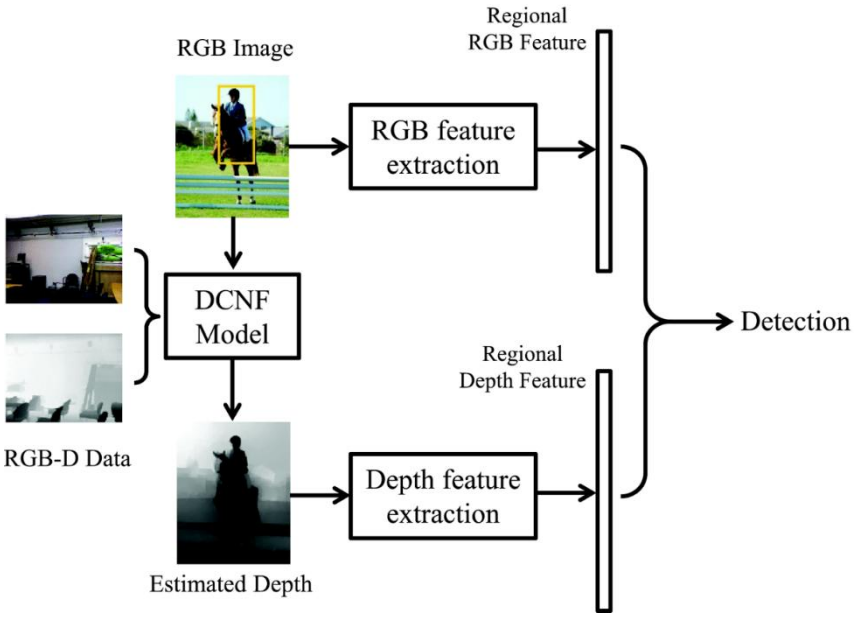

Figure 4.5: An overview of our RGB-D detection system [11].

Depth information is estimated from input RGB-D data using deep conditional neural fields (DCNF) model [11] and the features are extracted from both RGB image and the depth data such as color and texture. Finally, the extracted features are concatenated for object detection.

In the paper by Asvadi et al.[15], the integration of 2D-RGB camera images and 3D-LIDAR data is done by using the Kalman Filters (KF) based algorithm, which is the measurement fusion model [20]. KF-based fusion [16] is used to integrate point cloud data with RGB image data to get the fused centroid in 3D.

I. I. Fouad et al.[17] combined the depth and RGB segmentation using the following method. To extract boundaries of objects, edge detection is the first step

applied to RGB and Depth images separately. To enhance the edges by filling gaps between edges and to remove noise, the morphological operations is the second step applied to the first step result. The third step, "connected component labeling" [17], is applied to assign a label to continuous regions. The fourth step is to extract missing components in RGB and depth image separately. The final step is combining the result of RGB and depth images segmentation to form the final result.

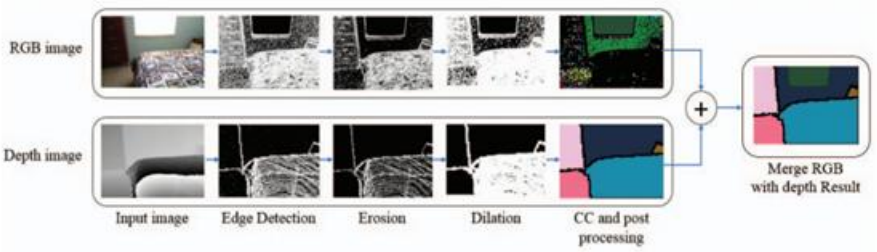

Figure 4.6: RGB-D full segmentation process [17].

M. J. Dahan et al.[18] combines the depth and color image information to get enhanced image segmentation from real devices in synergy using upsampling and downsampling methods as shown in Figure 4.7. Depth resolution was increased by following steps.

The first step is that the capturing devices must be calibrated respectively. The second step is to down-sample the high-resolution image to the same resolution as the depth image, and then down-sample both images to a lower resolution. The third step is to use joint bilateral up-sampling in two consecutive steps.

