REAL TIME INDOOR ENVIRONMENT MODELING

by
Deniz Beker

Submitted to the Institute of Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science
in
Electrical and Electronics Engineering

Yeditepe University
2014

REAL TIME INDOOR ENVIRONMENT MODELING

APPROVED BY:

Prof. Dr. Cem Ünsalan (Thesis Supervisor)	[illegible]
Assoc. Prof. Dr. Yusuf Sinan Akgül	[illegible]
Assist. Prof. Dr. Dionysis Goularas	[illegible]

DATE OF APPROVAL:

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my family for always being supportive of my endeavors.

I would like to express my deep gratitude both personally and professionally, to my advisor, Cem Ünsalan, without whose guide and encouragement the completion of this study would be impossible. I will remember every word of his valuable advices through all my future studies.

I would like to thank EE faculty and research assistants for making my two years at Yeditepe University a great experience.

Finally, I would like to thank all my friends and loved ones outside of school for their motivation.

ABSTRACT

REAL TIME INDOOR ENVIRONMENT MODELING

Building survey and indoor 3D mapping is getting popular as their application areas expand. A sub area for building survey is architectural renovation. There are various 3D range sensors for these applications. Among these, laser scanners are the most precise ones. However, they are expansive and overqualified for most architectural renovation projects. Therefore, we propose using an MS Kinect sensor for this purpose. Besides, we develop a complete embedded system both in hardware and software. In our system, range and RGB data is grabbed via a TI OMAP4460 based PandaBoard-ES. Then, the same board transfers this data to a remote PC through the TCP/IP protocol in real time. There, the received data is processed via Point Cloud Library (PCL) and Open CV. In processing, we take indoor modeling as wall segmentation and we merge segmented planes. To do so, we use RANSAC and SURF. We tested the performance of our system in various indoor scenarios. We provided the experimental results as well as their evaluation in the conclusion section of this study.

ÖZET

GERÇEK ZAMANLI İÇ MEKAN MODELLEME

İç mekan modelleme ve üç boyutlu haritalandırma işlemleri gün geçtikçe daha yaygın kullanılmaya başlanmıştır. Bunlardan bir tanesi de mimaride restorasyon için kullanılan ölçü (rölöve) alma işlemidir. Bu tarz uygulamalar için günümüzde farklı üç boyutlu mesafe sensörleri kullanılmaktadır. Bunlar arasında otomatik lazer tarayıcılar en hassas sonucu vermektedir. Ancak bu cihazlar pahalı oldukları gibi birçok kullanım için de ihtiyaçtan fazlasını sunarlar. Bu nedenle, bu çalışmada ölçü alma işlemi için MS Kinect tabanlı yeni ve ucuz bir sistem önerilmektedir. Hazırladığımız sistemde sensörden alınan RGB ve derinlik bilgisi TI OMAP4460 tabanlı PandaBoard-ES ile işlenerek uzak bir sunucuya TCP/IP protokolü ile gerçek zamanlı olarak aktarılmaktadır. Sunucu tarafında da alınan bilgileri işlemek için Point Cloud Library (PCL) ve OpenCV kullanılmıştır. Burada iç mekan modellemesi ağırlıklı olarak duvar bölütlemesi ile gerçeklenir. Bunun için de RANSAC ve kesişme saptama yöntemleri kullanılmıştır. Ayrıca, elde edilen görüntüleri hızlı ve yüksek doğruluk oranında birleştir-mek için SURF tabanlı bir yöntem uyguladık. Algılanan her imge arasındaki dönme açısını SURF yardımı ile hesaplayarak üç boyutlu ortamda bölütlendirmeleri birleştir-dik. Önerdiğimiz bu sistemi çeşitli iç mekan koşullarında test ettik. Bunları çalışmamızın sonunda yorumladık.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS/ABBREVIATIONS

ALS	Alternating Least Squares
ARSSAC	Adaptive Real-Time Random Sample Consensus
CRF	Conditional Random Field
CS	Consensus Set
$d(\overrightarrow{r_1}, \overrightarrow{r_2})$	Euclidian distance between two points
$D(\overrightarrow{p}, X)$	The distance between a point and model set
IDL	Interface Description Language
MLE	Maximum Likelihood Estimation
MSS	Minimal Sample Sets
OLS	Ordinary Least Squares
OS	Operating System
PCL	Point Cloud Library
PDF	Probability Density Function
PLS	Partial Least Squares
RANSAC	Random Sample Consensus
WLS	Weighted Least Squares

1. INTRODUCTION

Modern range scanning technologies have been advancing towards the 3D modeling of a real world scene. These ranging techniques is commonly used in the dangerous areas for humans (with the help of robotics), medical imaging (3D visualization of organs and body), virtual reality (reconstruction of existing buildings), computer graphics (3D models, motion capture and adaptation to 3D) and computer vision (object, background, face recognition).

In architecture, building survey is mainly used for renovation purposes. For renovation, the building plan has to be reconstructed. Besides, the building should be modeled in 3D. Therefore, there are various studies focusing on this problem. As for range sensing, two generally accepted solutions are professional LIDAR range scanners and laser meters. Although laser meters are affordable, they do not provide a comprehensive solution to the problem. On the other hand, LIDAR range scanners provide the best precision in a professional manner. Unfortunately, they are expensive and need a setup to function effectively. Besides, their mobilization is a problem since they are too sensitive to any jolt during operation. Therefore, they need to be setup many times to measure blind spots in indoor environments. On the other hand, for most renovation projects the resolution provided by a LIDAR sensor is not necessary. Therefore, an end to end system may be preferable to achieve the acceptable resolution with minimal cost and flexible operation conditions.

There are various 3D surface acquisition methods. We can classify these methods in two main branches: active and passive. Passive scanning is a group of techniques which are not radiating any sort of light, pattern or track to capture the difference according to the environment [1]. Instead, they use the environment's attributes to obtain range data. Stereo vision is the commonly used method in passive branch. The term stereo vision (or binocular vision) is the method to obtain 3D models from the 2D images which are taken from the different angle of an object. The taken images are reconstructed according to the matching line and the geometry calculations. These 2D image captures can be obtained by multiple stationary or single mobile camera. The stereo vision triangulation is illustrated in Fig. 1.1.

Structured light is the method to project a pre-known light pattern to the surface and measure the differences of the pattern on the surface [2]. Usually a laser or an infrared light source is used not to disturb the other capture systems or not to get disturbed by the visible light sources. This method is used commonly to detect the object or the surface. It has also the usage area in industry as automatize the robots. The structured light illustration is shown in Fig. 1.2.

We can represent 3D data as 2D depth image or as 3D point cloud. In 2D depth image,

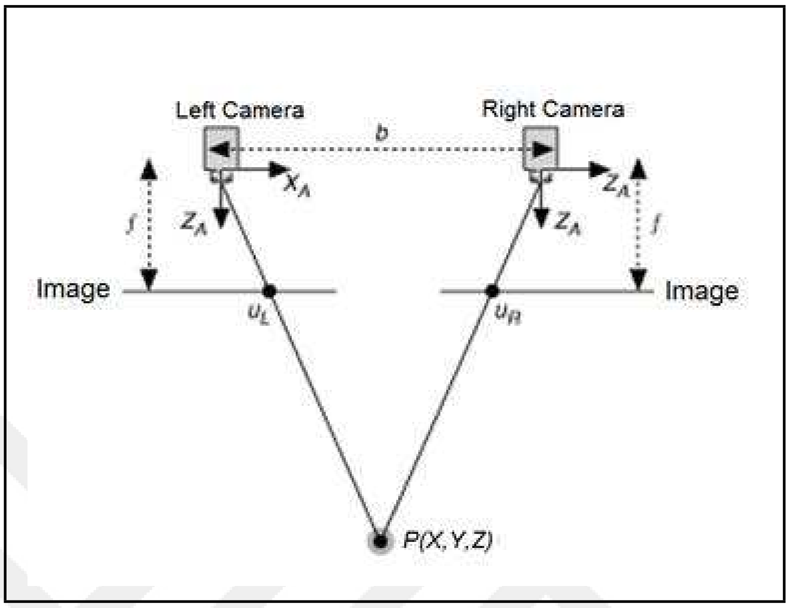


Figure 1.1. Stereo vision triangulation

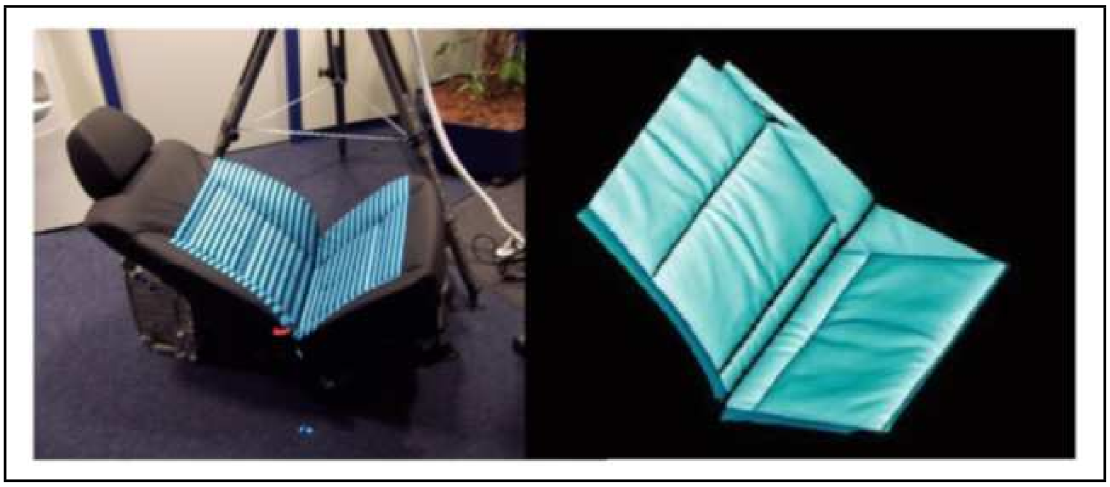


Figure 1.2. Structured light system

the data is represented as distance or disparity. In distance based depth image, all the pixel values are kept as camera to object distance instead of RGB layers. Each pixel is equal to real world distance or a proportion of it. However, in disparity maps, the values which represent the difference with neighbor pixels are imaginary.

Microsoft Kinect is a compact, calibrated, end user product which is designed as a motion capture device for Xbox games. It provides depth images along with RGB. We use it in the proposed system to capture environment's depth and color values. We also use TI OMAP 4460 based PandaBoard-ES as the capturing and networking processor board.

The Point Cloud Library (PCL) is a standalone, large scale, open project for 2D/3D image and point cloud processing. The PCL framework contains numerous algorithms including filtering, feature estimation, surface reconstruction, registration, model fitting and segmentation [3]. It allows to create 3D RGB-D clouds [4]. We use PCL to process 3D RGB-D data for segmentation of the planes and registration. In next chapters, we will explain the hardware and environment preparation (implementation details), methodologies used, serialization, RANSAC, SURF, registration and the software.

2. LITERATURE REVIEW

This thesis is based on two main parts. The first one is the theoretical approach to 3D environment modeling. The second one is the real time application of it. Therefore, we divided the literature review chapter into two main sections.

2.1. 3D ENVIRONMENT MODELING

In this section, we provide a wide scale 3D environment modeling techniques which use any kind of depth sensing system and optionally a robot system. Du *et al.* [5] present a real time and offline 3D indoor mapping system. They use a consumer depth camera to capture RGB and depth frames. The system also offers registration of all captured frames by using a failure detection system. This offers online feedback and hints, tolerates human errors and alignment failures. In interactive method, they can provide an alignment error minimum of 0.05 meters and maximum of 0.11 meters.

Xiong and Huber [6] present semantic 3D models of building to determine the key components of a facility, such as walls, floors, and ceilings. They aim to automate the modeling process of an indoor environment. They use a Conditional Random Field (CRF) model to discover contextual information. They also extract planar surfaces and provide the comparison results between the CRF algorithm and norm based Regularized Logistic Regression.

Liu *et al.* [7] present a new algorithm for generating compact 3D models of indoor environments with mobile robots. They fit a low-complexity planar model to 3D data collected from a range finder. Fitting to a planar model minimizes the measurement error, point cloud (data) size and instantly creates a 3D model.

Biber *et al.* [8] propose a method to acquire a realistic, visually convincing 3D model of indoor environments based on a mobile platform. This platform is equipped with a laser range scanner and a panoramic camera. They separated the system in four part: calibration, map generation, texture generation, and stereo processing. They obtain 2D/3D map and walls from map generation part. They obtain wall and floor textures from the texture generation part. They obtain point clouds from stereo processing. They combine all and provide final point cloud.

Lu and Wang [9] propose an approach to model indoor environments from depth videos. In their scene, they use a stationary camera to extract 3D spatial layout of the room and modeling objects as 3D cubicles. Their difference from similar works is that they use human-environment interaction to determine objects.

Haehne *et al.* [10] propose an algorithm for full 3D shape reconstruction of indoor and outdoor environments with mobile robots. They obtain data by using a laser range finder which is installed on a mobile robot. Their approach is to reduce the complexity of the models by using a mesh simplification technique to approximate environments using flat surfaces. They introduce indoor and outdoor performance and the comparison with the conventional methods.

Silberman *et al.* [11] present an indoor scene segmentation technique to distinguish objects by using color and depth images. Their method is based on CRF model to evaluate a range of different representations for depth information. They propose a novel prior on 3D location. As result, they clearly demonstrate the utility of structured light sensors for scene understanding.

Filliat *et al.* [12] present an autonomous mobile robot to explore unknown indoor environments and build a semantic map. Their aim is to obtain information like rooms, their connectivity, and the material of wall and ground. Their object detection approach is based on color and depth images to create 3D RGB-D maps, color and texture information through a neural network for robust object recognition. They also provide the performance results in real environments.

Henry *et al.* [4] present a 3D RGB-D based mapping system which utilizes a novel joint optimization algorithm combining visual SIFT features and shape-based alignment. They combine visual and depth information to obtain view-based loop closure detection. Then, they apply pose optimization to achieve globally consistent maps. They provide two scenes as test data which are large indoor environments.

Kunze *et al.* [13] present a semantic based autonomous indoor environment modeling. Their purpose is to create an autonomous robot which makes daily tasks like calling elevator, preparing coffee, ordering a sandwich while it detects and classifies the objects to perform these tasks. They create a semantic map to determine the environment type (like office room, kitchen, and laboratory) by using objects found in a room. The decision algorithm is based on the probability of objects that can be found in the related room. Then, they relate detected rooms with possible tasks.

Bills *et al.* [14] present an autonomous flying vehicle based on 2D images. Their purpose is to detect and classify the indoor environment in order to plan and to control the Miniature Aerial Vehicles (MAVs). They implement vision algorithms based on perspective cues to estimate the flight direction. They also provide test results realized with a co-axial miniature helicopter and a toy quad-rotor.

Khalil *et al.* [15] present a method to extract the information of an indoor scene and its reconstruction by combining geometric and topological information of an indoor environment. They extract geometric and semantic data simultaneously by using a digital photometry software and an independent Graphical User Interface (GUI). They also detect surfaces by using two images and fit them to a 3D model.

Cheng *et al.* [16] present a method for texture mapping 3D models of indoor environments with an autonomous manner. Their method starts by selecting images whose camera poses are well-aligned in two dimensions. They apply alignment in the third dimension. Then, they align images to geometry and to each other. This method allows us to obtain visually consistent textures even in the presence of inaccurate surface geometry and noisy camera poses. Then, they compose a final texture mosaic. They project it onto surface geometry for visualization.

Ikeda *et al.* [17] provide a method to generate 3D environment models. They use an omni-directional stereo based robot which is equipped with a laser range finder. They aim to create simple 3D models in a fast manner at the cost of precision. They first create a robust 3D models composed of layered contours of free spaces. Then, they fit textures extracted from images to these 3D models for visualization.

Elgazzar *et al.* [18] provide a method to model indoor environments by using a low-cost, compact and active range camera. To achieve this, they used BIRIS which is mounted on a pan and tilt motor unit to form a robot. They used it to capture, segment out and register range images. They provide the results of indoor surveying along with the limits and capabilities for an outdoor usage.

Thrun *et al.* [19] provide a new real-time algorithm to acquire compact 3D maps of indoor environments by using a mobile robot equipped with range and imaging sensors. Their work extends maximum likelihood estimation algorithm to multi-surface models and makes it amenable to real-time execution. They experimented their new algorithm in corridor-type environments to illustrate that the successful acquisition of maps in real-time is possible.

Sequeira *et al.* [20] provides a 3D scene analysis system. Their aim in this work is to create CAD models by autonomously scanning indoor environments. They use a mobile robot and a laser range finder on it. The range data is used to build 3D models of objects in real world scenes. They first extract the surface characteristics from range images and approximate the geometry by a 3D triangular surface mesh. Then, they merge the surface descriptors obtained from different viewpoints.

Williams *et al.* [21] describe an approach to register color images with 3D laser scanned models automatically. They use the chi-square statistic to compare color images to polygonal models texture mapped with acquired laser reflectance values. However, according to them, chi-square method is not suitable to the complicated scenes. To prevent it, they introduce two techniques to obtain the initial pose. The first one is to attach a camera to the laser scanner. The second one is to track objects to decouple. The estimated poses serve as an initial guess for their optimization method.

There is also a web site based on similar applications [22]. In this site, the topic "Hooking up a Kinect to your Computer (Using Ubuntu)" is indicated. In this topic, the process to capture three dimensional information by using Kinect, OpenNI and Ubuntu is expressed. Our work contains similar structure in client side. However, we added a remote data communication property to this work. Also, we are using OpenNI library to capture depth and RGB image, then we convert them to 3D point cloud differently. In the same web site, the topic "3D Print and Animate Yourself" is indicated. In this topic, the process to capture an object three dimensionally, to process it with MeshLab and 3D printing the model are expressed. As the alignment is done with MeshLab, it has not the property of real time processing. Also, the model's data capture is done here, where capturing device turns around the object theoretically. However, we have different case in our work. In our case, the capturing device is stable in the mean of three dimensional coordinates and its viewpoint is changing. By using the methods that we describe in further sections, we also applied real time capability to our work by using different alignment methods.

2.2. POINT CLOUD LIBRARY

In this section, the capabilities and limitations of Point Cloud Library (PCL) will be explained. Rusu *et al.* [3] present a new point cloud perception which is PCL. PCL provides an extensive approach to the subject of 3D perception. It is designed to provide features for all the common 3D building blocks. The library contains algorithm modules for filtering, feature estimation, surface reconstruction, registration, model fitting, and segmentation. PCL provides support for OpenMP and Intel Threading Building Blocks (TBB) libraries for multi-core parallelization. K-nearest neighbor search operations are handled with the FLANN library. They also provide the implementation strategies and algorithmic capabilities of this library. PCL is the backbone of our system as all 3D point cloud processing is made by using it in real tine.

Steder *et al.* [23] provide a novel interest keypoint extraction method that operates on 3D point clouds. The algorithm tracks transitions from foreground to background in order to obtain the borders of objects to be identified. Then, they present a feature descriptor that uses the same information. They provide some implementation strategies along with experiments in which individual components are analyzed with respect to their ability to be repeated and matching capabilities. Lastly, they evaluate the usefulness of these components for point feature based object detection methods.

Muja *et al.* [24] present a new method to identify objects and their pose at high speeds. They developed an algorithm called REIN (REcognition INfrastructure) to realize it. REIN can combine a multitude of 2D/3D object recognition and pose estimation techniques in parallel.

The system is based on two new classifiers which are Binarized Gradient Grid Pyramids (BiGGPy) for 2D and VFH (Viewpoint Feature Histograms) for 3D. As we are extracting planar surfaces, this work can be combined to increase effectiveness. There will always be some objects close to planar surfaces. They can be eliminated by using this technique. After elimination, we can apply 3D inpainting methods to estimate the planar surface behind the object.

Holz *et al.* [25] present a segmentation technique to obtain planes in a point cloud. They acquire semantic 3D information by using an RGB-D camera at a frame rate up to 30 Hz. This rate is enough for a robot to detect obstacles and to segment graspable objects while keep tracking of surfaces in general geometry. They use integral images to compute local surface normals. Then they cluster, segment, and classify in normal space and spherical coordinates. They also provide information and experimental results in some situations that both planes and obstacles/objects exist. This article is the most similar one to our work. As they classify the scenes in X,Y,Z direction to obtain planar surfaces, we find the largest planes which are perpendicular to each other by turning point cloud by $\pi/2$ radians each time. The main difference of this article and our thesis is that, we can increase the iteration number to obtain better results and accuracy while they can only get information from three points of view. As we also discard the detected surfaces from the point cloud, each iteration may provide non-repeated results.

Meeussen *et al.* [26] describe an autonomous robotic system which is capable to navigate indoor area. The robot can detect the doors, handles and it can open it. By that way, it is possible for this robot to navigate through all areas. At the same time, it can detect electrical outlets to charge itself when needed. This work provides a combination of the process of precision in micro-scale such as plugging in and a process of accuracy in macro-scale such as indoor modeling in real time. They create a 2D map of the area, excluding height to describe the indoor path. While moving, robot instantly captures depth data and converts it to a 3D point cloud to detect door surface planes. By using Haar Classifier-Cascade method on 2D images, it detects and classifies the door handles and matches with the surface plane which is obtained from 3D plane segmentation. This work is a good combination of 2D feature usage along with 3D ones. Working in 2D is relatively fast and accurate while extracting features as 3D data may contain too much noise along with incomplete/undetected areas in 3D point cloud.