1. The first step is aimed at filling in the missing depth regions based on the color information

2. The second step aligns the two images while up-sampling the depth to the higher resolution.

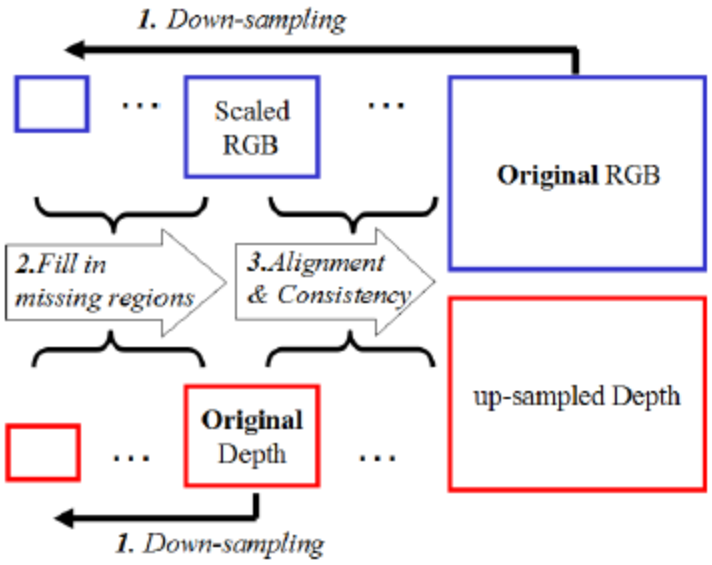

Figure 4.7: Down-sampling the color and depth images and up-sampling the depth image using multistep joint bilateral upsampling [18].

This research is inspired by all the above papers, to merge the camera image pixels with point cloud distance data using the TIFF image file format.

CHAPTER 5: CURRENT RESEARCH WORK

The goal of the research was to develop a software method for embedding spatial (distance) data into a visual image in Tagged Image File Format (TIFF). In order to do this research spatial data of the object was required. To get the distance data of the object, it was decided to use Kinect v2 sensor and Robotic Operating System (ROS).

libfreenect2 driver and iai_kinect2 library for the Kinect v2 were installed to interface with the ROS in Ubuntu. To install these packages, I followed the installation steps from “ROS Robotics by Example - Second Edition” book by Carol Fairchild and Dr. Thomas L. Harman [9].

The experimental setup has the different color pages (objects) with the flat whiteboard as a background, at a particular distance from the Kinect sensor.

Figure 5.1: Kinect sensor and red color page on the whiteboard

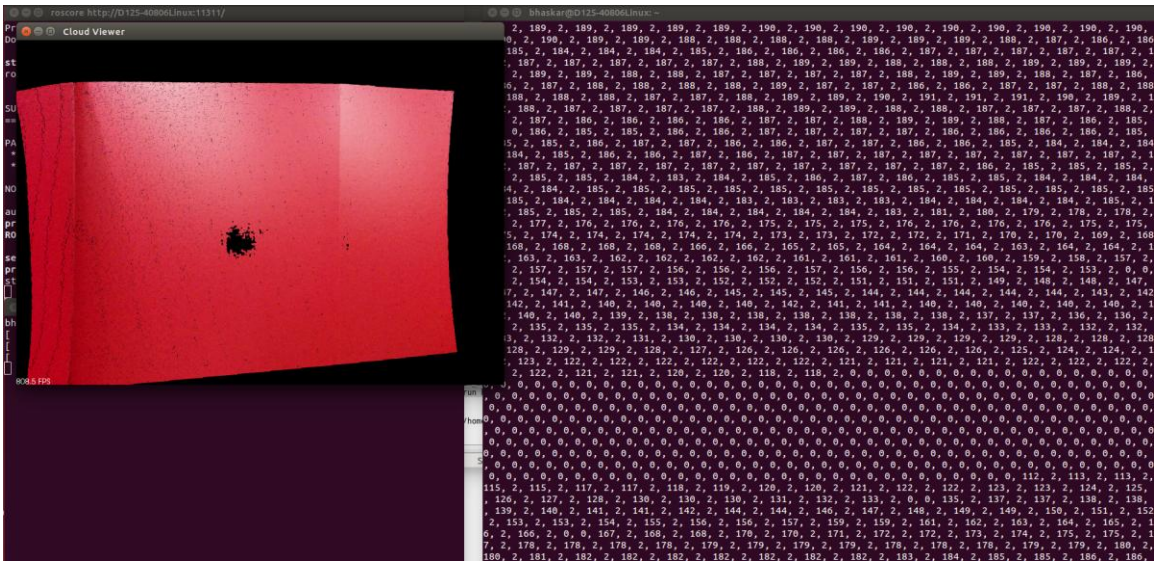

Figure 5.2: Color image and depth values from Kinect sensor

Kinect digital cameras captured a still image. Collected the distance data of objects using

$rostopic list /kinect2/hd/ image_depth_rect

5.1 Collected depth image bytes from kinect V2 sensor

Below is the part of collected image_depth data for 96 cm between the sensor and the object on the white board.

$rostopic echo /kinect2/hd/image_depth_rect | tee hd_depth_rect_whited0.txt

header:

seq: 27421

stamp:

secs: 1491676173

nsecs: 331020959

frame_id: kinect2_rgb_optical_frame

height: 1080

width: 1920

encoding: 16UC1

is_bigendian: 0

step: 3840

data: [0, 0, ...200,3,201,3.....]

5.2 Getting distance data from collected depth image bytes

Kinect image depth data consists of two numbers that have to be combined to get the actual distance. Using this formula below [31] will calculate the distance using two depth_image data points.

temp_val = depth_image->data[index] + (depth_image->data[index + 1] << 8);

=200 + 3 (shift left by 8)

=~96 cm

I have written C program to read the collected depth image bytes and to convert to distance values. The program is available in Appendix A.

5.3 Libraries and Applications:

We used TIFF Image to embed distance data in the image data, as TIFF has clear specifications and is easy to code. To read and write TIFF files, the libTIFF library [30] is required. Please refer to Appendix B for the installation steps of the LibTIFF library.

TIFF 4.0.9 library can be downloaded from any of the following links
http://www.simplesystems.org/libtiff/ or
http://www.linuxfromscratch.org/blfs/view/svn/general/libtiff.html or
ftp://download.osgeo.org/libtiff/

To load and save a TIFF image, a graphics library imlib2 [29] is necessary. Hence we used imlib2.loader_tiff.c files to load and save the TIFF image. Please refer to Appendix B for the installation steps of Imlib2 which are used for fast file loading, saving, rendering, and manipulation of all image types particularly TIFF image by using the libTIFF library.

imlib2 1.4.10 can be downloaded from following link
http://www.linuxfromscratch.org/blfs/view/cvs/x/imlib2.html

We choose open source tool SXIV [28], which supports viewing the TIFF image using the imlib2 library in Ubuntu.

The latest version of SXIV can be downloaded from the following links
https://github.com/muennich/sxiv or
https://launchpad.net/ubuntu/+source/sxiv or

http://manpages.ubuntu.com/manpages/trusty/man1/sxiv.1.html

Please refer to Appendix B for installation steps.

Two symbolic links are created in /usr/lib/x86_64-linux-gnu

1. SXIV to Imlib2 and then
2. Imlib2 to libTIFF using sudo ln -s command

A symbolic link, also termed a soft link, is a special kind of file that points to another file.

$sudo ln -s /home/bhaskar/imlib2-1.4.10/src/lib/.libs/libImlib2.so.1.4.10 libImlib2.so.1

$sudo ln -s /home/bhaskar/tiff-4.0.8/libtiff/.libs/libtiff.so.5.2.6 libtiff.so.5

$ ls -la /usr/lib/x86_64-linux-gnu | grep "\-> /home"

lrwxrwxrwx 1 root root 68 Nov 6 17:39 libImlib2.so.1 -> /home/bhaskar/thesis/imlib2-1.4.10/src/lib/.libs/libImlib2.so.1.4.10

lrwxrwxrwx 1 root root 55 Nov 7 18:38 libtiff.so.5 -> /home/bhaskar/tiff-4.0.8/libtiff/.libs/libtiff.so.5.2.6

5.4 Commands to view the image in sxiv by installing libraries

$ cd tiff-4.0.9/

$ make clean

$ sudo make

$ sudo make install

```
$ cd imlib2-1.4.10/
$ make clean
$ sudo make
$ sudo make install

$ cd sxiv-master/
$ make clean
$ sudo make
$ sudo make install

$ sxiv rgb.tiff
```

Figure 5.3:TIFF image shown in SXIV image viewer

5.5 Merging distance data to TIFF fields:

To merge distance data to the TIFF image, we wrote a new code:

Created new tag to store distance data as below

```
#include "tiffio.h"

#define TIFFTAG_T01    35001  //88 B9

#define TIFFTAG_T09    35009  //88 BA
```

Created a custom field for the created tags in uhclFields stucture

```
static const TIFFFieldInfo
uhclFields[] = {
```

```
          { TIFFTAG_T01, 1, 1, TIFF_ASCII, FIELD_CUSTOM,
TRUE, 0, "uhcl01"},
         { TIFFTAG_T09, 1, 1, TIFF_ASCII, FIELD_CUSTOM,
TRUE, 0, "uhcl09"},
```

We merged the above created uhcl fileds to existing the TIFF fields' structure using TIFFMergeFieldsInfo in Tiff_dirinfo.c file.