Rusu *et al.* [27] present a new approach for labeling 3D points with different surface primitives using a novel feature descriptor. The Fast Point Feature Histograms (FPFH) is used to encode the underlying surface geometry around a point *p*. FPFH is successful even with a noisy point cloud and it is not dependent on pose or sampling density. By using Conditional Random Fields (CRFs) and by classifying 3D geometric surfaces, their system can successfully segment and label 3D point clouds. They also present experiment results obtained in indoor environments. This article has high importance as it introduces FPFH algorithm which is a fast and accurate 3D histogram extraction method. Our work can be enhanced by implementing FPFH based segmentation along with machine learning system.

Rusu *et al.* [28] present a framework for 3D geometric shape segmentation for close-range scenes. It is mainly aimed to be used in robotics applications as grasping and moving objects autonomously. The scene is captured by a laser scanner and mapped in a 3D point cloud. The objects are segmented out by applying surface normal estimation, Euclidean clustering, and planar decomposition respectively. Then, the algorithm fits segmented model to geometric primitive classes such as planes, spheres, cylinders, and cones. According to the best fitting, the missing point cloud data is reconstructed. Rest of the points are resampled and triangulated to create smooth surfaces. This method has a special importance as it proposes the missing point recovery to increase the efficiency of future point cloud processing.

3. HARDWARE AND SOFTWARE DETAILS OF THE SYSTEM

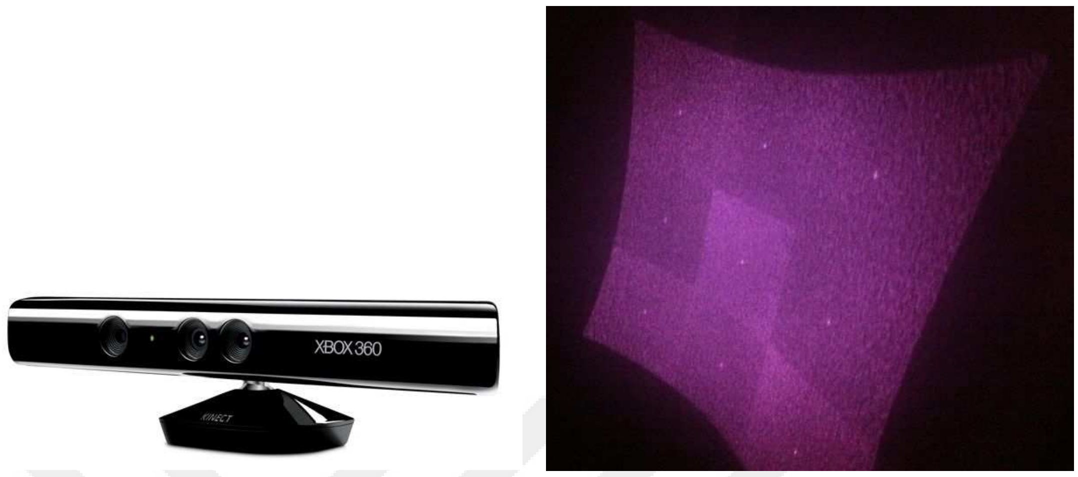
Our system has two main parts: client and server sides. In the client side, we used an embedded hardware and software to capture RGB and depth data, to compress and to transmit data to server side. Even though we just implemented on a rotational table, client side is designed to be adaptable to any mobilized robot. In the server side, we used a PC to decompress, convert 2D RGB and depth images to 3D RGB-D point cloud, process related data in 3D and visualize it. This design allows us to separate the capturing and processing systems. Besides mobilization, by using this topology we can increase the process power of the server side (PC) while the client side stays untouched. Also, it allows us to capture from multiple clients simultaneously in real time. This process may increase the capturing performance which leads to the time efficiency of measurement. In this section, we provide the hardware implementation details of our system. We first focus on the general layout. Then, we focus on the installation details.

3.1. GENERAL LAYOUT

A typical structured light range scanner system consists of a stripe source, a camera, and a processing unit. We used Microsoft's Kinect as an all built-in device. It provides us depth image which has VGA resolution at 30 fps and RGB image with 1280×1024 pixels (SXGA) resolution at 10 fps or 640×480 pixels (VGA) resolution at 30 fps. It has 43° vertical and 57° horizontal field of view. It also has a 2G/4G/8G accelerometer for future mobile implementations. It also provides a measurement range between 0.3 m - 3.9 m (near mode) and 0.7 m - 10 m (far mode) [29]. We used far mode in implementation.

MS Kinect is a compact product based on structured light. The dot based pattern is emitted to the frustum and simultaneously obtained by the IR sensor. It is processed via a PS1080 SoC from raw pattern-illuminated frame to depth image array. Kinect and its pattern can be seen in Fig. 3.1.

As the processing unit for Kinect, we used the TI OMAP4460 processor based PandaBoard-ES and TI DM3730 based BeagleBoard-XM. OMAP4460 has a 1.2 GHz dual core ARM based CPU which is enough for real time data streaming. PandaBoard-ES has also two USB 2.0 ports. We used them to connect Kinect. It also has a 802.11 b/g/n Wi-Fi and a 10/100 Ethernet module. They allow us to stream data over TCP/IP network. We provide the image of this board in Fig. 3.2. We provide the actual hardware setup as a block diagram in Fig. 3.3.



(a) Microsoft Kinect for Xbox 360 (b) Kinect dot based pattern

Figure 3.1. Kinect and its pattern

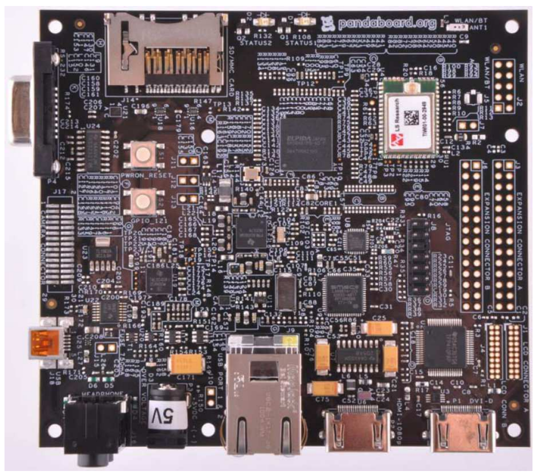


Figure 3.2. PandaBoard-ES embedded board

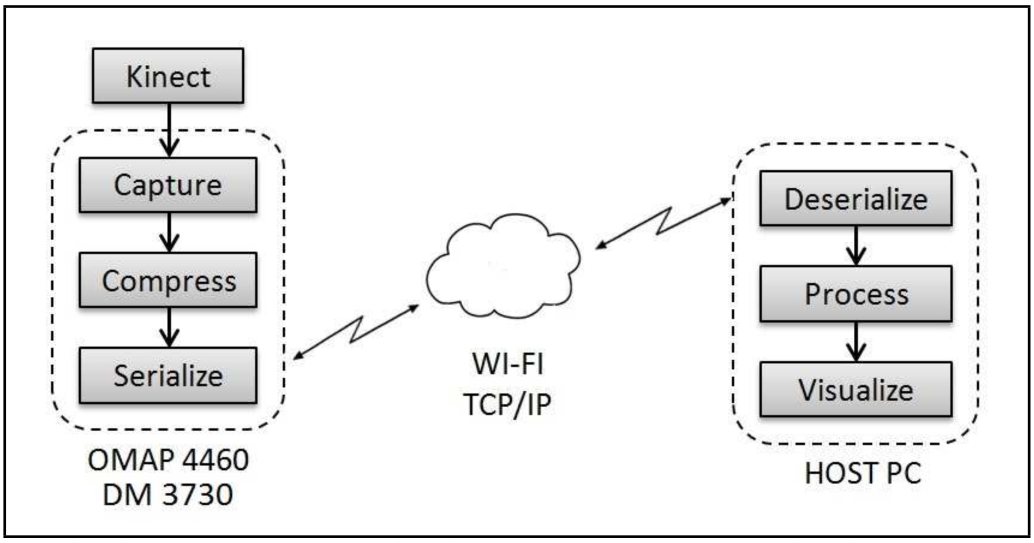


Figure 3.3. Block diagram representation of our system

3.2. ENVIRONMENT INSTALLATION

As mentioned in the previous section, we picked the ARM based PandaBoard-ES as the hardware platform. Due to stability and high support of packages, we selected Ubuntu 12.04 LTS as the operating system. We used the server version of OS to eliminate the excessive resource usage of GUI. We used OpenNI 1.5.4.0 and SensorKinect from Git-hub as the framework and driver. We used OpenNI library because of its capability to be compiled on cross-platforms and its unique structure for different capturing devices [30].

We installed OpenCV, gcc-multilib, libusb, git-core, build-essential, doxygen, graphviz, default-jdk, and freeglut3 libraries from the apt repository for successful compilation of OpenNI on the ARM platform. As the ARM core has floating point unit, we modified the compiler options from soft to hard floating point and compiled OpenNI accordingly [31]. We also edited the floating point configuration parameter of the Kinect driver from soft to hard and compiled it accordingly.

We used the Protobuf library for network serialization. This library is used to serialize the structured object into a streaming buffer in C++, Java, and Python programming languages. This also adds a flexibility to our system like mobile to mobile processing instead of mobile to PC structure. Protobuf uses a specific file structure called “proto” as the Interface Definition Language (IDL) to unify the objects shared between languages. We designed the proto file for the objects that we serialize and compiled it for C++. We also used boost library to parallelize the processes and to handle network communication. It is a general, open-source C++ library used to enhance the capabilities of C++. We used the pre-compiled version for ARM from Ubuntu repository.

For 3D processing, we used the Point Cloud Library (PCL) on the host PC along with 64-bit versions of Boost, OpenCV, and Protobuf libraries. PCL is a large scale, open project for point cloud processing. It contains modules for RGB-D filtering, reconstruction, and visualization.

3.3. SERVER SIDE NETWORKING

In this section, we explain the server side data capturing implementation details. As the data is streamed from client as a proto file, we created a network listener which accepts connections and parses proto files to objects. To make the networking part and processing part independent, we created a mutex controlled list which contains the parsed information. Network process parses proto file and insert the necessary information to shared list. Then, it notifies the 3D processor thread to process incoming data. This way, we multi-parallelized capturing and processing parts. It allows us to capture from multiple clients, fast capturing capability and elasticity. This scheme is given in Fig. 3.4.

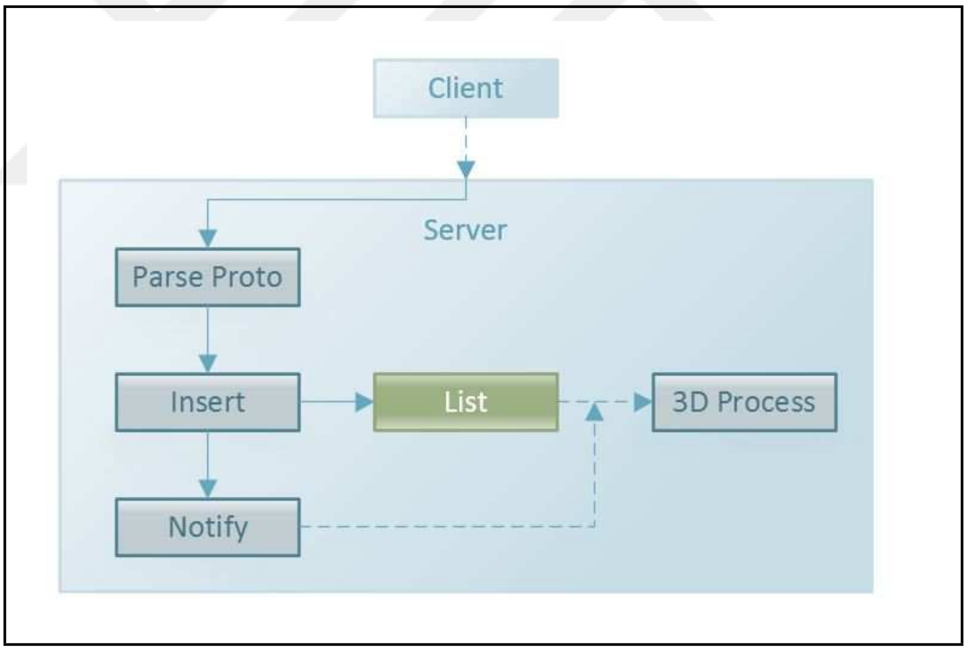


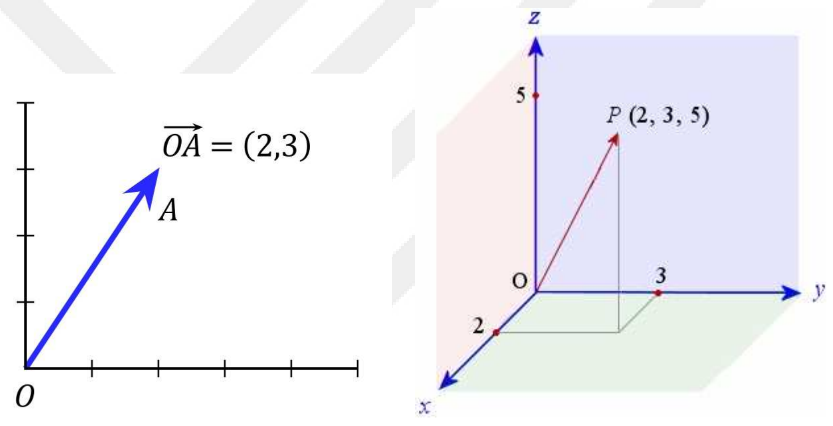
Figure 3.4. Block diagram representation of the server side

4. MATHEMATICAL METHODS USED IN THE SYSTEM

In this chapter, we explain the mathematical methods in our system. These are surface normal extraction, RANSAC plane extraction, and RGB-D registration.

4.1. VECTORS

A Euclidean vector is a geometric quantity which has a magnitude and direction expressed numerically as [x,y] in two dimensions and as [x,y,z] in three dimensions. It is defined in Cartesian domain and it is represented as an arrow starting from origin point of Cartesian coordinates (0,0). In the Fig. 4.1, 2D and 3D vector representations can be found.



(a) 2D vector representation (b) 3D vector representation

Figure 4.1. Vectors

4.1.1. Orthogonality

Dot product of two vectors is defined as $A \cdot B = ||A||||B|| \cos\theta$ where θ indicates the angle between two vectors and $||A||$ and $||B||$ indicate the magnitude of related vectors.

As $\cos\theta = 0$ when the angle between two vectors (θ) is equal to $\frac{\pi}{2} + \pi k$ where $k \in \mathbb{Z}$, two vectors are perpendicular (or orthogonal). Then, their dot product equals to 0.

4.1.2. Parallelism

Cross product of two vectors is defined as $A \times B = ||A||||B|| \sin\theta n$, where θ indicates the angle between two vectors, $||A||$ and $||B||$ indicate the magnitude of related vectors and n indicates the direction of cross product according to right hand rule [32].

As $\sin\theta = 0$ when the angle between two vectors (θ) is equal to $\pi + \pi k$ where $k \in \mathbb{Z}$, two vectors are parallel. Then, their cross product equals to 0. We will use these properties to group the plane's points and to distinguish the orthogonal planes in 3D point cloud.

4.2. SURFACE NORMALS

Surface normal, $\vec{n}$, is the vector which is perpendicular to the plane represented. In concave or convex surfaces, the surface normal can be represented by obtaining the tangent plane of the surface point. There are always two normal vectors at a point which has π radian angle difference to each other. We will indicate vector and plane representation in point cloud.

If we represent surface plane as $ax + by + cz + d = 0$, then $\vec{n_1} = (a, b, c)$ and $\vec{n_2} = (-a, -b, -c)$ are the normal vectors of this plane. In Fig. 4.2, the surface normals of a plane is given where $\vec{n_1}$ and $\vec{n_2}$ are directed in opposite directions.

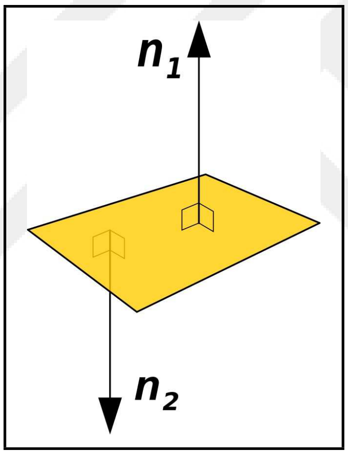


Figure 4.2. Normal vectors of a surface

Defining surface normals in multi-planed shapes has benefits. In this thesis, we will use them to track plane continuity, to group surfaces, to obtain the average surface normal vector, to eliminate small measurement errors and to use in RANSAC process.

4.3. POINT CLOUD

Point cloud is a set of points which are defined in a coordinate system. In this thesis, we will use four dimensional points which will have scalar values in [x,y,z] directions and an RGB value for color information. It is provided by the 3D scanner. According to the scanner's sensitivity, the depth data in 3D (and optionally the color data) is obtained discretely and mapped in a coordinate system. The depth data is first formed as a 2D depth image which contains object-to-camera distance at each of its pixels. It can be represented as a disparity map where two or more images of the same scene is taken from different but pre-known positions. These images are combined to create which pixel has the minimum change along with the maximum. As the nearest pixels will have the maximum change in the same scene and the farmost ones will have the minimum, the proportional disparity maps can be created. The scanned values don't contain any geometric shape like plane or sphere. In Fig. 4.3, a scene and its depth image is provided. In this figure, grayscale coloring is applied such that bright areas are closer.

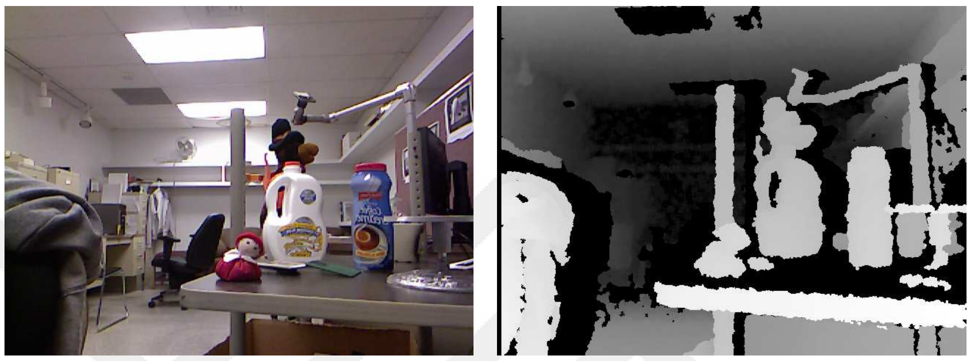

(a) Original image. (b) Depth map.

Figure 4.3. Depth image of a scene

RGB-D refers to the concept of 3D point clouds with a color information at each point. Normally, point clouds don't contain any color information as they are derived from a 2D depth images. It is usually described as a pair of RGB image and depth image. That's why, the image set in Fig. 4.3 becomes RGB-D image. The obtained points can be used for fast 3D geometry processing along with texturing and creating surfaces by using normal vectors of point cloud. These surfaces can represent a 3D Model for different usages including CAD and computer graphics.

4.3.1. RGB-D Point Cloud Creation

In this section, we explain the 2D to 3D transformation. After obtained uncompressed depth and RGB images, we converted them to 3D RGB-D data by using

$$P_x = D_{(x,y)} \frac{(x - \frac{R_x}{2})}{f} \tag{4.1}$$

$$P_y = D_{(x,y)} \frac{(y - \frac{R_y}{2})}{f} \tag{4.2}$$

$$P_z = D_{(x,y)} \tag{4.3}$$

$$P_{rgb} = C_{(x,y)} \tag{4.4}$$

where P is the point in 3D, R is the dimension of 2D image, $C_{(x,y)}$ is the color pixel at location (x, y) and f is the focal length of depth camera which is 580 for Kinect [33].

4.3.2. Rotation

Rotation of point clouds can be handled by a transformation matrix. It has a different format for each of axis. $R_x(\theta), R_y(\theta), R_z(\theta)$ represents the rotation matrix among x,y,z axes for θ degree. These are defined by

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\theta) & -sin(\theta) \\ 0 & sin(\theta) & cos(\theta) \end{bmatrix} \tag{4.5}$$

$$R_y(\theta) = \begin{bmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{bmatrix} \tag{4.6}$$

$$R_z(\theta) = \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.7}$$

If there is a multi axis rotation with different angles, we can use matrix multiplication to find the necessary transformation matrix as defined as $R = R_x(\gamma)\, R_y(\beta)\, R_z(\alpha)$ where γ, β, α are the Euler angles in x,y,z directions respectively.

4.3.3. Determining Surface Normals

Determining surface normals, which define planar surfaces and their geometry properties, is the key process for our system. By estimating them, we can group, intersect and extract planar surfaces. Surface normals can be obtained by two methods. In the first method, the triangulation can be used to create rough surfaces. Then, normals of these surfaces can be calculated. They can be averaged using

$$\overrightarrow{n_p} = \frac{1}{n}\sum_{i=1}^{n}\overrightarrow{n_i} \tag{4.8}$$

to calculate the surface normal on a point where $\overrightarrow{n_p}$ is the surface normal existing on point's location and $\overrightarrow{n_i}, i \in \mathbb{N}$ are the normal vector set of surfaces which are connected to the point.