New distance data is added to TIFF fields using TIFFSetField()

```
if (!TIFFSetField(tif,TIFFTAG_T09,dist))
```

SaveImage_uhcl() function in sxiv_main.c file is used to save the added new tags in the original TIFF image with the help of imlib2.loader_tiff.c file.

LoadImage_uhcl() function in sxiv_main.c file is used to view the updated TIFF image using imlib2.loader_tiff.c file.

TIFF fields' values are read using TIFFGetField()

```
if (!TIFFGetField(tif,TIFFTAG_T09,&val5))
```

To accomplish the merging distance data to TIFF image in backward compatibility, we have written some other functions. Please refer to supplemental file for the source code of this research.

CHAPTER 6: RESULTS

The distance data collected by Kinect v2 sensor is augmented to the TIFF image by adding new tags 35001 and 35009 using libraries LibTIFF, Imlib2, and the application SXIV.

The section 6.1 below shows the input image details before adding distane data.

6.1 Input image details: rgb.tiff (70 KB)

Tif header magic number (byte order) is 0x4d4d

Tif base (byte order) MM

Tif header version is 0x2a

Tif (directory) Offset is 0x115da (71130)

Tif file size 0x1172e (71470)

Tif dircount 0xe (14)

Tif dirsize 0xc (12)

256 tag with name ImageWidth has value 157,0,0

257 tag with name ImageLength has value 151,0,0

258 tag with name BitsPerSample has value 8,0,0

259 tag with name Compression has value 1,0,0

262 tag with name PhotometricInterpretation has value 2,0,0

269 tag with name DocumentName has value rgb-3c-8b.tiff,28871984

273 tag with name StripOffsets has value 28873680,0,0

277 tag with name SamplesPerPixel has value 28835843,0,0

278 tag with name RowsPerStrip has value 17,0,0

279 tag with name StripByteCounts has value 28873760,0,0

284 tag with name PlanarConfiguration has value 28835841,0,0

297 tag with name PageNumber has value 28835840,1,0

305 tag with name Software has value GraphicsMagick 1.2 unreleased Q16 http://www.GraphicsMagick.org/,28873984

339 tag with name SampleFormat has value 28835841,1,0

The section 6.2 shows the output image details after adding distance data as 2 new tags. We can observe the TIFF dircount is increased by 2 and 2 new tags information is added at the end of 6.2 section.

6.2 Output image details: uhcl_tag.TIF (321 KB)

Tif header magic number (byte order) is 0x4d4d

Tif base (byte order) MM

Tif header version is 0x2a

Tif (directory) Offset is 0x115da (71130)

Tif file size 0x1172e (71470)

Tif dircount 0x10 (16)

Tif dirsize 0xc (12)

256 tag with name ImageWidth has value 157,0,0

257 tag with name ImageLength has value 151,0,0

258 tag with name BitsPerSample has value 8,0,0

259 tag with name Compression has value 1,0,0

262 tag with name PhotometricInterpretation has value 2,0,0

269 tag with name DocumentName has value rgb-3c-8b.tiff,28871984

273 tag with name StripOffsets has value 28873680,0,0

277 tag with name SamplesPerPixel has value 28835843,0,0

278 tag with name RowsPerStrip has value 17,0,0

279 tag with name StripByteCounts has value 28873760,0,0

284 tag with name PlanarConfiguration has value 28835841,0,0

297 tag with name PageNumber has value 28835840,1,0

305 tag with name Software has value GraphicsMagick 1.2 unreleased Q16 http://www.GraphicsMagick.org/,28873984

339 tag with name SampleFormat has value 28835841,1,0

35001 tag with name TIFFTAG_T01 tag has 0x1bc9970 and string Following are own uhcl tags

35009 tag with name TIFFTAG_T09 tag has 0xab1050 and value is 0,0xab1050

Picture comparison of the input and output images shown in Figure 6.1.

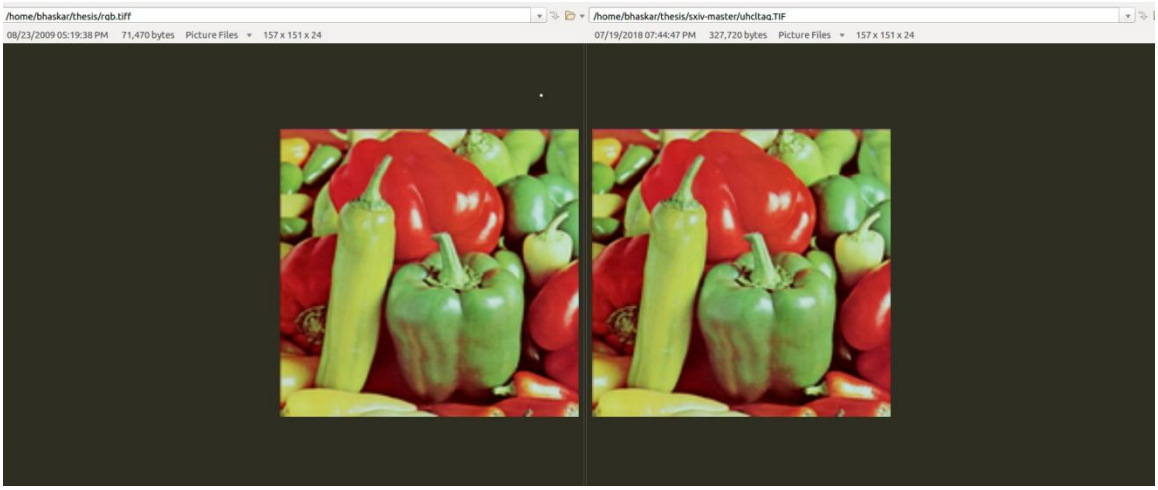

Figure 6.1: Picture comparison of the input and output images

The hexadecimal format of input and output images (Figure 6.2) shows that input image has 14 tags whereas output image with distance data has 16 tags as we added 2 new tags: 88 B9 (35001) and 88 BA (35009).

Figure 6.2: Hexadecimal comparison of the input and output images

The HTML report of both the TIFF input and output images in Hexadecimal form is available in the supplemental file.

CHAPTER 7: CONCLUSION AND FUTURE WORK

Developed a software method for embedding spatial (distance) data into a visual image in Tagged Image File Format (TIFF) to, possibly, avoid optical illusions which occur due to lack of distance information and to create 3D images using embeded distance data with the image. The spatial data is collected from the Kinect v2 sensor which uses structured light of sensor to get the accurate distance for small range (upto 4 meters). Distance information is then merged with the image in TIFF as TIFF has good documentation and has open source LibTIFF library. TIFF also uses lossless compression technique when re-editing, saving the image. Finally, the distance data collected by Kinect v2 sensor is augmented to the TIFF image by adding new tag values 35001 and 35009 (which are only for local use) using libraries LibTIFF, Imlib2, and the application SXIV.

The TIFF flexibility to add new tags and portability gives a lot of scope for current research work (embeding spatial data with the visual image) and possible expansion in future.

Future work can capture the distance data more accurately by using LIDAR instead of Kinect v2 sensor. LIDAR technology provides horizontal and vertical information at the high spatial resolution and vertical accuracies, whereas, the RGB-D camera has less accuracy but is very cost effective. Kinect's RGB-D sensor is also unreliable for outdoor situations and has a maximum range of about 4 meters precluding applications that require detection at larger distances. By contrast, 3D laser range sensors, also known as Light Detection and Ranging (LIDAR), such as the Velodyne HDL-64E, can easily operate outdoors and have maximum ranges upwards of 50 meters.

Current image viewers, will not be able to interpret the spatial data, but in a backward compatible manner will still be able to display the original image. To see the benefit of the spatial data that may be embedded in the TIFF file we would need to create a new TIFF viewer. The new TIFF viewer that can properly interpret and display the combined visual and distance data in interesting and useful ways.

REFERENCES

[1] Juan Li, "Fusion of LIDAR 3D points Cloud with 2D Digital Camera Image," M.S. thesis, Dept. Elect and comp. Eng., Oakland Univ., Rochester, Michigan, 2015.