In second method, which we have implemented in our work, frames the estimating surface normals as a problem of analyzing eigenvectors and eigenvalues of a covariance matrix created from the nearest neighbors of the query point [34]. This is called as Principal Component Analysis (PCA). Detailed information on PCA can be found in the next section.

4.3.4. Normal Estimation with PCA

The neighbor points P^q of a point p, can be used to estimate the local features of the geometry which is represented by the point group P^q. We will estimate the surface normals as local features. We will use the first order 3D plane fitting as proposed by Berkmann *et al* [35]. The problem of surface normal estimation on a point can be handled in two main parts: Estimating the plane tangent at point p to the surface of the geometry formed by point cloud group P^q and calculating the surface normal of this plane at point p. Then it becomes a least-square plane fitting estimation problem in point cloud group P^q [36].

As we are handling this case as least-square fitting problem, we accept the distance from the point p to plane d as 0. $d_i = 0 = (p_i - x)\overrightarrow{n}$. This way, we can calculate x and $\overrightarrow{n}$. As we know all the point's coordinates in P^q, we can obtain the center of this geometry by averaging all the points as

$$x = \frac{1}{k}\sum_{i=1}^{k} p_i \tag{4.9}$$

where x is the center of geometry and $p_i \in P^q$. This way, we can handle this case as a least-square fitting problem.

By analyzing the covariance matrix $C \in \mathbb{R}^{3\times3}$ and its eigenvalues and eigenvectors, we can obtain the surface normal $\overrightarrow{n}$ at point p. The solution for $\overrightarrow{n}$ can be obtained by solving the covariance matrix C.

In Eqns. 4.10 and 4.11, the covariance matrix and eigenvalues and eigenvectors are give. Here, ψ represents the possible weight for p_i and is accepted as 1. C is symmetric, positive semi-definite, and its eigenvalues are real numbers $\lambda_i \in \mathbb{R}$.

$$C = \frac{1}{k}\sum_{i=1}^{k} \psi(p - x) \cdot (p - x)^T \tag{4.10}$$

$$C \cdot \overrightarrow{\mathrm{v}_j} = \lambda_i \cdot \overrightarrow{\mathrm{v}_j}, j \in 0, 1, 2 \tag{4.11}$$

If $0 \leq \lambda_0 \leq \lambda_1 \leq \lambda_2$, the eigenvector with smallest eigenvalue λ_0 is an approximation of $+\overrightarrow{n}$ or $-\overrightarrow{n}$. $\overrightarrow{n} = \{n_x, n_y, n_z\}$ can be represented as a pair of angles (Φ,θ) in spherical coordinates as [37]

$$\Phi = \arctan\left(\frac{n_z}{n_y}\right), \Phi = \arctan\left(\frac{\sqrt{(n_y^2 + n_z^2)}}{n_x}\right) \tag{4.12}$$

Using two surface normals with π angles, it is impossible to solve the sign of $\overrightarrow{n}$. That's why viewpoint is also used. As the scene has to be captured from a viewpoint, this information can be used [34]. Here, the opposite sided normals are filtered using $\overrightarrow{n_i} \cdot (v_p - v_i) > 0$.

4.4. RANSAC

The RAndom SAmple Consensus (RANSAC) Algorithm is proposed by Fischler and Bolles [38]. It is an iterative method to estimate the parameters of a mathematical model. The algorithm classifies the input data as inliers and outliers. As there is various sample consensus algorithm, we will use RANSAC. Raguram *et al* [39] proposed a comparative analysis of RANSAC techniques. They classify and compare the optimization techniques which are $T_{d,d}$ test, Bail-Out test, WaldSAC, PROSAC, MLESAC, Lo-RANSAC and the variants of RANSAC which are Preemptive RANSAC, Adaptive Real-Time Random Sample Consensus (ARRSAC), and regular RANSAC. The solutions are handled in four criteria: inliers found, number of models initiated, the number of verification per model and relative speed up obtained related to regular RANSAC. Even though the performance can be increased significantly in 2D RGB images by using ARRSAC, the implementation time for 3D RGB-D data would be long. Due to time limitations of this work and known performance of RANSAC, we prefer to use it.

4.4.1. Random Sample Consensus

The goal of RANSAC is to find inlier model parameters θ which maximize a cost function $JS(\theta, C_{in}, \phi)$ from input point cloud C_{in} where the threshold value ϕ is the input parameter to RANSAC. In standard notation, JS denotes the number of inliers in a given point cloud C_{in}. In a simpler manner, RANSAC is in a search of the maximum points consistent with the plane which parameter space is θ. The error function $e(x, \theta)$ represents the distance of a data point to a model. RANSAC algorithm mainly composed of two steps which is repeated iteratively:

Hypothesize

Minimal Sample Sets(MSS) are randomly selected from input dataset. The model parameters (which will be explained) are calculated only using the elements of MSS. The smallest cardinality is sufficient to determine model parameters.

Test

Atthis step, RANSAC checks which elements of entire dataset is related with the model parameters calculated at first step. The set of such elements is called Consensus Set (CS).

This process is terminated when the probability to find a better CS drops below a certain threshold. As it is an iterative process, the calculation speed is determined inverse proportional to the point cloud size, hence the iteration count. In the following sections, we explain *Hypothesize* and its *Test* conditions.

4.4.1.1. *Iteration Count.*

As RANSAC iteratively and randomly takes minimal sample sets (MSS), examining every single match option would take enormous time and process power. Hence, the algorithm stops when the probability p to find model whose inlier point count is larger than I_{k-1} in k^{th} iteration, falls under the threshold ϕ. As the probability to find a better model is p, the probability to choose a MSS containing at least one outlier is $1-p$. If we construct h different MSSs, this probability will be denoted as $(1-p)^h$. This function converges to zero. This guarantees that a good pick can be made. So, the iteration count h has to be chosen large enough such that the probability $(1-p)^h$ is smaller or equal to given threshold ϕ, $(1-p)^h \leq \phi$. This function can be inverted as $h \geq \frac{\log \phi}{\log(1-p)}$.

In this work, as we aim the maximum calculation speed. Therefore, we selected $h = 0.50$. This threshold value prevents the long computation time while providing an adequate output.

4.4.1.2. *MSS Creation and Probability Calculation.*

Theoretically, by using a noise-free point cloud, all elements in the dataset have the same probability to be selected. So, the probability to obtain a MSS composed only of inliers is denoted as

$$p = \frac{\binom{C_I}{k}}{\binom{C}{k}} = \frac{C_I!(C-k)!}{C!(C_I-k)!} = \prod_{i=0}^{k-1} \frac{C_I - i}{C - i} \tag{4.13}$$

where C_I is the total number of inliers in a point cloud C and k is the cardinality of C_I.

If $N, N_I \gg k$, then p is approximately equivalent to the probability of picking for k times with re-insertion an inlier from the dataset as

$$p = \prod_{i=0}^{k-1} \frac{C_I - i}{C - i} \approx \left(\frac{N_I}{N}\right)^k \tag{4.14}$$

This equation contains a dependency to N_I which is generally not known, to calculate probability p. To eliminate it, we can verify that for any $\hat{\mathrm{C}}_I \leq C_I$, there exists $p(\hat{\mathrm{C}}_I) \leq p(C_I)$

and $(1 - p(C_I))^h \geq (1 - p(\hat{\text{C}}_I))^h$ where $\hat{\text{C}}_I$ is the largest set of inliers found so far. Hence, by using the maximum number of iteration parameter and $h \geq \frac{\log \phi}{\log(1-p)}$, we can summarize the formula as

$$h = \frac{\log \phi}{\log(1 - p(\hat{C}_I))} \tag{4.15}$$

In original RANSAC, ranking the Consensus Sets are nothing but its cardinality. That guarantees us to have the biggest plane as a result each time.

4.4.2. Plane Finding using RANSAC

As mentioned before, we used RANSAC to determine the planes. In a standard room model, there are four walls, one floor and one ceiling forming a cubical or rectangular prismatic shape. So, our captured frames may contain all of these elements. We calculated surface normals of the input cloud. Then, we separated the cloud points according to their normals. The separation is made according to the angle between surface normal vector and unit vectors in z direction.

4.4.2.1. *Normal based Separation.*

In our algorithm, we calculate all the surface normals by using k neighbor points. As we don't pre-process the cloud, it contains noise which is caused by the transformation and the capture device. To eliminate it, we use a bigger neighbor number, $k = 40$, to compute a point's surface normal. If we use a bigger number, we are losing details and it causes us to lose data on small perpendicular surfaces while increasing the already-known surface's point count by affecting like a low pass surface normal filter. If we use a smaller number, the details become more effective and the noise on separated surfaces increases.

After obtaining surface normals, we classify them according to the angle between viewpoint's unit vector $\vec{u} = (0, 0, 1)$ and surface normal vector $\vec{n}$. To calculate this angle, we use

$$\alpha = \arccos\left(\frac{\vec{u} \cdot \vec{v}}{\sqrt{||\vec{u}||\,||\vec{v}||}}\right)\frac{180}{\pi} \tag{4.16}$$

where α represents the angle between two vectors $\vec{u}$ and $\vec{v}$, $||\vec{u}||$ and $||\vec{v}||$ represent the squared norms of vectors $\vec{u}$ and $\vec{v}$.

After obtaining the angle, we decide if the surface normal vector is likely to be on the z-axis or x/y axes. If the angle α can be defined as $\frac{7\pi}{4}+2k\pi \leq \alpha \leq \frac{\pi}{4}+2k\pi$, it means that it is more likely to be on z axis. So, we classify it as the first iteration. If, the angle α can be defined as $\frac{\pi}{4}+2k\pi < \alpha < \frac{3\pi}{4}+2k\pi$, it means that it is more likely to be on y or x axis. Based on x axis, if x component of the normal vector is greater than 0, it is classified as a right planar surface, second iteration, else, it is classified as a left planar surface, third iteration.

To prepare the point cloud to RANSAC, the outlier filter is applied. The filter searches for the k neighbor points in a given radius r. If there are not enough points in the given radius, the point is deleted from the point cloud. For practical usage, $k = 5$ and $r = 10cm$ are used to eliminate small point groups which may cause a wrong RANSAC output. Then, RANSAC is applied to each of the sub point clouds. We check the output of RANSAC as explained in the next section. In case of success, the point cloud is kept for the next process. Otherwise, it is deleted.

4.4.2.2. *Checking Eligibility.*

We find the minimum rectangle prism that encapsulates the plane point cloud by finding the maximum points of it in (x, y, z) directions. If this plane is not the first one, the process continues. The plane has to intersect with the first plane. We compare the density of the rectangle prism. The density function is given as

$$\frac{(|x^{\xi_0}_{max} - x^{\xi_0}_{min}|)(|y^{\xi_0}_{max} - y^{\xi_0}_{min}|)(|z^{\xi_0}_{max} - z^{\xi_0}_{min}|)}{C_{\xi_n}} \tag{4.17}$$

where C_{ξ_n} is the point count for plane.

In some cases, RANSAC may give us wrong results as given in Fig. 4.4. To eliminate it, we compare the density of points that lie inside its boundary rectangle. We decided to use 120 as threshold for this application. This value is based on experiments. We found that with the lesser count, the noise and wrong RANSAC result increases. With the higher count, the true detections are eliminated.

4.5. SURF FOR RGB-D REGISTRATION

As mentioned before, we capture RGB and depth images and convert them to RGB-D point clouds. As we capture many samples, the registration of these planes are needed. Even though ICP is a good method to merge two point clouds, its process time is heavily dependent

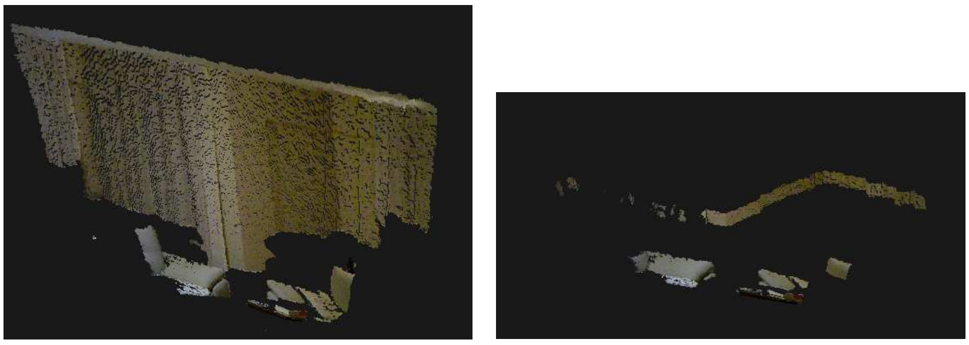

(a) Original point cloud. (b) RANSAC result.

Figure 4.4. Wrong RANSAC result

on point cloud sizes [40]. As we process RGB-D images with a maximum of 307200 points (VGA Resolution), merging two clouds are taking excessive times in an iterative manner. Also, ICP may not correctly estimate the transformation matrix when the input point clouds contain errors and noise. To eliminate it and the processing time handicap, we used 2D RGB images of related point clouds. We captured two consecutive images with θ angle between them from a stationary platform. We found SURF descriptors in each image and matched them. We triangulated the three points: The 3D viewpoint coordinate and the coordinates of the 3D points that SURF descriptor exist and match for new and old RGB images. Then, we find the rotation angle. By this method, we can find the rotation angle in a fast manner. By using the rotation matrix which is explained in Section 4.3.2, we calculate the actual position of the point cloud.

SURF (Speeded Up Robust Features) is a robust feature detector algorithm which is proposed by Bay *et al.* [41]. It is a speeded up version of Scale Invariant Feature Transform (SIFT) [42]. The aim of SURF is to find the correspondence in images. The work is divided in three steps: finding interest points like blobs, corners and T-junctions; finding the neighborhood of the interest points and representing it as feature vectors; matching the similar vectors in two images. The matching process is mainly based on Euclidean distance between two vectors.

The SURF descriptor is based on similar properties and consists of two steps. The first step is to fix a reproducible orientation based on information from a circular region whose radius is determined by user input around the interest point. Then, a construction of a square region aligned to the selected orientation is made to extract the SURF descriptor. Next, we will explain our usage of SURF for registration, Hessian Matrix, Orientation Assignment, Descriptor Components and FLANN matching algorithm.

4.5.1. Hessian Matrix

The calculation process of SURF is based on Hessian Matrix as it gives good performance in computation time and accuracy. Given a point $p = (x, y)$ in an image I, the Hessian matrix $H(p, \sigma)$ in p at scale σ is defined as

$$H(p,\sigma) = \begin{bmatrix} L_{xx}(p,\sigma) & L_{xy}(p,\sigma) \\ L_{xy}(p,\sigma) & L_{yy}(p,\sigma) \end{bmatrix} \tag{4.18}$$

where $L_{xx}(p,\sigma), L_{xy}(p,\sigma), L_{yy}(p,\sigma)$ are the convolutions of the Gaussian second order derivative $\partial p^2 g(\sigma)$ with the image I in point p. Gaussian filters are approximated with box filters by discretizing them as shown in Fig. 4.5 where the gray regions are equal to zero.

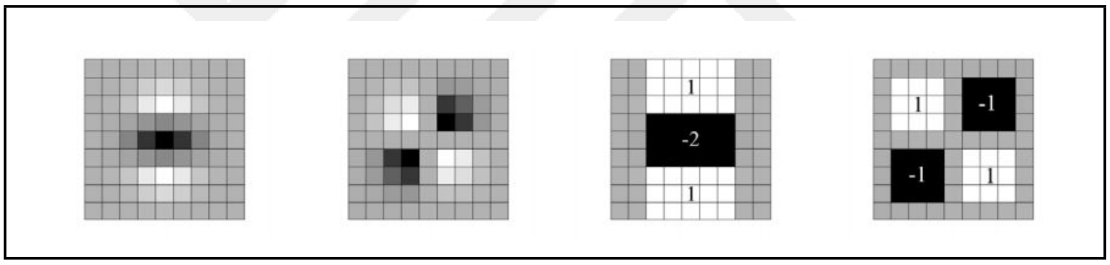


Figure 4.5. Gaussian second order partial derivatives in y-direction and xy-direction (left two), and SURF approximations of using box filters(right two)

The 9×9 box filters in Fig. 4.5 are approximations of Gaussian second order derivatives with $\sigma = 1.2$ and represent the lowest scale. D_{xx}, D_{yy}, and D_{xy} are the denotations of the approximations. The relative weights in the expression are balanced for the Hessian's determinant by

$$\frac{\left|L_{xy}(1.2)\right|_F \left|D_{xx}(9)\right|_F}{\left|L_{xx}(1.2)\right|_F \left|D_{xy}(9)\right|_F} = 0.912 \cong 0.9 \tag{4.19}$$

where $\left|p\right|_F$ is the Frobenius norm and which will lead to

$$\left|H_{approx}\right| = D_{xx}D_{yy} - (0.9D_{xy})^2 \tag{4.20}$$

By this process, the filter responses are normalized with respect to the mask size. It creates a constant Frobenius norm for any filter size. It also allows us to apply Gaussian filters which are being doubled by size for each step. This process creates an image pyramid. To obtain and to localize the interest points, a non-maximum suppression in a $3 \times 3 \times 3$ neighborhood is applied. The maxima of the determinant of the Hessian matrix are then interpolated in scale and image space with the method as proposed by Brown *et al* [43].

4.5.2. Orientation Assignment

In this step, a reproducible orientation for the interest points is identified by calculating the Haar-wavelet responses in both x and y directions and by circulating neighborhood of radius $6s$ around the interest point where s is the scale at which the interest point was detected. To preserve the ratio, wavelet responses are calculated in the current scale. To prevent the long computation times, as the big scaled wavelet's sizes are big, integral images are used. Hence, only six operations are needed to compute the response in x or y direction at any scale.

Then, the responses are weighted with a Gaussian $(\sigma = 2.5s)$ centered at the interest point. The responses are represented as vectors in a space with the horizontal response strength along the abscissa and the vertical response strength along the ordinate. To estimate the dominant orientation, the sum of all responses within a sliding orientation window covering an angle of $\frac{\pi}{3}$ are calculated. The horizontal and vertical responses in the orientation window are summed to yield a new vector. The longest of these vectors lends its orientation to the interest point. The size of the sliding window is a parameter which leads the exposition of the single dominating wavelet responses when it is small and which leads the exposition of the apparent ones.

4.5.3. Descriptor Components

To extract the descriptor, a square region which is centered around the interest point and which is oriented according to the orientation obtained in Section 4.5.2 is constructed. Then, this constructed region is divided into 4×4 window. For each sub-region, the Haar wavelet responses(d) are calculated in x and y directions (d_x, d_y) with a Gaussian $(\sigma = 3.3s)$. Then, x and y components are summed up for each sub-region to form a first set of entries to the feature vector. The four-dimensional descriptor vector $v = (\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|)$ is obtained by including the polarity by calculating the absolute values.

According to the experiments made by Bay, 3×3 subregions allow very fast matching with an adequate performance while 5×5 and bigger subregions are less robust and they increase computation times too much. Hence, 4×4 subregion matrix is used in this study.

4.5.4. Matching Algorithm

FLANN is an abbreviation of "Fast Library for Approximate Nearest Neighbors". It is introduced by Muja *et al.* [44]. We preferred to use FLANN as it is specially designed for vector-based features. FLANN algorithm is based on the GNAT tree structure [45].

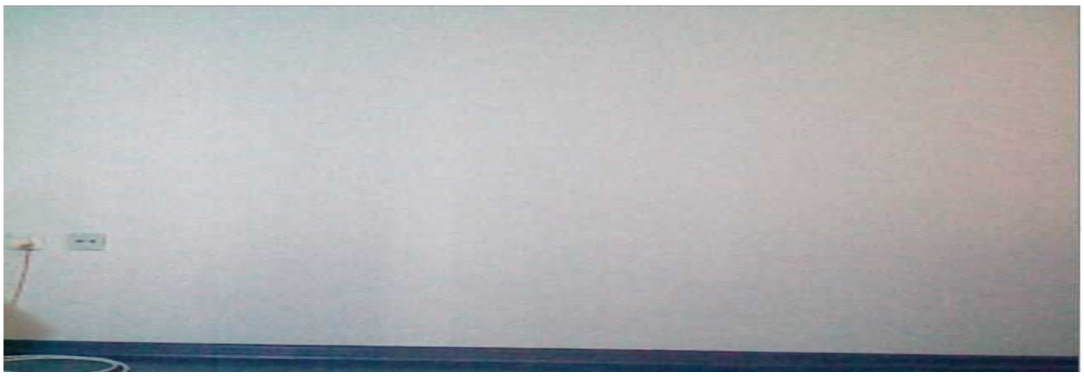
The search ends when the number of points examined exceeds a maximum limit given as a parameter to the search algorithm. This limit specifies the degree of approximation desired from the algorithm. The higher the limit, the more exact neighbors are found, but the search costs more calculation power.

4.5.5. Registration with SURF

As explained above, SURF is a very fast feature detection method. Hence, we aim to use this advantage to increase the speed of the registration process. As we capture the scene images with a stationary camera in horizontal axis with α degree between each of them, we expect that these two images will have common textures in small angles. Therefore our method is based on three main steps. The first step is to find good matches of similar textures between two consecutive RGB images. As the second step, we calculate the angle between two scenes by including the depth image data. As the last step, we refine the results in a similar manner to ICP.

To find the good match of similar textures, we used OpenCV library's SURF detector functions. SURF detector function is applied to estimate the key-points in images separately. Then, SURF descriptor extractor function is applied to estimate the descriptors in images separately. By using FLANN matcher function, similar RGB textures in both scene are estimated. As there are too many matches which can be seen in Fig. 4.6, a refinement needs to be applied.

As the scene images are captured horizontally, the two consecutive image's matched descriptors must have the similar vertical coordinates in the frame. The only error can be caused by the focal properties of the camera. That's why we used a parameter to filter the maximum vertical coordinate difference between matched pairs (o_k, s_k) as


(a) Original image

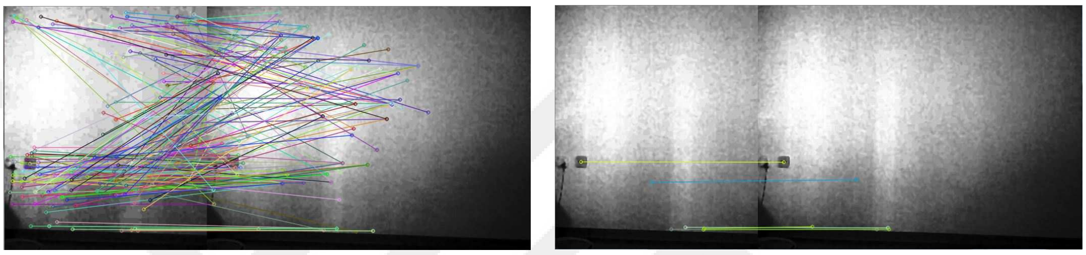

(b) Unfiltered matches on grayscale image. (c) Filtered matches on grayscale image.

Figure 4.6. FLANN matching example

$$\tau < |o_{ky} - s_{ky}| \tag{4.21}$$
$$d(o_k, s_k) > 2min(d(o_k, s_k)) \tag{4.22}$$

where τ is the error parameter and o_{ky}, s_{ky} are the vertical coordinates of matched pairs. Also, a distance based filter is applied. As the matched pair's Euclidean distance in means of coordinate in related frames must be similar, we first calculated the minimum distance between pairs ($min(d(o_k, s_k))$) and filtered the matches which distance is smaller than $min(d(o_k, s_k))$.

As good matches are obtained, the next process is to estimate the turn angle α between images. Here, the aim is to create a 3D triangle (ABC) where each corner corresponds to a 3D coordinate in point cloud. As we have 2D (x,y) position of matched points, and as each RGB + Depth image pair is representing a 3D real world coordinate in two 2D image, we use these values to calculate z parameters of ABC triangle. In this representation, A is indicating the viewpoint coordinate which is always equal to $(0, 0, 0)$ as the viewpoint coordinate is not moving in this work. B is the point which is a 3D projection of a SURF descriptor found on previous frame's RGB image. To calculate this projection, we reused Eqn. 4.1 for each of the pixels which coordinates are obtained via SURF. By using this process, the 3D coordinates of B point are obtained in a manner of (x, y, z). C is the point which is also a 3D projection of a SURF descriptor found on current frame's RGB image. To calculate this projection, we reused Eqn. 4.1 for each of the pixels which coordinates are obtained via SURF.

As ABC triangle's each point's real world coordinates are calculated, by using the general triangulation formula, the turn angle is calculated as

$$d(v_p, o_k) = \sqrt{(v_{px} - o_{1x})^2 + (v_{py} - o_{1y})^2 + (v_{pz} - o_{1z})^2} \tag{4.23}$$

$$d(v_p, s_k) = \sqrt{(v_{px} - s_{1x})^2 + (v_{py} - s_{1y})^2 + (v_{pz} - o_{1z})^2} \tag{4.24}$$

$$d(o_k, s_k) = \sqrt{(o_{1x} - s_{1x})^2 + (o_{1y} - s_{1y})^2 + (o_{1z} - s_{1z})^2} \tag{4.25}$$

$$\alpha_k = \frac{360}{\pi} \cos\left(\frac{d(v_p, o_k)^2 + d(v_p, s_k)^2 - d(o_k, s_k)^2}{2d(v_p, o_k)d(v_p, s_k)}\right) \tag{4.26}$$

$$\alpha = \frac{\sum_0^k \alpha_k}{k} \tag{4.27}$$

where $d(o_k, s_k), d(v_p, o_k), d(v_p, s_k)$ indicate the Euclidean distance between related points. As all angles are obtained for all good matches, arithmetic average is taken to eliminate the calculation error where $k \in \mathbb{N}$ is the total number of good matches. The angles are cumulative which means that each frame taken is cumulatively dependent on the previous frame's angles.

5. RESULTS

In this chapter, we first introduce experiment results. Then, we discuss the error percentages and their reasons. Finally, we discuss our system's effectiveness, strong and weak sides. In discussion, the main two criteria are extracting planes and walls correctly and merging the scenes successfully. All the images are taken once in the same angle and the platform is turned. As outlier removal and RANSAC is applied to the point cloud, the holes can be seen as noise. These holes can be filled up correctly by taking more samples at the same angle.

5.1. EXPERIMENT RESULTS

We use image sequences from fourteen different scenes which are separated as *simple scenes* and *complex scenes* in experiments to measure performance in real time. Simple scenes include floor, ceiling, and wall. Complex scenes include floor, ceiling, wall, furniture and other objects. Images are taken as described in previous chapters. We captured several images and merged them. We summarized performance tests as simple and complex scenes in Table 5.1.

Table 5.1. Simple and complex test scenes

Scene	Type	Contents
1	Simple	One wall
2	Simple	Two walls
3	Simple	Two walls, one floor, closed door
4	Simple	Two walls, one floor, open door
5	Simple	Two walls, one ceiling, one floor, closed windows
6	Simple	Two walls, one ceiling, one floor, open windows
7	Simple	Three walls, one ceiling
8	Complex	One wall, one closet with stuffs
9	Complex	One wall, one closed door, two closets
10	Complex	One wall, one bed, one desk with stuffs
11	Complex	Two walls, one floor, one sunshade
12	Complex	Two walls, one TV, one closet
13	Complex	Two walls, one sofa, one human
14	Complex	Two armchairs, one sunshade, one sofa

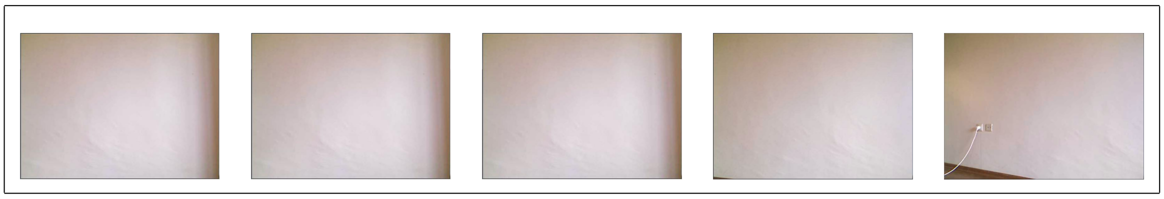
We also summarized accuracy tests in three different scenes which are indicated in Table 5.2. In these tests, we measured the method's accuracy. We applied ICP [40] for accurate merge of the scenes which are roughly merged by using SURF descriptors. The ICP algorithm is applied in MATLAB and plane fitting is applied to output planes for visualization purposes by using PCL. We also provide system measurement error by comparing the real world distance and point cloud's distance.

Table 5.2. Accuracy test scenes

Scene	**Type**	**Contents**
15	Accuracy	One wall, one bed, one desk with stuffs
16	Accuracy	One wall, one closed door, two closets
17	Accuracy	Two walls, one TV, one closet

In our scene's performance results, we provide several results as follows. *3D conversion* timing indicates the duration of 2D to 3D RGB-D conversion. *Normal Calculation* timing indicates the duration of surface normal calculation for single point cloud by using four cores parallelization. *Outline Removal* timing indicates the duration of three sub-plane creation and outline removal over them in total. *RANSAC* timing indicates the duration of RANSAC process of three sub-plane in total. *Turning Cloud* timing indicates the duration of turn angle calculation by using SURF and the duration of turning point cloud. At each scene, we can't apply SURF to the first frame. Therefore, the process timings for these frames are indicated as N/A. We also provide total turn angle and view angle for each scene. The RGB images of scenes are indicated at Table 5.3 and Table 5.4.

Table 5.3. RGB images of scenes 1 to 6

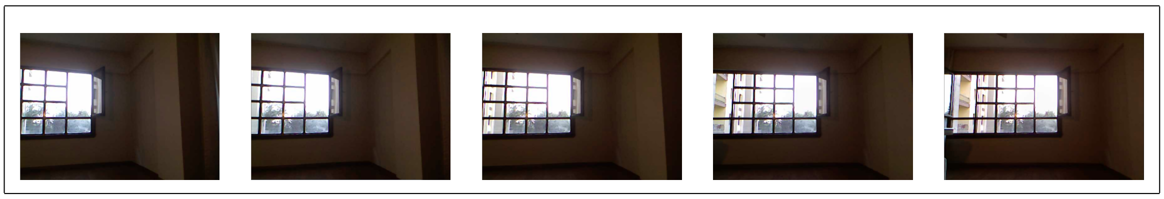

RGB images of scene 1

RGB images of scene 2

RGB images of scene 3

RGB images of scene 4

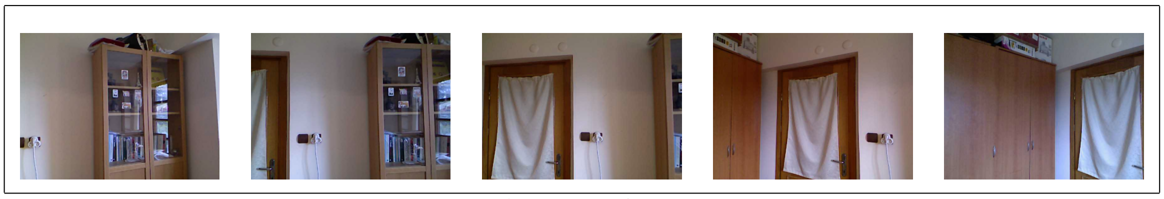
RGB images of scene 5

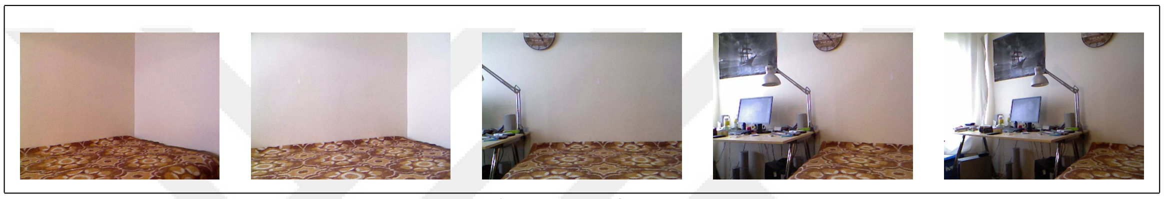

RGB images of scene 6

RGB images of scene 7

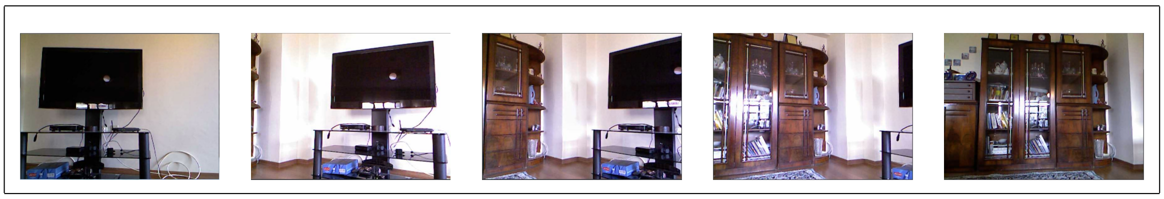
Table 5.4. RGB images of scenes 7 to 14

RGB images of scene 8


RGB images of scene 9


RGB images of scene 10

RGB images of scene 11


RGB images of scene 12

RGB images of scene 13

RGB images of scene 14

5.1.1. Scene 1

In this scene, we experimented the simplest case which has only one wall and nothing else in it. In Fig. A.1, we captured the wall information in roughly 30 degree with three images. Each frame contains only the wall. The last frame only contains a wall plug and a cable connected to it. These extra components are not affecting the process.

In this experiment, the wall is successfully extracted as a plane. However, as our method is based on SURF detection for merging point clouds, and raw wall images do not contain enough information to match. Therefore, the angle cannot be calculated correctly. That's why the merging alignment is made at the same point and all the planes are stowed at the same place. The process timings can be found in Table 5.1.1.

Table 5.5. Computation times (in msec.) for scene 1

Frame	**3D Conv.**	**Normal Calc. (4 cores)**	**Outline Removal**	**RANSAC**	**Turning Cloud**
First	31	390	577	47	N/A
Second	32	374	577	16	47
Third	16	390	593	31	62
Fourth	31	406	592	32	62
Fifth	31	321	608	32	62
Total	**141**	**1881**	**2947**	**158**	**233**

In this test scene, total view angle was 57 degrees. The total time needed to obtain filtered 3D model for this scene is 5360 msec which is equal to 1072 msec per frame in average. Since we couldn't merge frames in this scene, the turn angle for the rotation table is 0 degrees in all frames. Our system was successful in this case in extracting planes, but failed in merging scenes.

5.1.2. Scene 2

In this scene, we experimented the simpler case which has only two walls which are perpendicular to each other.In Fig. A.2, we captured the wall information in roughly 30 degree with three images. Each frame contains only the wall information and ventilation shaft hole cover. This extra component does not affect the process.

In this experiment, the walls are successfully extracted as a plane. Also, the perpendicularity is preserved between walls. However, as our method is based on SURF detection for merging point clouds, and raw wall images do not contain enough information to match, the angle cannot be calculated correctly. That's why, the merging alignment is made at the same point and all the planes are stowed at the same place. The process timings can be found in Table 5.1.2.

Table 5.6. Computation times (in msec.) for scene 2

Frame	**3D Conv.**	**Normal Calc. (4 cores)**	**Outline Removal**	**RANSAC**	**Turning Cloud**
First	32	390	468	78	N/A
Second	32	405	452	110	47
Third	31	406	463	63	62
Total	**95**	**1201**	**1377**	**251**	**109**

In this test scene, total view angle was 57 degrees. The total time needed to obtain filtered 3D model for this scene is 3033 msec which is equal to 1011 msec per frame in average. Since we couldn't merge frames in this scene, the turn angle for the rotation table is 0 degrees in all frames. Our system was successful in this case in extracting planes, but failed in merging scenes.

5.1.3. Scene 3

In this scene, we experimented the a case which has two walls which are perpendicular to each other, one closed door and a floor. In Fig. A.3, we captured the wall information in roughly 45 degrees with five images. Each frame contains a door, walls, a plug and a cable. These extra components are affecting the process. The plug and the door regions are the ones that contain the most matched pairs with the previous frame. Also, luminance value varies from frame to frame. However, this variance does not affect our merging and plane extraction processes. This luminance difference only causes textures to mismatch by color and causes a bad view by color in 3D cloud.

In this experiment, the walls are successfully extracted as a plane. Also, the door is accepted as a wall. Theoretically, it should be a wrong plane detection but the door represents a plane and its plane is convenient with the one that it follows. That's why we did not accept it as a wrong detection. As there are enough point to match, the angle is determined correctly and the scenes are merged successfully. The process timings can be found in Table 5.1.3.

Table 5.7. Computation times (in msec.) for scene 3

Frame	**3D Conv.**	**Normal Calc. (4 cores)**	**Outline Removal**	**RANSAC**	**Turning Cloud**
First	31	390	451	110	N/A
Second	31	406	546	110	47
Third	31	406	422	62	78
Fourth	31	312	452	32	62
Fifth	31	468	452	47	78
Total	**155**	**1982**	**2323**	**361**	**265**

In this test scene, total view angle was 96.12 degrees. The total time needed to obtain filtered 3D model for this scene is 5086 msec which is equal to 1017 msec per frame in average. The turn angle for the rotation table is 39.12 degrees in all frames. Our system was successful in this case in extracting planes, as well as successful in merging scenes.

5.1.4. Scene 4

In this scene, we experimented a case which has two walls perpendicular to each other, one opened door and a floor. In Fig. A.4, we captured the wall information in roughly 45 degrees with five images. Each frame contains a door, walls, a plug and a cable. These extra components are affecting the process. The plug and the door regions are the ones that contain the most matched pairs with the previous frame. Also, luminance varies from frame to frame. However, this variance does not affect our merging and plane extraction processes. This luminance difference only causes the textures to mismatch by color and causes a bad view by color in 3D cloud.

In this experiment, the walls are successfully extracted as a plane. Also, the door is accepted as a wall. Theoretically, it should be a wrong plane detection but the door represents a plane and its plane is convenient with the one that it follows. That's why we did not accept it as a wrong detection. Also, a piece of the far wall is accepted as a plane. It is accepted as a correct detection in the third iteration as there were no other bigger plane convenient to the small angle difference of surface normals because of the open door. As there are enough points to match, the angle is determined correctly and the scenes are merged successfully. The process timings can be found in Table 5.1.4.

Table 5.8. Computation times (in msec.) for scene 4

Frame	**3D Conv.**	**Normal Calc. (4 cores)**	**Outline Removal**	**RANSAC**	**Turning Cloud**
First	32	390	312	140	N/A
Second	31	374	437	94	62
Third	31	390	468	78	78
Fourth	31	390	452	32	62
Fifth	31	390	452	79	62
Total	**156**	**1934**	**2121**	**423**	**264**

In this test scene, total view angle was 92.33 degrees. The total time needed to obtain filtered 3D model for this scene is 4898 msec which is equal to 980 msec per frame in average. The turn angle for the rotation table is 35.33 degrees in all frames. Our system was successful in this case in extracting planes, as well as successful in merging scenes.

5.1.5. Scene 5

In this scene, we experimented a case which has two walls perpendicular to each other, one closed window and a floor. In Fig. A.5, we captured the wall information in roughly 15 degrees with five images. Each frame contains a window and walls. There are no extra components that affect the depth image. But there are some trees at outdoor that can be seen in RGB images. These trees affect the SURF process. However, they have no effect on plane extraction. Because the depth data capturing is not working properly on glass. Besides, the material behind the glass is too far away to detect depth. Therefore the glass area seems empty as it does not contain any point information. Another hard situation in this experiment is that the light source. As outdoor is brighter than the indoor, and the images are taken from counter-light, indoor areas are darker. However, our algorithm is not affected by that effect. Only the texture color mismatch occurred.

In this experiment, the walls are successfully extracted as a plane. Also, the window is accepted as a wall. Theoretically, it should be wrong plane detection but the windows represent a plane which is convenient with the plane of the wall that it follows. That's why we did not accept it as a wrong detection. As there are enough points to match, the angle is determined correctly and the scenes are merged successfully. The process timings can be found in Table 5.1.5.

Table 5.9. Computation times (in msec.) for scene 5

Frame	**3D Conv.**	**Normal Calc. (4 cores)**	**Outline Removal**	**RANSAC**	**Turning Cloud**
First	31	328	357	63	N/A
Second	31	328	327	94	78
Third	31	312	312	78	94
Fourth	31	312	327	32	93
Fifth	31	296	297	47	92
Total	**155**	**1576**	**1620**	**314**	**357**

In this test scene, total view angle was 71.04 degrees. The total time needed to obtain filtered 3D model for this scene is 4022 msec which is equal to 804 msec per frame in average. The turn angle for the rotation table is 14.04 degrees in all frames. Our system was successful in this case in extracting planes, as well as successful in merging scenes.

5.1.6. Scene 6

In this scene, we experimented a case which is similar to the Scene 5. The only difference is the open window. In Fig. A.6, we captured the wall information in roughly 15 degrees with five images. Each frame contains a window and walls. There are no extra components that affect the depth image. But there are some trees at outdoor that can be seen in RGB images. These trees are affecting the SURF process. However, they have no effect on plane extraction as in scene five. However, the opened window affected the merging algorithm. In the second RGB image, open window caused more local luminance. This affected SURF extraction as the unbalanced light suppress local descriptors. That caused a misalignment between first, second and third frames. However, in fourth and fifth frames, the luminance is balanced and more SURF descriptors are obtained. Therefore, fourth and fifth frames are aligned correctly but the first three frames are accepted as a misalignment. This error is also caused by the lack of the capability of the Kinect's RGB camera.

In this experiment, walls are successfully extracted as a plane. Also, the window is accepted as a wall. However, open window affected the luminance. Hence, enough SURF descriptor couldn't be detected. Therefore, the angle is determined falsely for the first three frames and correctly for last two frames. Hence, only the last two frames merged correctly. We accepted this case as a conditional success. The process timings can be found in Table 5.1.6.

Table 5.10. Computation times (in msec.) for scene 6

Frame	**3D Conv.**	**Normal Calc. (4 cores)**	**Outline Removal**	**RANSAC**	**Turning Cloud**
First	31	328	405	78	N/A
Second	31	312	405	63	78
Third	31	312	311	79	94
Fourth	15	328	208	63	94
Fifth	15	281	296	47	109
Total	**123**	**1561**	**1625**	**330**	**375**

In this test scene, total view angle was 64.94 degrees. The total time needed to obtain filtered 3D model for this scene is 4014 msec which is equal to 803 msec per frame in average. The turn angle for the rotation table is 7.94 degrees in all frames. Our system was successful in this case in extracting planes, but conditional successful in merging scenes.