[2] https://www.awaresystems.be/imaging/tiff/specification/TIFF6.pdf

[3] https://www.awaresystems.be/imaging/tiff.html

[4] https://cdn.earthdata.nasa.gov/conduit/upload/6852/geotiff-1.8.1-1995-10-31.pdf

[5] https://trac.osgeo.org/geotiff/

[6] https://earthdata.nasa.gov/user-resources/standards-and-references/geotiff

[7] http://wiki.ros.org/indigo/Installation/Ubuntu from 1.1 to 1.7 'Getting rosinstall'

[8] http://wiki.ros.org

[9] "ROS Robotics by Example - Second Edition" book by Carol Fairchild and Dr. Thomas L. Harman, 2017.

[10] http://www.fileformat.info/format/tiff/egff.htm

[11] Y. Cao, C. Shen, and H. T. Shen, "Exploiting Depth from Single Monocular Images for Object Detection and Semantic Segmentation," in IEEE Transactions on Image Processing, vol. 26, no. 2, pp. 836-846, Feb. 2017. doi: 10.1109/TIP.2016.2621673

[12] S. Gupta, R. Girshick, P. Arbelaez, and J. Malik, "Learning rich features from RGB-D images for object detection and segmentation," in Proc. Eur. Conf. Comp. Vis., 2014.

[13] S. Song and J. Xiao, "Sliding shapes for 3d object detection in depth images," in Proc. Eur. Conf. Comp. Vis., 2014.

[14] L. Bo, X. Ren, and D. Fox, "Depth kernel descriptors for object recognition," in Proc. IEEE/RSJ Int. Conf. Intell. Robt. Syst., September 2011.

[15] A. Asvadi, P. Girão, P. Peixoto and U. Nunes, "3D object tracking using RGB and LIDAR data," 2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC), Rio de Janeiro, 2016, pp. 1255-1260. doi: 10.1109/ITSC.2016.7795718

[16] J. Gao, C. J. Harris, "Some remarks on kalman filters for the multisensor fusion", Information Fusion, vol. 3, no. 3, pp. 191-201, 2002.

[17] I. I. Fouad, S. Rady and M. G. M. Mostafa, "Efficient image segmentation of RGB-D images," 2017 12th International Conference on Computer Engineering and Systems (ICCES), Cairo, Egypt, 2017, pp. 353-358. doi: 10.1109/ICCES.2017.8275331.

[18] M. J. Dahan, N. Chen, A. Shamir and D. Cohen-Or, "Combining color and depth for enhanced image segmentation and retargeting," 2011 Vis Comput (2012) 28:1181–1193. DOI 10.1007/s00371-011-0667-7.

[19] B. Yohannan and D. A. Chandy, "A novel approach for fusing LIDAR and visual camera images in unstructured environment," 2017 4th International Conference on Advanced Computing and Communication Systems (ICACCS), Coimbatore, 2017, pp. 1-5. doi: 10.1109/ICACCS.2017.8014604

[20] L. Zhou, "Fusing laser point cloud and visual image at data level using a new reconstruction algorithm," 2013 IEEE Intelligent Vehicles Symposium (IV), Gold Coast, QLD, 2013, pp. 1356-1361.doi: 10.1109/IVS.2013.6629655.

[21] C. Premebida, J. Carreira, J. Batista, and U. Nunes, "Pedestrian detection combining RGB and dense LIDAR data," 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, Chicago, IL, 2014, pp. 4112-4117. doi:10.1109/IROS.2014.6943141

[22] Du Hao, Henry Peter, Ren Xiaofeng, Cheng Marvin, Dan B Goldman, Steven M. Seitz, Dieter Fox, "Interactive 3D modeling of indoor environments with a consumer depth camera", Proceedings of the 13th International Conference on Ubiquitous Computing, 2011.

[23] Julius Schoning and Gunther Heidemann, “Taxonomy of 3D sensors” A Survey of State-of-the-Art Consumer 3D-Reconstruction Sensors and Their Field of Applications, Published in the proceedings of VISAPP 2016.