5.1.7. Scene 7

In this scene, we experimented a case which has three walls perpendicular to each other and one ceiling. In Fig. A.7, we captured the wall information in roughly 90 degrees with three images. Each frame contains walls and ceiling information. There are no extra components that affect the depth and RGB images.

In this experiment, walls are successfully extracted as a plane. However, as our method is based on SURF detection for merging point clouds, and raw wall images do not contain enough information to match, the angle cannot be calculated correctly. Therefore, the merging alignment is made at the same point and all the planes are stowed at the same place. The process timings can be found in Table 5.1.7.

Table 5.11. Computation times (in msec.) for scene 7

Frame	3D Conv.	Normal Calc. (4 cores)	Outline Removal	RANSAC	Turning Cloud
First	31	437	468	63	N/A
Second	32	374	483	109	47
Third	16	405	454	92	47
Total	**79**	**1216**	**1405**	**264**	**94**

In this test scene, total view angle was 57 degrees. The total time needed to obtain filtered 3D model for this scene is 3058 msec which is equal to 1019 msec per frame in average. The turn angle for the rotation table is 35.60 degrees in all frames. Our system was successful in this case in extracting planes, but failed in merging scenes.

5.1.8. Scene 8

In this scene, we experimented a case which has one wall and one closet. In Fig. A.8, we captured the wall information in roughly 30 degrees with four images. Each frame contains the wall and closet information.

In this experiment, the surface of the closet is accepted as wall. Theoretically, it is an error. However, this is an expected result as there is not enough information about the wall behind the closet. Hence, it is not counted as an error. The wall information is easily extracted. The merging is based on the stuff which is on top of the closet. This information is filtered in 3D plane extraction. But it served to merge the planes successfully. As the captured frames don't have the same luminance values, the textures are not continuous. In this experiment, the walls and closet are successfully extracted as a plane. The closet is accepted as a planar surface. The process timings can be found in Table 5.1.11.

Table 5.12. Computation times (in msec.) for scene 8

Frame	3D Conv.	Normal Calc. (4 cores)	Outline Removal	RANSAC	Turning Cloud
First	31	422	546	31	N/A
Second	31	359	515	93	62
Third	32	390	515	109	46
Fourth	31	375	483	94	62
Total	**125**	**1546**	**2059**	**327**	**170**

In this test scene, total view angle was 88.57 degrees. The total time needed to obtain filtered 3D model for this scene is 4227 msec which is equal to 1057 msec per frame in average. The turn angle for the rotation table is 31.57 degrees in all frames. Our system was conditionally successful in this case in extracting planes, and successful in merging scenes.

5.1.9. Scene 9

In this scene, we experimented a case which has one wall, one door and two cabinets. In Fig. A.9, we captured frames in roughly 90 degrees with five images. Each frame contains the wall and at least a cabinet.

In this experiment, all the planes are extracted successfully. Also, the front plane of left cabinet is filtered which increases the success rate. The filter is applied to it because it contains many glasses which makes the backplane visible. Hence, background points form the largest planar surface and front plane is filtered. Right cabinet's surface, similarly to scene eight, is accepted as a wall which leads to a conditional success. Also, in each frame, many matching points are found. This increased the merging success ratio in a wide angle. This experiment has one of the widest angle as it can be seen in Table 5.1.9. In this experiment, the walls are successfully extracted as a plane even though the cabinet with glass. Right cabinet's surface is accepted as wall which leads to a conditional success. The process timings can be found in Table 5.1.9.

Table 5.13. Computation times (in msec.) for scene 9

Frame	**3D Conv.**	**Normal Calc. (4 cores)**	**Outline Removal**	**RANSAC**	**Turning Cloud**
First	31	390	577	140	N/A
Second	32	390	592	78	110
Third	31	390	578	77	94
Fourth	16	405	547	62	109
Fifth	16	406	499	31	93
Total	**126**	**1621**	**2793**	**388**	**406**

In this test scene, total view angle was 122.22 degrees. The total time needed to obtain filtered 3D model for this scene is 5334 msec which is equal to 1067 msec per frame in average. The turn angle for the rotation table is 65.22 degrees in all frames. Our system was conditionally successful in this case in extracting planes, and successful in merging scenes.

5.1.10. Scene 10

In this scene, we experimented a case which has one wall, one bed and one desk with stuffs on top of it. In Fig. A.10, we captured frames in roughly 120 degrees with five images. Each frame contains the wall, bed and/or desk information. In this experiment, all the planes are extracted successfully. Also, bed is counted as a plane. As we are in search of walls, we can count it as a conditional success. However, even though the desk is a large planar surface, the stuffs on top of it prevented the plane shape. As a result the desk is filtered. This provided more success rate. Planes are successfully merged.

In this experiment, the walls are successfully extracted as a plane even though the desk's plane which is filtered. The process timings can be found in Table 5.1.10.

Table 5.14. Computation times (in msec.) for scene 10

Frame	**3D Conv.**	**Normal Calc. (4 cores)**	**Outline Removal**	**RANSAC**	**Turning Cloud**
First	15	403	483	63	N/A
Second	31	375	484	46	125
Third	32	374	499	47	125
Fourth	15	359	467	48	125
Fifth	31	359	404	63	141
Total	**124**	**1870**	**2337**	**267**	**516**

In this test scene, the total view angle was 119.65 degrees. The total time needed to obtain filtered 3D model for this scene is 5114 msec which is equal to 1023 msec per frame in average. The turn angle for the rotation table is 62.65 degrees in all frames. Our system was successful in this case in extracting planes, but failed in merging scenes.

5.1.11. Scene 11

In this scene, we experimented a case which has two walls perpendicular to each other, one floor and one sunshade which is the complex scene component. In Fig. A.11, we captured the wall information in roughly 30 degrees with five images. Each frame contains walls, floor and sunshade information.

In this experiment, the part without sunshade is similar to scene two. As sunshade has a curly shape, the surface normals vary too much. Hence, the sunshade area has holes in 3D point cloud. It also served to extract SURF descriptors. As the first two frames contain only one matched pair, the alignment is not successful. However, in next frames, more common points are found because of the sunshade. Therefore, the registration was successful. In this situation, the first frame's point cloud has an alignment error according to the next one. In this experiment, the walls are successfully extracted as a plane. The sunshade has some holes in 3D cloud, based on its shape. It is accepted as a planar surface. The process timings can be found in Table 5.1.11.

In this test scene, total view angle was 85.22 degrees. The total time needed to obtain filtered 3D model for this scene is (157+1950+2305+487+249) 5148 msec which is equal to 1030 msec per frame in average. The turn angle for the rotation table is 28.22 degrees in all

Table 5.15. Computation times (in msec.) for scene 11

Frame	**3D Conv.**	**Normal Calc. (4 cores)**	**Outline Removal**	**RANSAC**	**Turning Cloud**
First	31	406	436	79	N/A
Second	32	390	467	79	46
Third	31	359	467	63	63
Fourth	31	405	452	157	62
Fifth	32	390	483	109	78
Total	**157**	**1950**	**2305**	**487**	**249**

frames. Our system was successful in this case in extracting planes, but failed in merging scenes.

5.1.12. Scene 12

In this scene, we experimented a case which has two walls perpendicular to each other, one TV and one closet. In Fig. A.12, we captured the wall information in roughly 60 degrees with five images. Each frame contains walls, TV and closet information.

In this experiment, the TV unit has been successfully filtered. As the TV has similar properties like glass, the reflection of depth laser was not ordered. Hence, TV area is accepted as a big rectangular hole. It affected RANSAC to extract the correct planar surface. The TV unit also affected SURF by acting as a matching texture. Between each frame enough matching point is found. Hence, a perfect registration is obtained. However, because of the luminance, textures in 3D point cloud could not fit correctly and caused a bad look. Also, small surfaces which lay on corners are detected successfully. The surface of cabinet is accepted as a wall which caused a conditional error as the largest planar surface in fourth and fifth frames is the surface of the cabinet.

In this experiment, the walls are successfully extracted as a plane. Also, obstructions like the TV unit are filtered correctly. The cabinet surface caused error for plane detection. But this is normal as it is a continuous surface plane parallel to wall and it is the largest plane in frames. The frames are merged successfully even in a cornered vision. This experiment shows us the algorithm's success on turns and corners. The process timings can be found in Table 5.1.12.

In this test scene, total view angle was 116.91 degrees. The total time needed to obtain filtered 3D model for this scene is 4883 msec which is equal to 977 msec per frame in average. The turn angle for the rotation table is 59.91 degrees in all frames. Our system was conditionally successful in this case in extracting planes, and successful in merging scenes.

Table 5.16. Computation times (in msec.) for scene 12

Frame	3D Conv.	Normal Calc. (4 cores)	Outline Removal	RANSAC	Turning Cloud
First	32	327	360	77	N/A
Second	16	312	343	140	125
Third	31	296	342	189	109
Fourth	31	359	406	157	124
Fifth	31	374	468	78	156
Total	**141**	**1668**	**1919**	**641**	**514**

5.1.13. Scene 13

In this scene, we experimented a case which has two walls perpendicular to each other, one sofa/bed and a human. In Fig. A.13, we captured the information in roughly 45 degrees with four images. Each frame contains walls, bed and a human in it.

In this experiment, a false detection exists. The walls are successfully extracted as planes as well as human and sofa. Even though the surface normals in human vary, they create a rough planar surface. It caused a false detection of planar surfaces. However, as human contains too much information in the manner of SURF descriptors, the merging of point clouds could be made precisely. The process timings can be found in Table 5.1.13.

Table 5.17. Computation times (in msec.) for scene 13

Frame	3D Conv.	Normal Calc. (4 cores)	Outline Removal	RANSAC	Turning Cloud
First	31	375	452	140	N/A
Second	31	390	469	140	78
Third	32	405	499	141	78
Fourth	31	406	482	111	109
Total	**125**	**1576**	**1902**	**532**	**265**

In this test scene, total view angle was 93.71 degrees. The total time needed to obtain filtered 3D model for this scene is 4400 msec which is equal to 1100 msec per frame in average. The turn angle for the rotation table is 36.71 degrees in all frames. Our system was conditionally successful in this case in extracting planes, and successful in merging scenes.

5.1.14. Scene 14

In this scene, we experimented a case which has two armchairs, one sofa and sunshade. In Fig. A.14, we captured the information in roughly 45 degrees with four images. Each frame contains walls, sofa and sunshade information in it.

In this experiment, armchairs did not affect the result as they are small compared to sofa and sunshades. As the surface normals in sunshades are not linear, in first two frames the sofa is accepted as a wall. This caused an error. In the next frames, the proportion of sunshade area is bigger than the sofa's area. Therefore, the noisy sunshade area is accepted as a wall. As sofa, armchairs and sunshades contain many descriptors, perfect alignment is made. The process timings can be found in Table 5.1.14.

Table 5.18. Computation times (in msec.) for scene 14

Frame	3D Conv.	Normal Calc. (4 cores)	Outline Removal	RANSAC	Turning Cloud
First	31	374	453	125	N/A
Second	32	405	468	109	78
Third	31	406	452	62	78
Fourth	31	437	436	157	78
Total	**125**	**1622**	**1809**	**454**	**234**

In this test scene, total view angle was 93.28 degrees. The total time needed to obtain filtered 3D model for this scene is 4244 msec which is equal to 1061 msec per frame in average. The turn angle for the rotation table is 36.28 degrees in all frames. Our system failed in this case in extracting planes, but successfully merged scenes.

5.1.15. Scene 15

In this scene, we experimented a case which has one wall, one bed and one desk with stuffs on top of it. In Fig. A.15, we captured frames in roughly 120 degrees with five images. Each frame contains the wall, bed and/or desk information.

In this experiment, we measured Hausdorff Distance [46] between point clouds for error calculation. We summarized results as

Table 5.19. Accuracy information for scene 15

Frame	Minimum Hausdorff	Maximum Hausdorff	RMS	Mean
First-Second (Rough)	0.10 mm	132.40 mm	40.75 mm	30.91 mm
Second-Third (Rough)	0.01 mm	132.40 mm	33.52 mm	22.55 mm
First-Second (Fine)	0.08 mm	125.62 mm	26.77 mm	14.70 mm
Second-Third (Fine)	0.05 mm	125.60 mm	30.36 mm	17.34 mm

These values show that, rough alignment via SURF contains nearly double the error than fine alignment via ICP. By applying fine alignment, maximum error difference is decreased, hence, RMS and mean errors are decreased too.

Also, in this experiment, we measured the real world distance of a specific point in each frame by using a regular meter. The difference between real world distance and fine registered point cloud's xyz coordinate is 2.54 mm for first frame, 2.01 mm for second frame and 2.25 mm for third frame. It shows that the error is increased because of SURF and ICP alignment applied to point cloud while accepting the first point cloud stationary. However, if the distance is around two meters, the error is minimized.

5.1.16. Scene 16

In this scene, we experimented a case which has one wall, one door and two cabinets. In Fig. A.16, we captured frames in roughly 90 degrees with four images. Each frame contains the wall and at least a cabinet.

In this experiment, we measured Hausdorff Distance between point clouds for error calculation. We summarized results as

Table 5.20. Accuracy information for scene 16

Frame	Minimum Hausdorff	Maximum Hausdorff	RMS	Mean
First-Second (Rough)	0.09 mm	106.94 mm	29.52 mm	21.40 mm
Second-Third (Rough)	0.24 mm	106.93 mm	19.16 mm	14.67 mm
Third-Fourth (Rough)	0.49 mm	106.93 mm	42.71 mm	39.31 mm
First-Second (Fine)	0.05 mm	128.45 mm	22.74 mm	12.40 mm
Second-Third (Fine)	0.07 mm	128.99 mm	20.83 mm	11.08 mm
Third-Fourth (Fine)	0.10 mm	128.99 mm	22.48 mm	11.58 mm

These values show that, rough alignment via SURF contains nearly double to four times the error than fine alignment via ICP. By applying fine alignment, maximum error difference is decreased, hence, RMS and mean errors are decreased too.

Also, in this experiment, we measured the real world distance of a specific point in each frame by using a regular meter. The difference between real world distance and fine registered point cloud's xyz coordinate is 2.31 mm for first frame, 3.10 mm for second frame and 2.89 mm for third frame. It shows that the error is increased because of SURF and ICP alignment applied to point cloud while accepting the first point cloud stationary. However, if the distance is around two meters, the error is minimized.

5.1.17. Scene 17

In this scene, we experimented a case which has two walls perpendicular to each other, one TV and one closet. In Fig. A.17, we captured the wall information in roughly 60 degrees with four images. Each frame contains walls, TV and closet information.

In this experiment, we measured Hausdorff Distance between point clouds for error calculation. We summarized results as

Table 5.21. Accuracy information for scene 17

Frame	Minimum Hausdorff	Maximum Hausdorff	RMS	Mean
First-Second (Rough)	0.20 mm	135.78 mm	29.53 mm	21.01 mm
Second-Third (Rough)	0.08 mm	135.76 mm	23.73 mm	16.07 mm
Third-Fourth (Rough)	0.15 mm	135.82 mm	28.46 mm	18.29 mm
First-Second (Fine)	0.08 mm	136.71 mm	21.42 mm	11.40 mm
Second-Third (Fine)	0.13 mm	136.72 mm	21.15 mm	13.10 mm
Third-Fourth (Fine)	0.11 mm	136.72 mm	38.57 mm	26.26 mm

These values show that, rough alignment via SURF contains nearly double the error than fine alignment via ICP. By applying fine alignment, maximum error difference is decreased, hence, RMS and mean errors are decreased too.

Also, in this experiment, we measured the real world distance of a specific point in each frame by using a regular meter. The difference between real world distance and fine registered point cloud's xyz coordinate is 2.23 mm for first frame, 2.81 mm for second frame and 3.01 mm for third frame. It shows that the error is increased because of SURF and ICP alignment applied to point cloud while accepting the first point cloud stationary.

5.2. ERROR AND SENSITIVITY ANALYSIS OF THE SYSTEM

In our experiments, we observed that a frame which contains depth and RGB data may be processed in less than one second. This time scale allows our system to be applied on high speed. Unfortunately, is caused less precise calculations. In this section, we discussed the error and sensitivity of the system. We also indicated causes for these.

5.2.1. Capturing Device Error

In our system Kinect is used to capture the depth and RGB frames. Each frame generated by the depth sensor is at VGA resolution (640 x 480 pixels), containing 11-bit depth values which provides 2048 levels of sensitivity. The output stream may run at a frame rate of 30 Hz. Also it has a sensitivity depending on the distance which is 3 mm at 2 m distance for the RGB image and 1 cm at 2 m distance for the depth image [47]. As can be seen in Fig. 5.1, the sensitivity of the depth image decreases by the distance.

Kinect's IR (depth) and RGB camera is physically close but they are different cameras. This causes a small difference in view angles. Also, an RGB pixel $C_{(x,y)}$ at the point (x, y) may not be the actual color value of depth pixel at the same point. That's why, we implemented

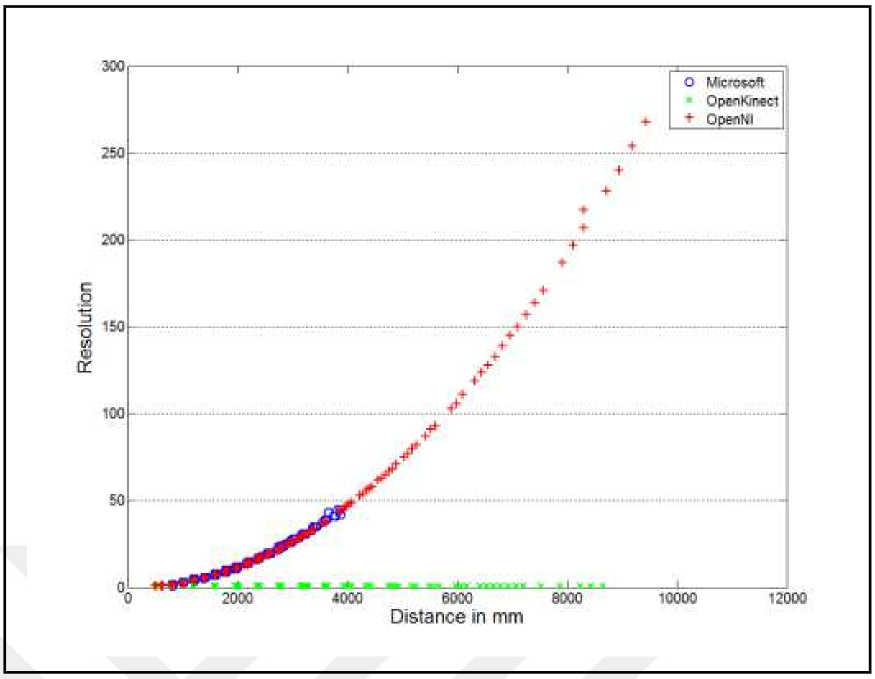


Figure 5.1. Kinect's resolution

RGB and depth frame alignment function which is provided by OpenNI SDK.

Also, Kinect's RGB camera is not working properly in capturing the luminance value as can be seen in many scenes. Even though images are taken consecutively, the luminance values differ. It effects the merging system as unmatched color planes point the same surface.

5.2.2. 3D Conversion Error

As images are taken in 2D, they should to be converted to 3D. To achieve this, we used Eqn. 4.1. Even though this is enough for this conversion, it caused an oblique surface as can be seen in Fig. 5.2. This error is caused by the difference of focal lengths of RGB and IR cameras. We used IR camera's focal length to achieve a higher success rate of plane matching. 2D to 3D conversion is made in 31 ms on average on a single core.

5.2.3. The Effect of Normal Vector Classification on RANSAC

We used surface normals to divide and classify the input point cloud. Surface normals are extracted by using Point Cloud Library. Then, outline removal is applied to filter small point groups. Surface normal calculation took approximately 390 ms in average for a frame. We used four cores parallel to calculate them. The duration is so long that prevents us to process input frames at 30 Hz which is the camera's capability. Hence, this process is a bottleneck with respect to calculation time. However, detecting surface normals provides us smaller

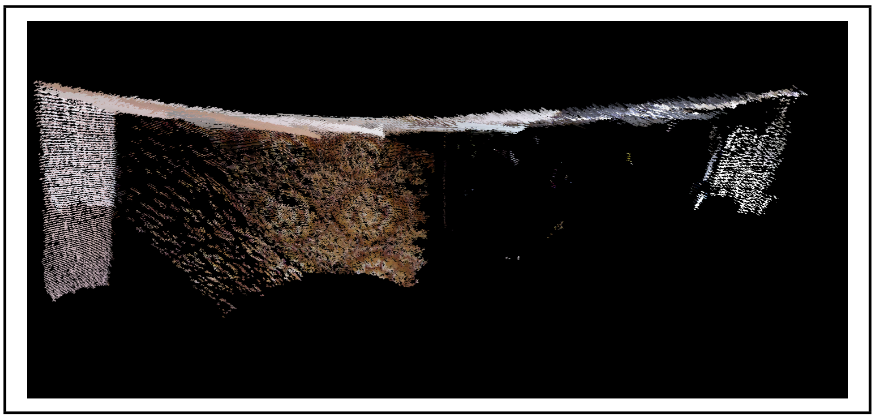


Figure 5.2. Oblique alignment error

point clouds which we can process more efficiently.

Outline removal took approximately 450 ms in average for a frame, a sum of three sub-planes. We used single core to calculate them. The duration is also so long that prevents us to process input frames at 30 Hz. This process is also a bottleneck with respect to calculation time. Filtering surface normals based sub-clouds by outline removal provides us more efficient surface plane extraction. Surface normal based classification may classify the same plane's adjacent points in different sub groups because of the capturing noise. Due to the capability of our capturing device, the noise may form a plane too. Outline removal process is filtering them by minimizing the count of point in noise group. Therefore, when RANSAC is applied only a good plane and some noise points exist in the point cloud. Hence, the RANSAC process did not consume much CPU time. As a result, RANSAC costs approximately 75 ms in average for a frame, a total of three sub clouds.

We have only one fail case in plane extraction. Also, we did not experience any completely failed RANSAC output. Each output was representing a plane in real life. So, we achieved to eliminate the error mentioned in Section 4.4.2.2. This method's only weakness is the calculation time. To detect planes in a frame, approximately 840 ms in average is used totally.

5.2.4. The Effect of SURF on Registration

Instead of merging point clouds conventionally by using ICP [40](which is an iterative method), we used SURF which provides us much faster solution. In our work, the advantage is that the images are taken from one camera which turns. Hence, the problem becomes calculating the turn angle and rotate the relevant point cloud by this angle. Therefore, we used RGB images to find the same points in different view angles. Calculating all the point matches on RGB image and rotating the point cloud costs approximately 78 ms in average for a frame. This value may be larger if the matched points increase.

By using SURF to register frames, we encountered problems on simple scenes containing only walls, ceiling and floor. We couldn't match enough points between RGB images to calculate turn angles. Hence, merging process failed. In our tests, we failed to merge point clouds only on four scenes out of fourteen. Compared to the calculation speed, it is an high success ratio. As a result, our system is able to merge point clouds too fast with small alignment error which can be discarded.

Using cumulative angle estimation has a weak point. If no point is matched in any of the frames, the angle cannot be calculated. This causes an error for current frame and it's consecutive frames. Also, the frame which contains error overlaps the correct point cloud. The calculated angle contains a precision error as we take the arithmetic average of all matched points. By adding the turn angle each frame, we are also adding error which increases at each subsequent frames.

6. CONCLUSION

In building renovation, survey is the core for planning and the designing steps. For most cases, there is no need for expensive and micron-precise range sensors in these steps. Especially on pricing and planning phases of the renovation project, users need fast operations and quick results. This can be obtained by easing depth data resolution constraints. To achieve this goal, we propose a fast and mobile end to end system. Our system can be used to extract building walls in an iterative manner. In implementation, we used the TI OMAP4460 processor based PandaBoard-ES to realize the mobile part of our system. This single-board computer provides us the processing power which is needed to capture the depth and RGB images from the Kinect sensor. It can also run the compression algorithm along with the on board wireless module to connect with a PC in real time. Our wall extraction software benefits from RANSAC and the perpendicularity constrains inherent in wall formations. Also, we applied a SURF based merging algorithm. We tested the performance of our system on two different scenarios. The results indicate that our system can be used as a prototype for building wall extraction in real time with acceptable performance. Our system costs less than $ 300 excluding the PC. Based on the resolution of the Kinect sensor, we have at most 7 cm depth perception error while surveying an environment within 0.7-5 m range. During tests, we observed that our system successfully detects walls, floor, and the ceiling. Our system may produce some false alarms when an object resembles a plane. However, our algorithm works much more efficient when there are various objects in scene. Therefore, in the present form of our system, we suggest it to be used in environments with fewer furniture and more objects. On the other hand, our system has a mobile data capturing part. This can be implemented on a moving robot to improve the performance of the system. Our system can also be improved by color based plane filtering and wall pattern tracking modules.

REFERENCES

1. Eren, G., “*3D Scanning of Transparent objects*”, *Universite de Bourgogne - Sabanci Universitesi, PhD thesis*, September 2010.

2. Valkenburg, R. and A. McIvor, “*Accurate 3D measurement using a Structured Light System*”, *Image and Vision Computing*, Vol. 16, pp. 99–110, 1996.

3. Rusu, R. and S. Cousins, “*3D is here: Point Cloud Library (PCL)*”, *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 1–4, May 2011.

4. Henry, P., M. Krainin, E. Herbst, X. Ren and D. Fox, “*Rgbd mapping: Using depth cameras for dense 3d modeling of indoor environments*”, *In RGB-D: Advanced Reasoning with Depth Cameras Workshop in conjunction with RSS*, 2010.

5. Du, H., P. Henry, X. Ren, M. Cheng, D. B. Goldman, S. M. Seitz and D. Fox, *Interactive 3D Modeling of Indoor Environments with a Consumer Depth Camera*.

6. Xiong, X. and D. Huber, “*Using Context to Create Semantic 3D Models of Indoor Environments*”, *Proceedings of the British Machine Vision Conference (BMVC)*, September 2010.

7. Liu, Y., R. Emery, D. Chakrabarti, W. Burgard and S. Thrun, “*Using EM to Learn 3D Models with Mobile Robots*”, *Proceedings of the International Conference on Machine Learning (ICML)*, 2001.

8. Biber, P., H. Andreasson, T. Duckett and A. Schilling, “*3D modeling of indoor environments by a mobile robot with a laser scanner and panoramic camera*”, Vol. 4, pp. 3430–3435 vol.4, Sept 2004.

9. Lu, J. and G. Wang, “*Human-centric Indoor Environment Modeling from Depth Videos*”, *Proceedings of the 12th International Conference on Computer Vision - Volume 2*, ECCV’12, pp. 42–51, 2012.

10. Hähnel, D., W. Burgard and S. Thrun, "*Learning Compact 3D Models of Indoor and Outdoor Environments with a Mobile Robot*", *Robotics and Autonomous Systems*, Vol. 44, No. 1, pp. 15–27, 2003.

11. Silberman, N. and R. Fergus, "*Indoor scene segmentation using a structured light sensor*", *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pp. 601–608, Nov 2011.

12. Filliat, D., E. Battesti, S. Bazeille, G. Duceux, A. Gepperth, L. Harrath, I. Jebari, R. Pereira, A. Tapus, C. Meyer, S.-H. Ieng, R. Benosman, E. Cizeron, J.-C. Mamanna and B. Pothier, "*RGBD object recognition and visual texture classification for indoor semantic mapping*", *Technologies for Practical Robot Applications (TePRA), 2012 IEEE International Conference on*, pp. 127–132, April 2012.

13. Kunze, L., M. Beetz, M. Saito, H. Azuma, K. Okada and M. Inaba, "*Searching objects in large-scale indoor environments: A decision-theoretic approach*", *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pp. 4385–4390, May 2012.

14. Bills, C., J. Chen and A. Saxena, "*Autonomous MAV flight in indoor environments using single image perspective cues*", *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 5776–5783, May 2011.

15. Khalil, O. A., P. Grussenmeyer and M. N. El Din, "*3D Indoor Modeling of Buildings Based on Photogrammetry and Topological Approaches*", *Surveying and documentation of historic buildings- monuments- sites International symposium; 18th, CIPA; Surveying and documentation of historic buildings- monuments- sites*, pp. 5776–5783, 2002.

16. Cheng, P., *Texture Mapping 3D Models of Indoor Environments with Noisy Camera Poses*, Master's thesis, EECS Department, University of California, Berkeley, Dec 2013.

17. Ikeda, S. and J. Miura, "*3D Indoor Environment Modeling by a Mobile Robot with Omnidirectional Stereo and Laser Range Finder*", *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pp. 3435–3440, Oct 2006.

18. Elgazzar, S., R. Liscano, F. Blais and A. Miles, "*Active range sensing for indoor envi-*

ronment modeling", *Instrumentation and Measurement, IEEE Transactions on*, Vol. 47, No. 1, pp. 260–264, Feb 1998.

19. Thrun, S., C. Martin, Y. Liu, D. Hahnel, R. Emery-Montemerlo, D. Chakrabarti and W. Burgard, "*A real-time expectation-maximization algorithm for acquiring multiplanar maps of indoor environments with mobile robots*", *Robotics and Automation, IEEE Transactions on*, Vol. 20, No. 3, pp. 433–443, June 2004.

20. Sequeira, V., J. G. M. Gonqalves and M. I. Ribeiro, *3D Modeling of In-door Scenes Using Laser Range Sensing*.

21. Williams, N., K.-L. Low, C. Hantak, M. Pollefeys and A. Lastra, "*Automatic image alignment for 3D environment modeling*", *Computer Graphics and Image Processing, 2004. Proceedings. 17th Brazilian Symposium on*, pp. 388–395, Oct 2004.

22. *Hooking up a Kinect to your Computer (Using Ubuntu)*.

23. Steder, B., R. Rusu, K. Konolige and W. Burgard, "*Point feature extraction on 3D range scans taking into account object boundaries*", *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 2601–2608, May 2011.

24. Muja, M., R. Rusu, G. Bradski and D. Lowe, "*REIN - A fast, robust, scalable REcognition INfrastructure*", *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 2939–2946, May 2011.

25. Holz, D., S. Holzer, R. Rusu and S. Behnke, "*Real-Time Plane Segmentation Using RGB-D Cameras*", T. Röfer, N. Mayer, J. Savage and U. Saranlı (Editors), *RoboCup 2011: Robot Soccer World Cup XV*, Vol. 7416 of *Lecture Notes in Computer Science*, pp. 306–317, Springer Berlin Heidelberg, 2012.

26. Meeussen, W., M. Wise, S. Glaser, S. Chitta, C. McGann, P. Mihelich, E. Marder-Eppstein, M. Muja, V. Eruhimov, T. Foote, J. Hsu, R. Rusu, B. Marthi, G. Bradski, K. Konolige, B. Gerkey and E. Berger, "*Autonomous door opening and plugging in with a personal robot*", *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pp. 729–736, May 2010.

27. Rusu, R., A. Holzbach, N. Blodow and M. Beetz, "*Fast geometric point labeling using conditional random fields*", *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pp. 7–12, Oct 2009.

28. Rusu, R., N. Blodow, Z. Marton and M. Beetz, "*Close-range scene segmentation and reconstruction of 3D point cloud maps for mobile manipulation in domestic environments*", *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pp. 1–6, Oct 2009.

29. Dostal, J., P. O. Kristensson and A. Quigley, "*The potential of fusing computer vision and depth sensing for accurate distance estimation*", *Proceedings of CHI'13*, 2013.

30. Andersen, M. R., T. Jensen, P. Lisouski, A. K. Mortensen, M. K. Hansen, T. Gregersen and P. Ahrendt, *Kinect depth sensor evaluation for computer vision applications*, Tech. Rep. Technical report ECE-TR-6, Aarhus University, 2012.

31. ARM, *Cortex-A9 Floating Point Unit, Technical Reference Manual*, Tech. Rep. Revision r2p2, ARM Company, 2010.

32. Weisstein, E. W., *"Right-Hand Rule." From MathWorld–A Wolfram Web Resource*.

33. Hogman, V., "*Building a 3D map from RGB-D sensors*", *Royal Institute of Technology (KTH), Stockholm, Sweden, MSc. Thesis*, 2012.

34. Rusu, R. B., *Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments*, phd, Tecnische Universitatet Muenchen, Munich, Germany, 10/2009 2009.

35. Berkmann, J. and T. Caelli, "*Computation of surface geometry and segmentation using covariance techniques*", *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, Vol. 16, No. 11, Nov 1994.

36. Shakarji, C. M., "*Least-Squares Fitting Algorithms of the NIST Algorithm Testing System*", *Journal of Research of the National Institute of Standards and Technology*, pp. 633–641, 1998.

37. Hetzel, G., B. Leibe, P. Levi and B. Schiele, "*3D object recognition from range images using local feature histograms*", *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, Vol. 2, pp. II–394–II–399 vol.2, 2001.

38. Fischler, M. A. and R. C. Bolles, "*Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography*", *Commun. ACM*, Vol. 24, No. 6, pp. 381–395, Jun. 1981.

39. Raguram, R., J.-M. Frahm and M. Pollefeys, "*A Comparative Analysis of RANSAC Techniques Leading to Adaptive Real-Time Random Sample Consensus*", D. Forsyth, P. Torr and A. Zisserman (Editors), *Computer Vision – ECCV 2008*, Vol. 5303 of *Lecture Notes in Computer Science*, pp. 500–513, Springer Berlin Heidelberg, 2008.

40. Besl, P. and N. D. McKay, "*A method for registration of 3-D shapes*", *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, Vol. 14, No. 2, Feb 1992.

41. Bay, H., T. Tuytelaars and L. V. Gool, "*SURF: Speeded Up Robust Features*", *Proceedings of the ninth European Conference on Computer Vision*, May 2006.

42. Lowe, D. G., "*Distinctive Image Features from Scale-Invariant Keypoints*", *International Journal of Computer Vision*, Vol. 60, pp. 91–110, 2004.

43. Brown, M. and D. Lowe, "*Invariant Features from Interest Point Groups*", *In British Machine Vision Conference*, pp. 656–665, 2002.

44. Muja, M. and D. G. Lowe, "*Fast Matching of Binary Features*", *Computer and Robot Vision (CRV)*, pp. 404–410, 2012.

45. Brin, S., "*Near neighbor search in large metric spaces*", *In Proceedings of the 21th International Conference on Very Large Data Bases*, 1995.

46. Aspert, N., D. Santa-cruz and T. Ebrahimi, "*MESH: Measuring Errors between Surfaces using the Hausdorff distance*", pp. 705–708, 2002.

47. M.R. Andersen, P. L. A. M. M. H. T. G., T. Jensen and P. Ahrend, “*Department of Engineering, Aarhus University. Denmark. 37 pp.*”, *Kinect Depth Sensor Evaluation For Computer Vision Applications*, 2012.

APPENDIX A: RESULT IMAGES

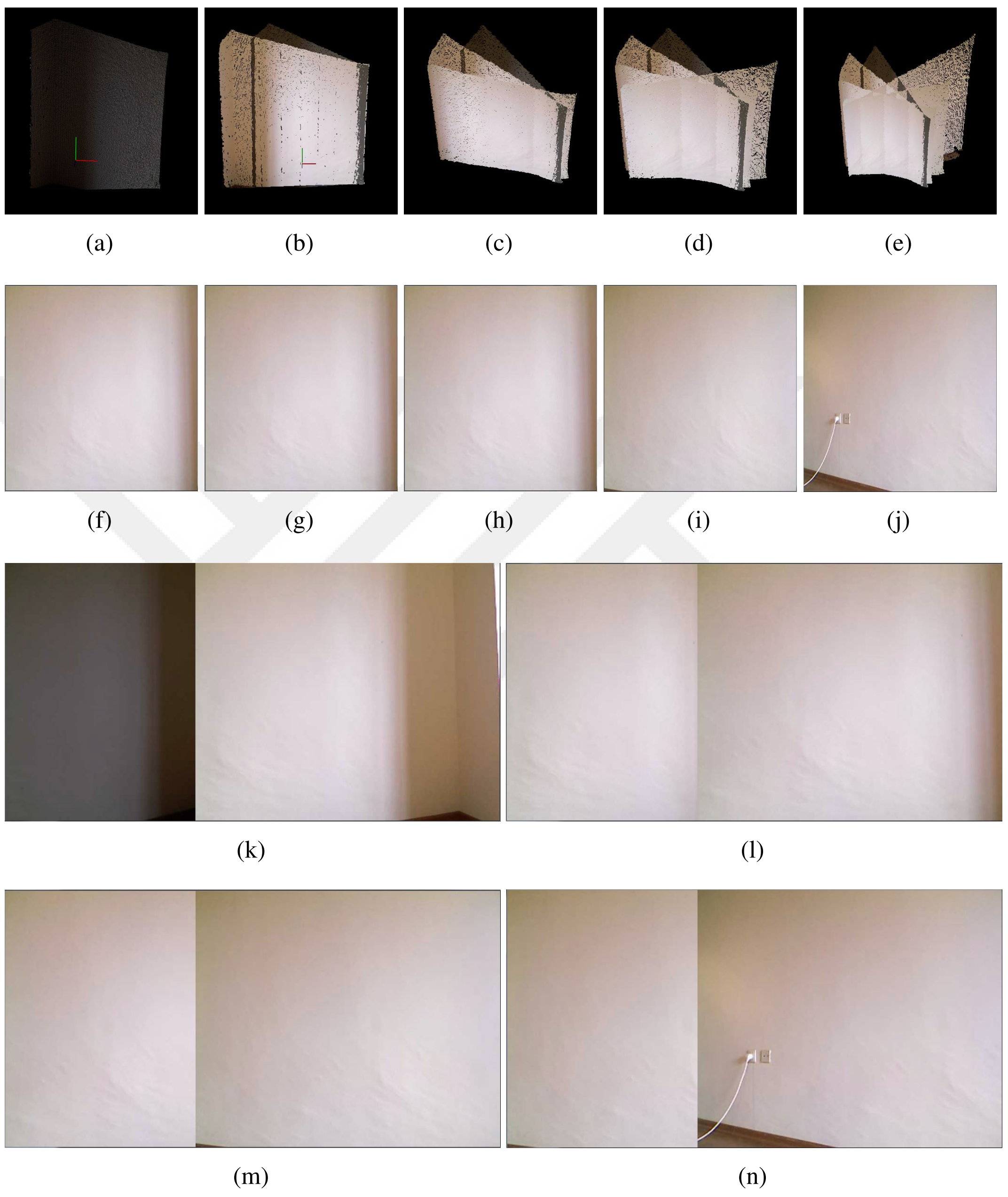


Figure A.1. Scene 1 images where (a)-(e) are 3D results, (f)-(j) are RGB images, (k)-(n) are matching visualization of the frames

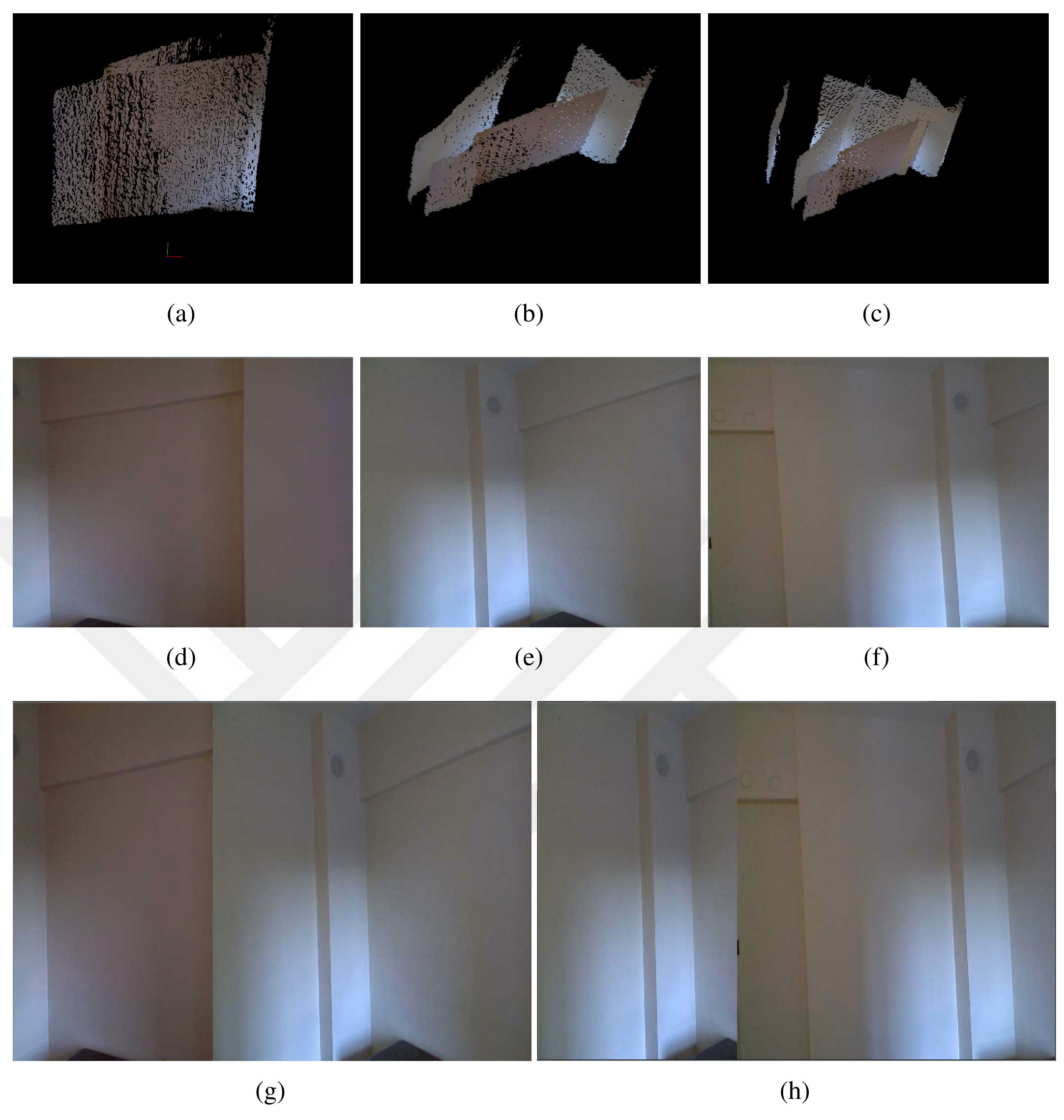


Figure A.2. Scene 2 images where (a)-(c) are 3D results, (d)-(f) are RGB images, (g)-(h) are matching visualization of the frames