[24] W. Nie, Q. Li, G. Zhong, and H. Deng, "Indoor 3D path planning using a Kinect V2 sensor," 2017 IEEE 3rd Information Technology and Mechatronics Engineering Conference (ITOEC), Chongqing, 2017, pp. 527-531. doi: 10.1109/ITOEC.2017.8122352

[25] B. Shen, F. Yin and W. Chou, "A 3D Modeling Method of Indoor Objects Using Kinect Sensor," 2017 10th International Symposium on Computational Intelligence and Design (ISCID), Hangzhou, China, 2017, pp. 64-68. doi: 10.1109/ISCID.2017.12

[26] Henry, P., Krainin, M., Herbst, E., Ren, X., & Fox, D. “RGB-D mapping: Using Kinect-style depth cameras for dense 3D modeling of indoor environments”. International Journal of Robotics Research 31.5 (2014), pp. 647–663.

[27] Newcombe, Richard A, “Kinect Fusion: Real-time dense surface mapping and tracking”. IEEE International Symposium on Mixed and Augmented Reality IEEE, 2012, pp. 127–136.

[28] https://github.com/muennich/sxiv

[29] http://www.linuxfromscratch.org/blfs/view/cvs/x/imlib2.html

[30] http://www.simplesystems.org/libtiff/

[31] https://answers.ros.org/question/257141/getting-and-reading-depth-values-from-kinect2-sensor/

APPENDIX A

C program to get the distance data from collected kinect depth image bytes

C program to calculate the distance data between kinect sensor and object from given input file

```
#include <stdio.h> //standard input output header
#include <stdlib.h> //standard library, includes functions involving memory allocation, process control, conversions and others

        long fsize; //Long is a data type with 32 bits (4 bytes)
        int cnt=0;
int main(int argc, char**argv)
{
        char *buf;  //buf is a pointer variable points to a character (used to store the address of character)
        static uint* dist;
        FILE *f;
        int ht,wt,count;
        int p1, p2,d,i;
        static int k=0;
        int readnumber_uhcl (int *i)
        {
                int p = *i;
                while(buf [++*i] != ',' )
                {
                        if (*i >= fsize)
```

```
                    break;
                }
                buf[*i]='\0';
            return atoi(&buf[p+1]); //converts the string argument buf to an integer
        }
    f = fopen("./hd_depth_rect_whited0.txt","ro"); //open the file in read only mode
//hd_depth_rect_whited0.txt
    if (f == NULL)
    {
       printf("File not present\n");
       exit(0);
    }

    fseek(f, 0, SEEK_END); //To move file pointer position to end of the file f
    fsize = ftello(f);      //Getting file size from current value of the position
indicator
    fseek(f, 0, SEEK_SET); //To set file pointer to beginning of the file

    buf = malloc(fsize + 1);//allocates requested (size) memory
    dist=calloc(fsize,sizeof(int));
    fread(buf, fsize, 1, f); //reads file data size to an array buff
    fclose(f);  //closes the file f

    buf[fsize]='\0';
```

```
//capture height and width
count =0; //--ignore first seven lines
while (count < 6)
{
  if(buf[k]=='\n')
    count ++;
  k++;
}

//7th line height:
sscanf(&buf[k],"height: %d",&ht); // reads height from character string buf
while(buf[k++]!= '\n'); //--skip to width
sscanf(&buf[k],"width: %d", &wt); // reads width from character string buf

printf("height:%d width:%d\n",ht,wt); // printing height, width values
while(buf[k++] != '['); //-- to get the data

      // to print distance in arrays
        while(k<=fsize -4)
      {
      p1= readnumber_uhcl(&k);
      p2= readnumber_uhcl(&k);
      d= (p1 + (p2 << 8))/10; //small number(p2) is left shift by 8 times and
result is in cm (1cm=10 mm)
      dist[cnt]=d;
```

```
        //dist[cnt]= convert(d);  //to convert distance data to binary
        // printf ("data %d \t ", d);
      cnt++;
        }
        printf ("Array count and fsize is %ld,%ld\n", cnt,fsize);
        for (i=0; i<cnt;i++)
                printf ("%d ", dist[i]);
        printf ("\n end \n");
        printf("\n  array pointer %d \t %d \t %d \n",dist,&dist,*dist);
        printf ("\n Array count and fsize is %ld,%ld\n", cnt,fsize);
        printf("\ndistance code ended\n");
 return 0;
}
```

APPENDIX B

Installing commands for libraries and application

Run following commands using readme file in libtiff

./configure

sudo make

sudo make install

Run following commands using readme file of imlib2

./configure

make

make install

Follow the commands using readme file of sxiv-master

make

make install

make config.h