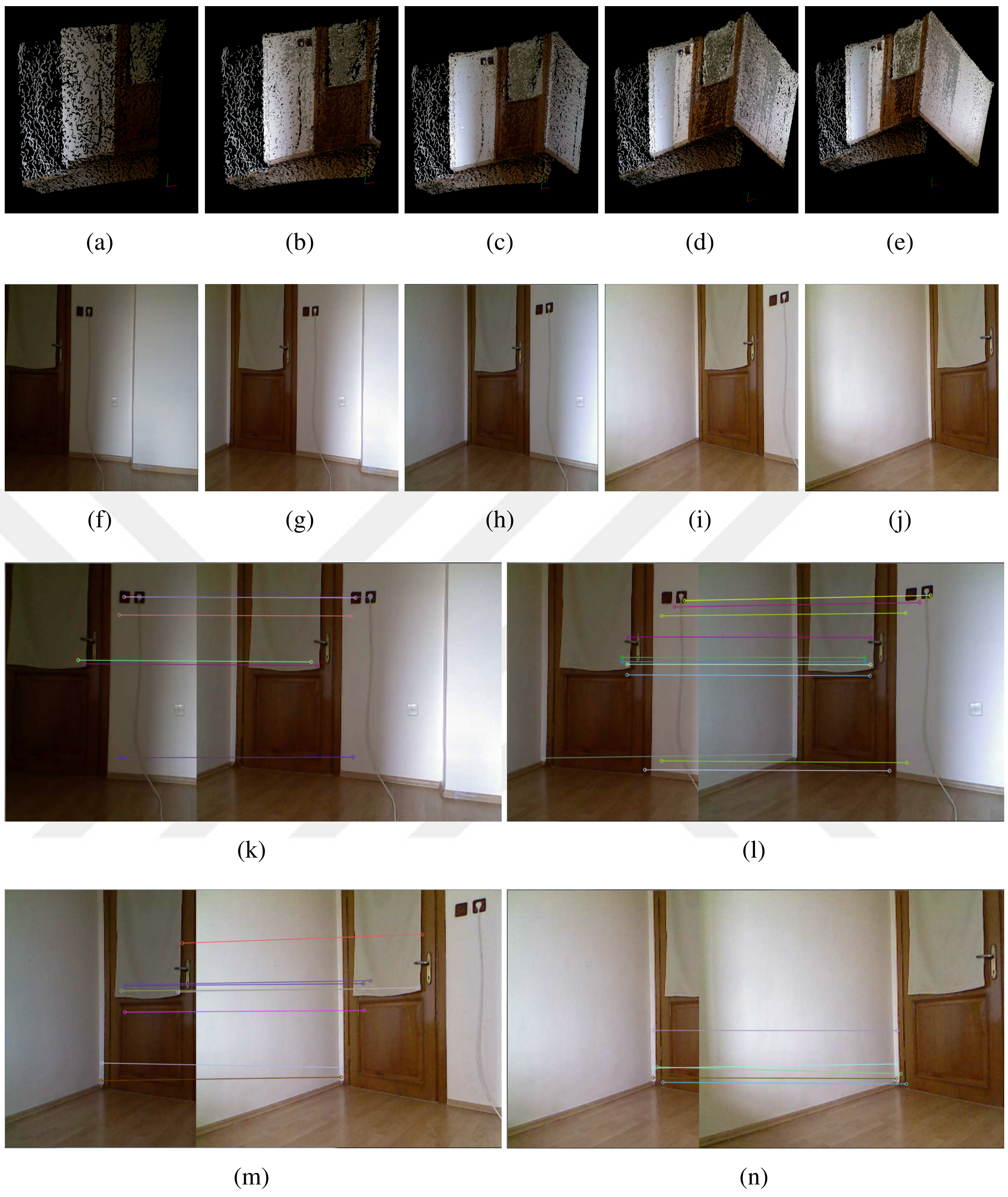


Figure A.3. Scene 3 images where (a)-(e) are 3D results, (f)-(j) are RGB images, (k)-(n) are matching visualization of the frames

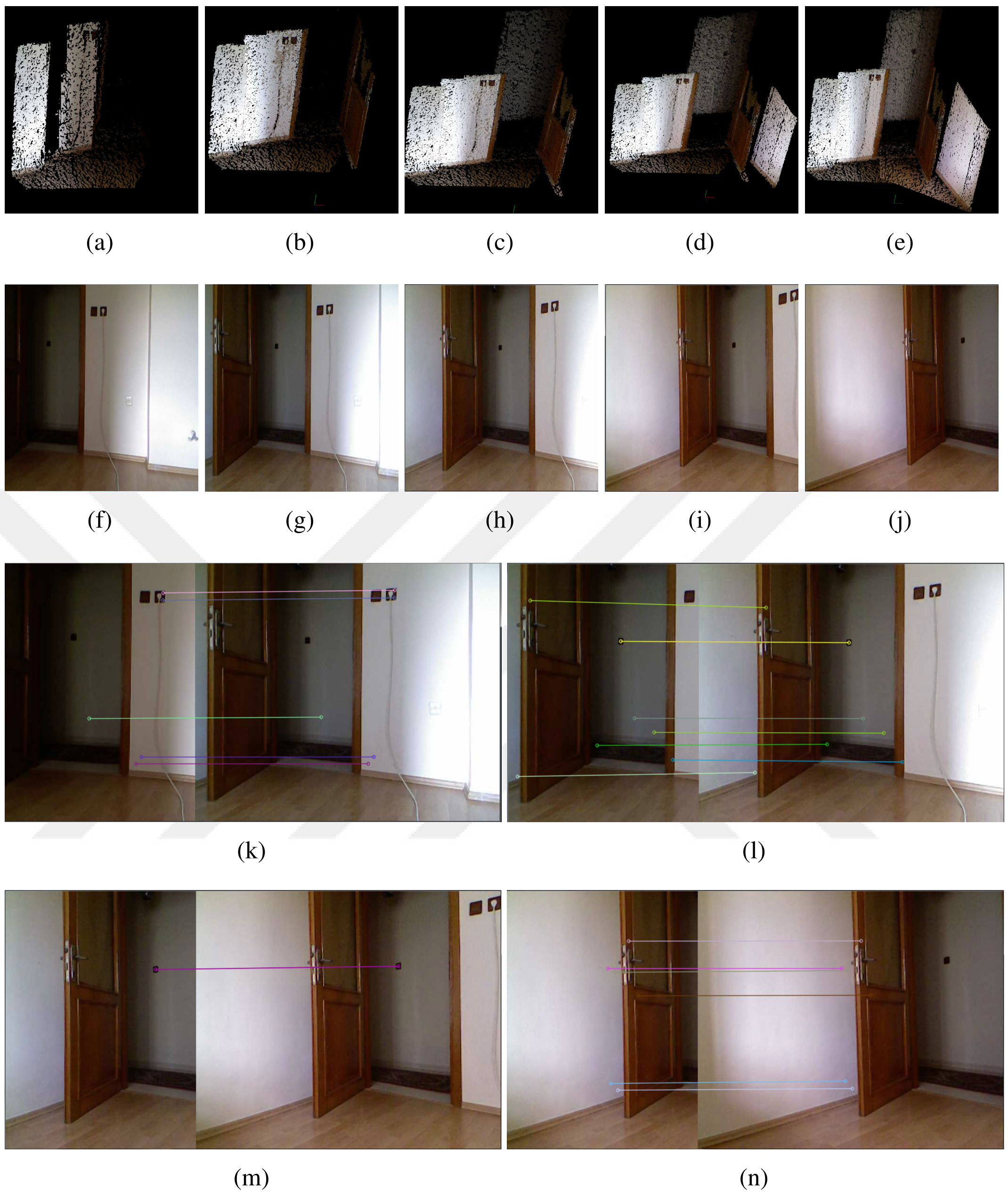
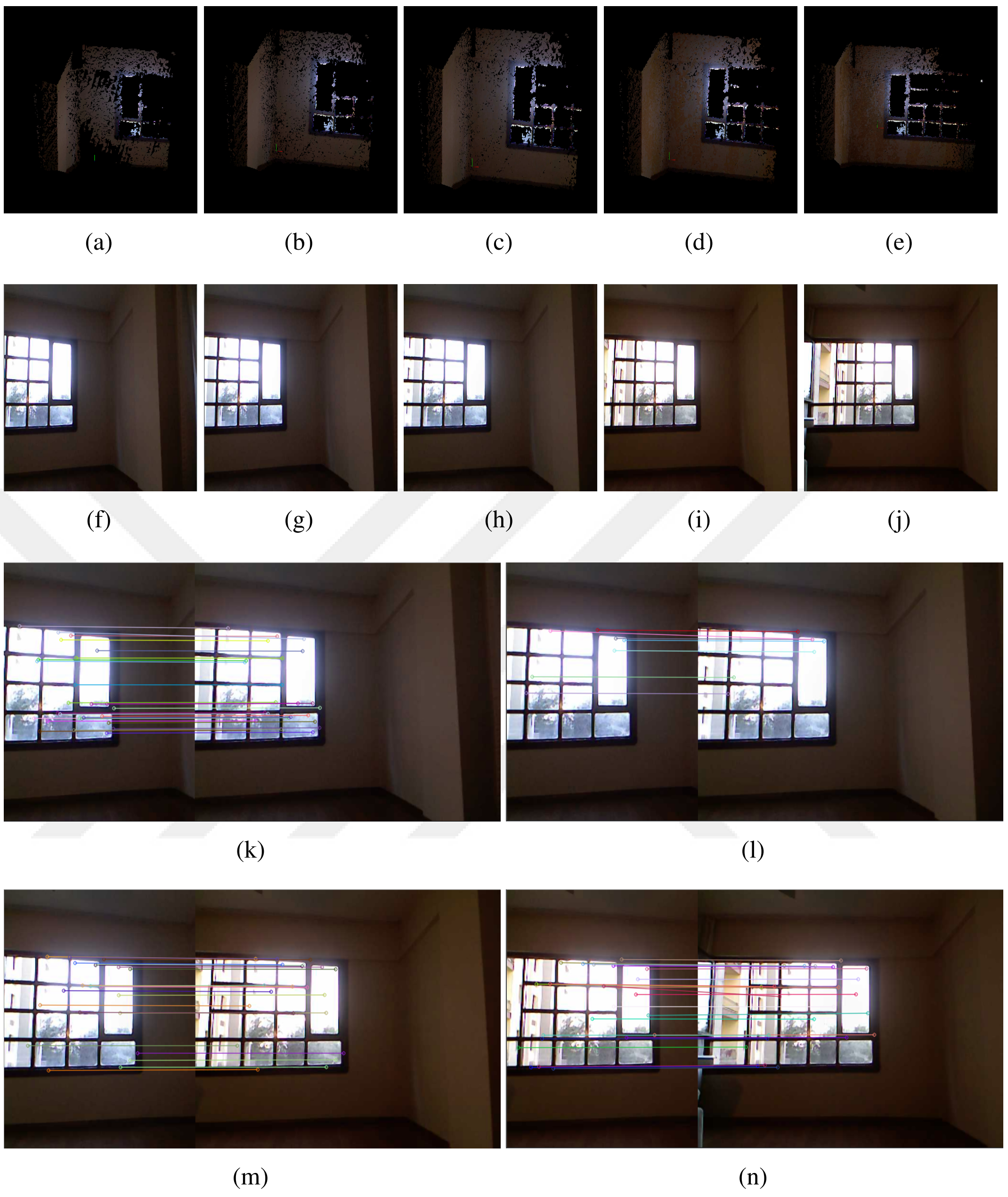


Figure A.4. Scene 4 images where (a)-(e) are 3D results, (f)-(j) are RGB images, (k)-(n) are matching visualization of the frames


(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

(k) (l)

(m) (n)

Figure A.5. Scene 5 images where (a)-(e) are 3D results, (f)-(j) are RGB images, (k)-(n) are matching visualization of the frames

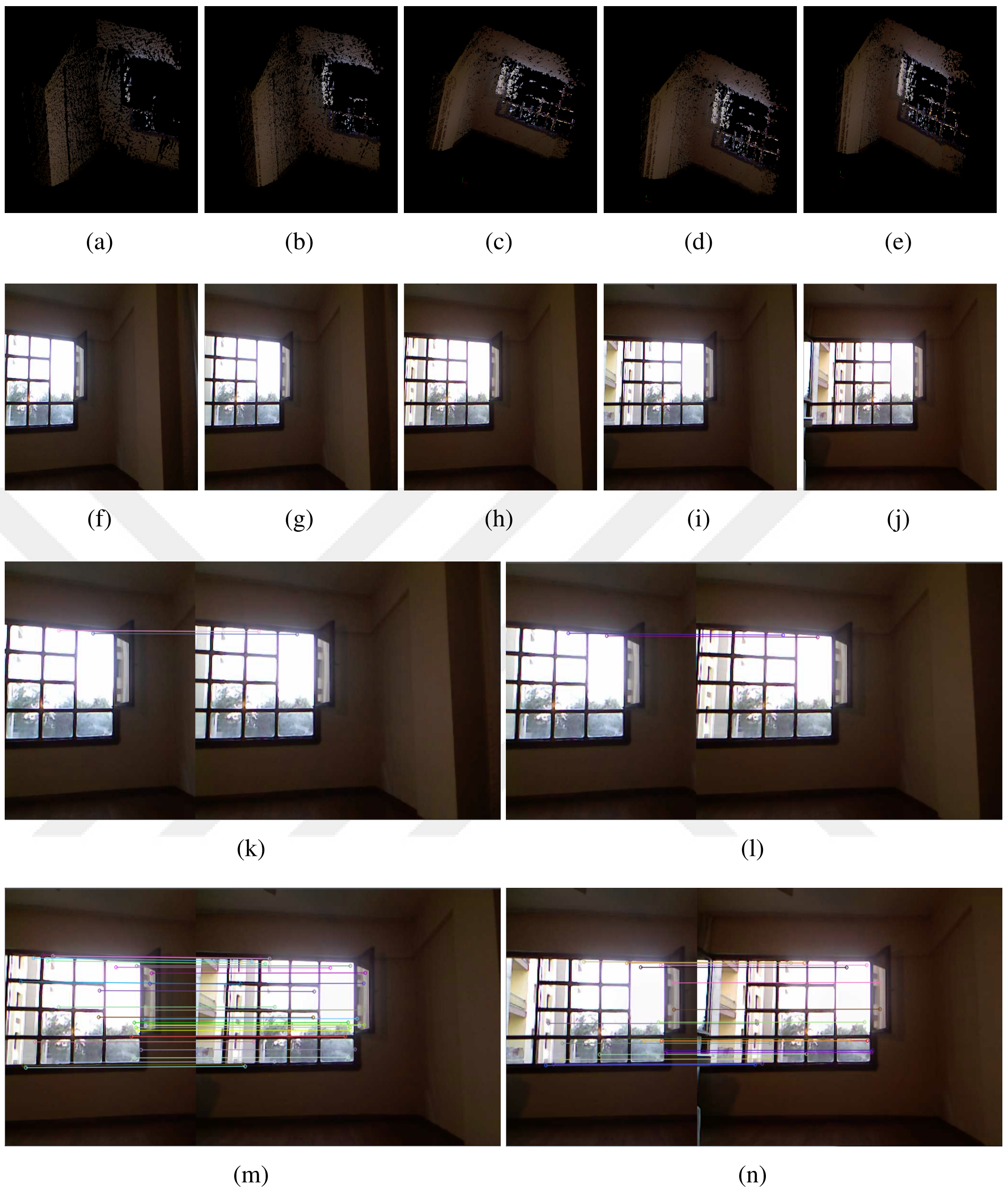


Figure A.6. Scene 6 images where (a)-(e) are 3D results, (f)-(j) are RGB images, (k)-(n) are matching visualization of the frames

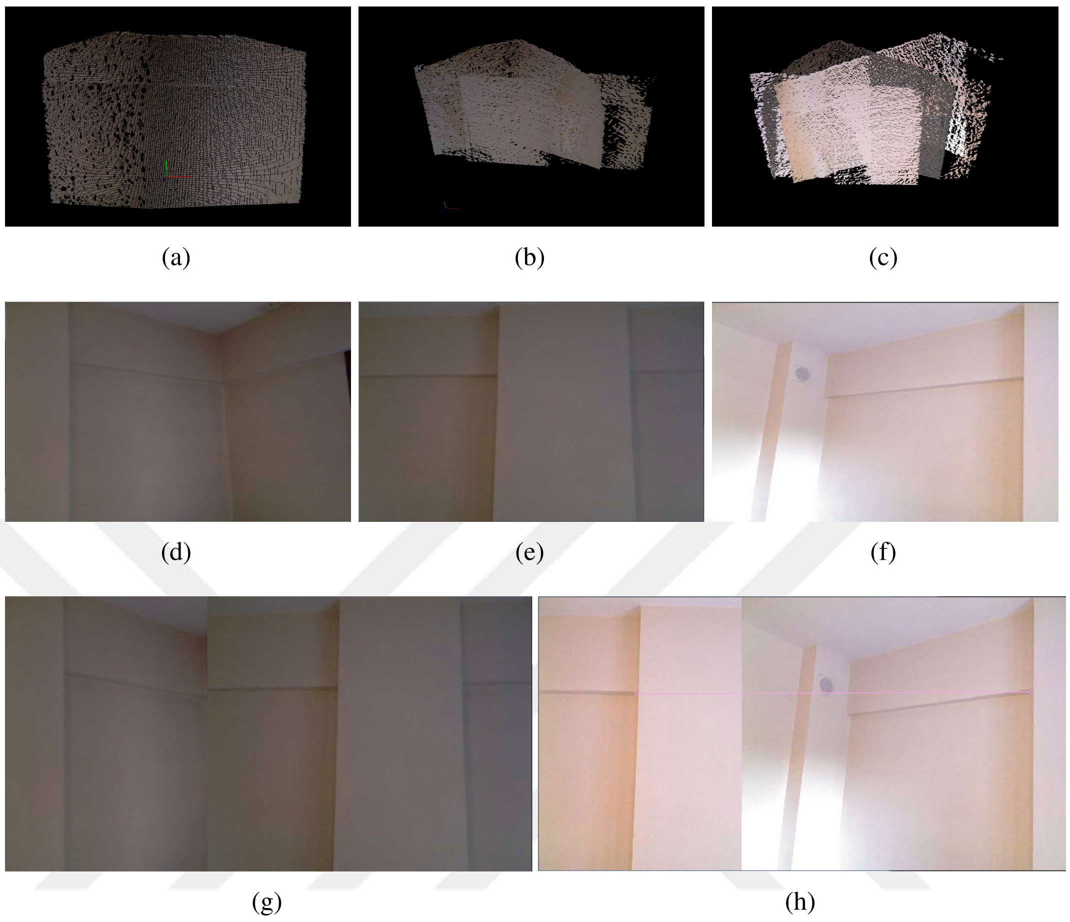


Figure A.7. Scene 7 images where (a)-(c) are 3D results, (d)-(f) are RGB images, (g)-(h) are matching visualization of the frames

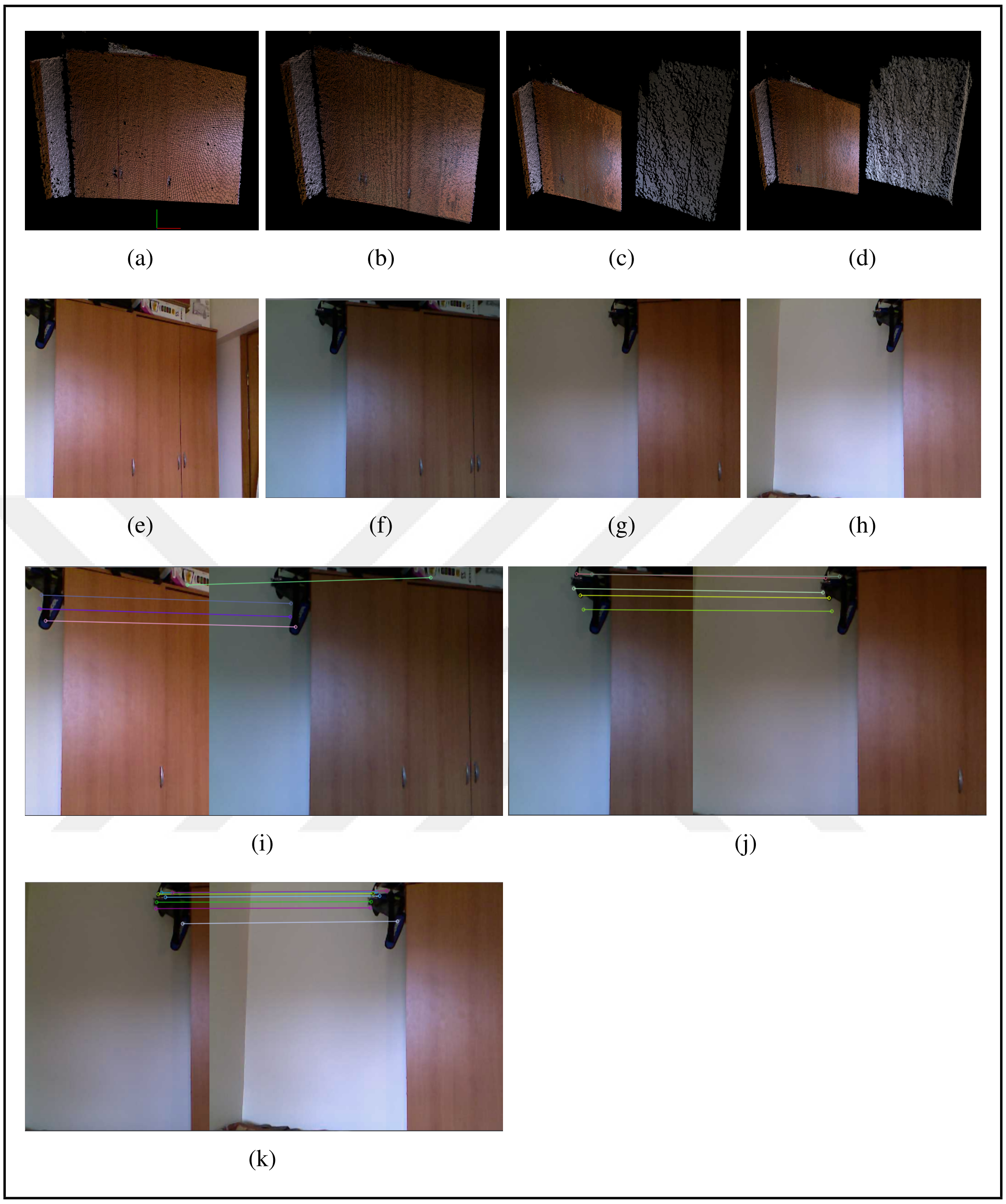


Figure A.8. Scene 8 images where (a)-(d) are 3D results, (e)-(h) are RGB images, (i)-(k) are matching visualization of the frames

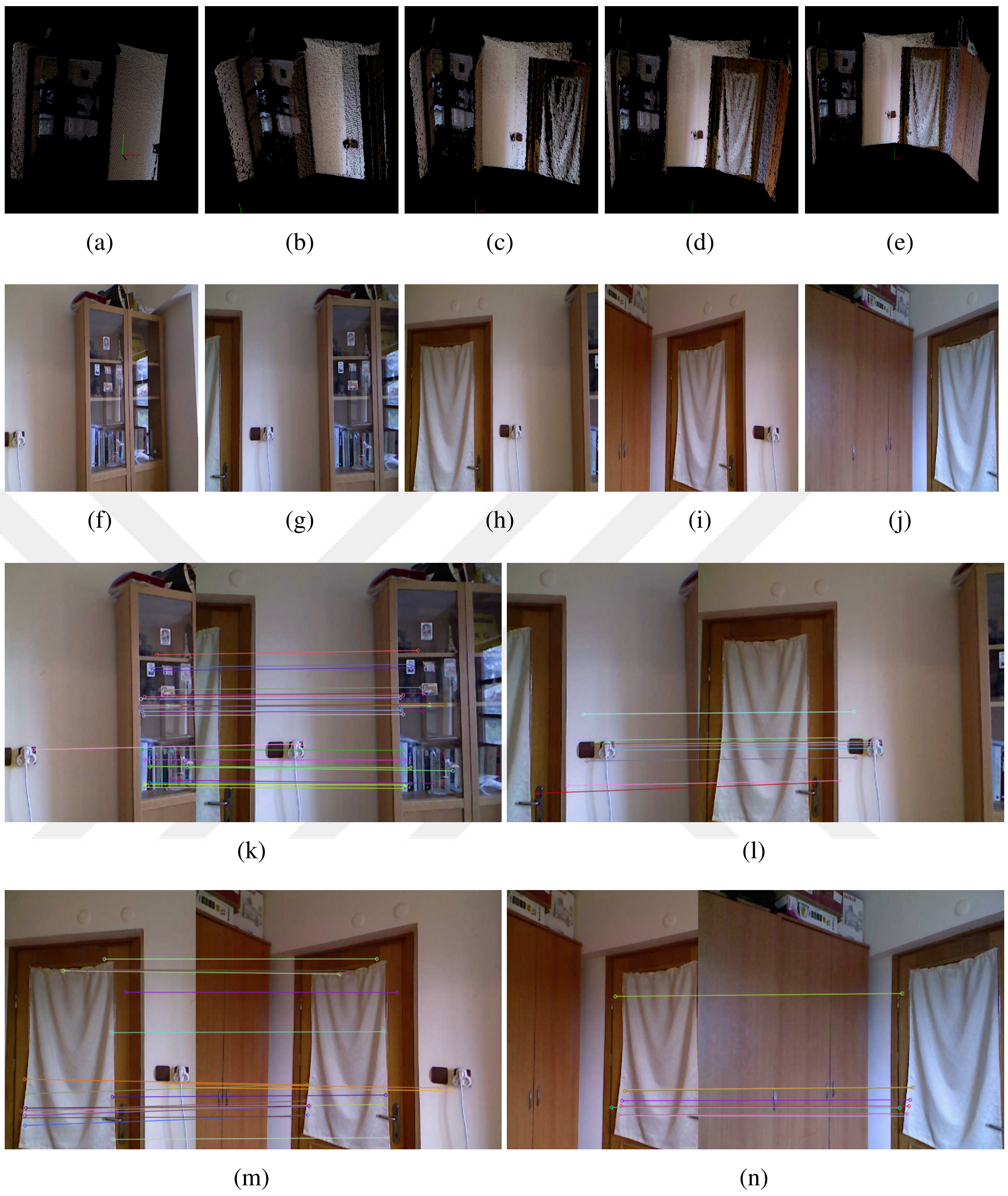


Figure A.9. Scene 9 images where (a)-(e) are 3D results, (f)-(j) are RGB images, (k)-(n) are matching visualization of the frames

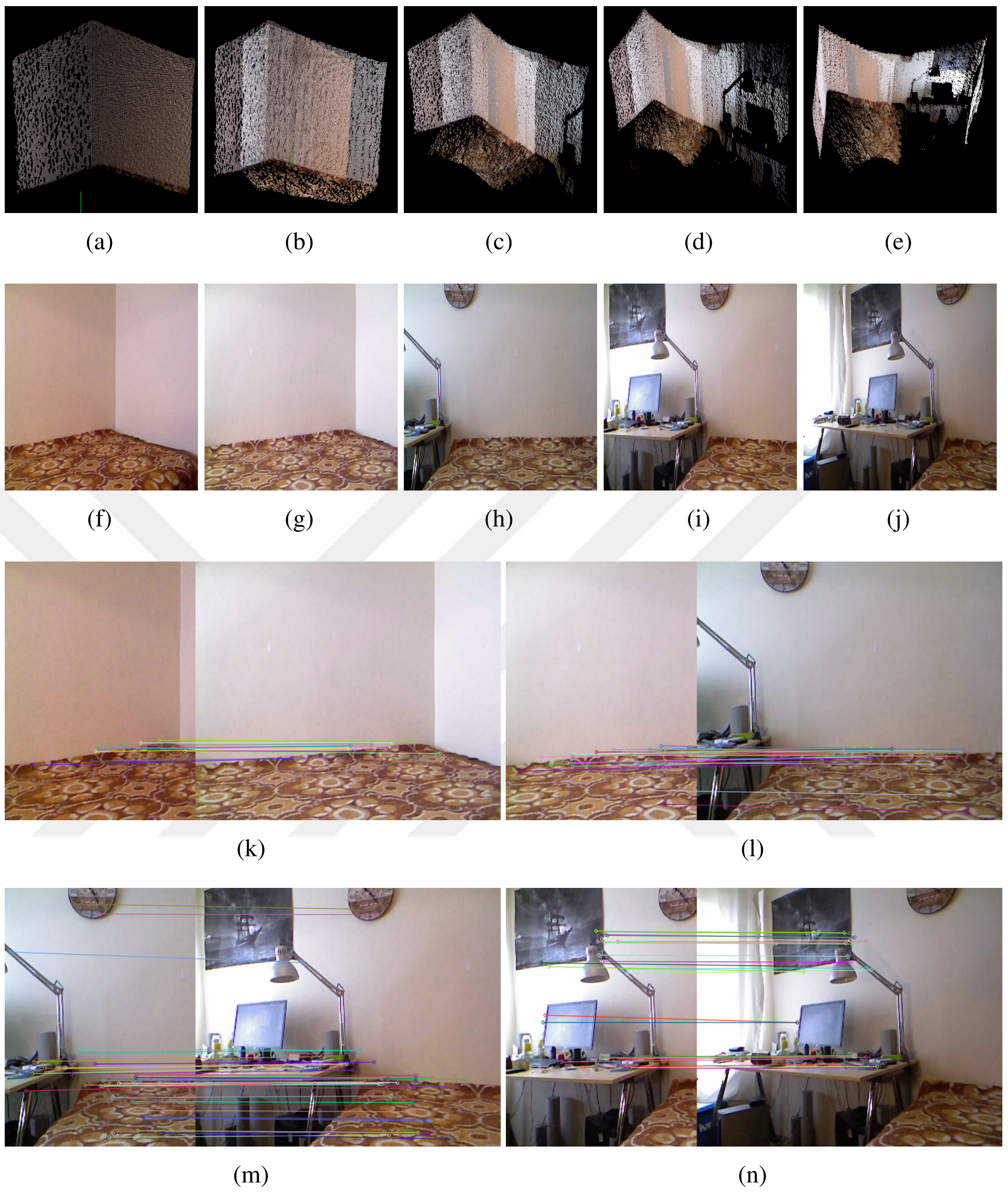


Figure A.10. Scene 10 images where (a)-(e) are 3D results, (f)-(j) are RGB images, (k)-(n) are matching visualization of the frames

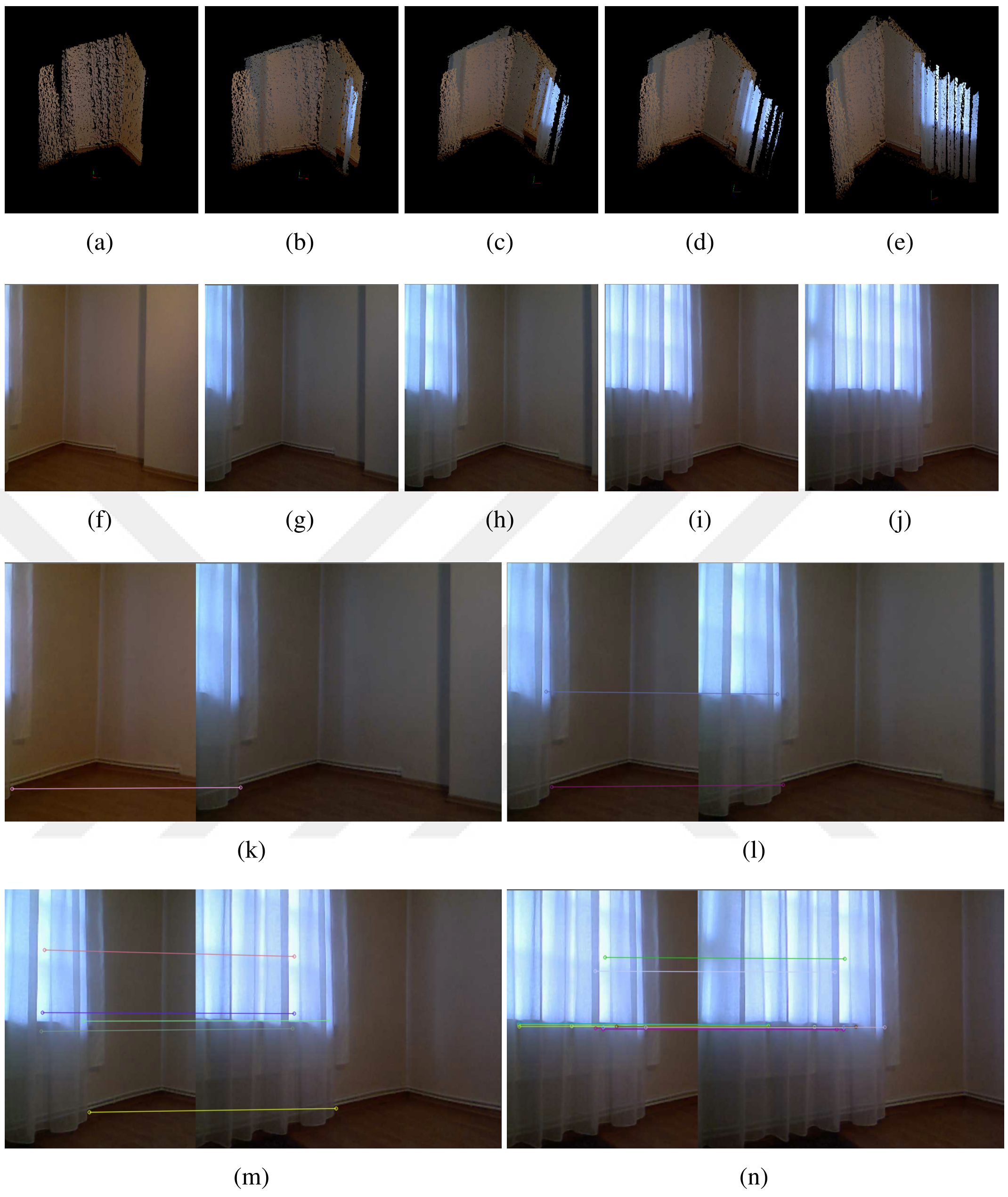


Figure A.11. Scene 11 images where (a)-(e) are 3D results, (f)-(j) are RGB images, (k)-(n) are matching visualization of the frames

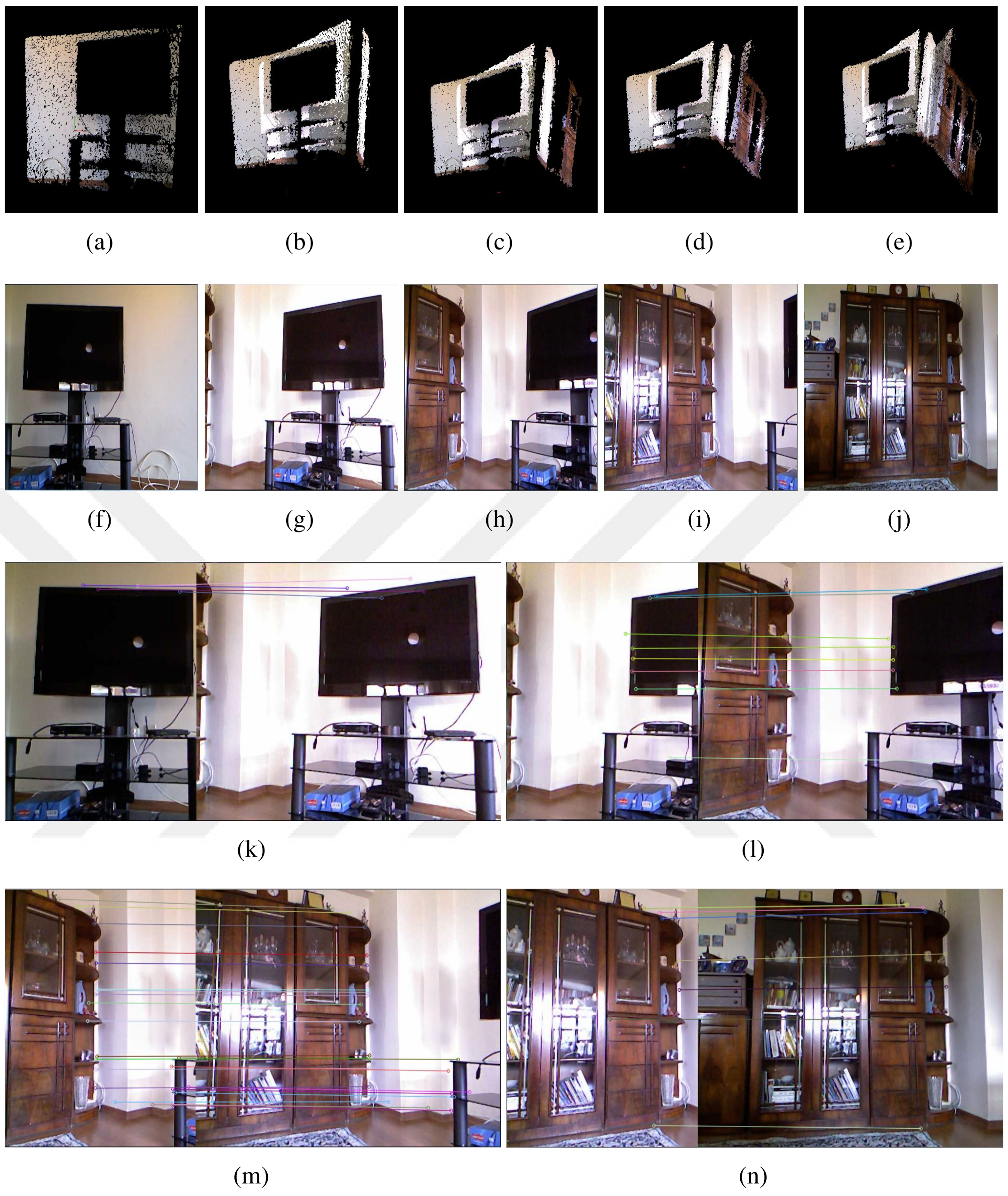


Figure A.12. Scene 12 images where (a)-(e) are 3D results, (f)-(j) are RGB images, (k)-(n) are matching visualization of the frames

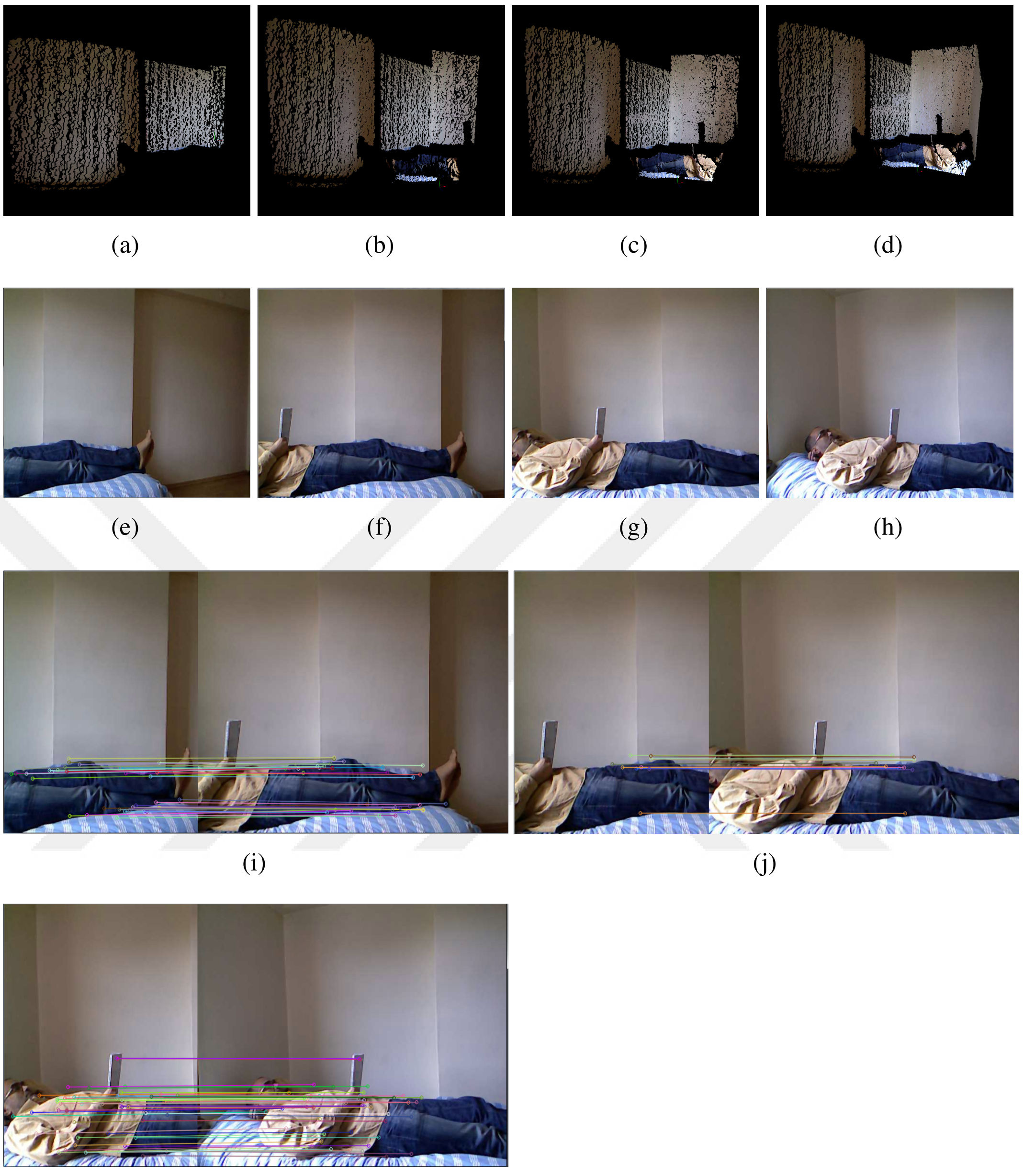


Figure A.13. Scene 13 images where (a)-(d) are 3D results, (e)-(h) are RGB images, (i)-(k) are matching visualization of the frames

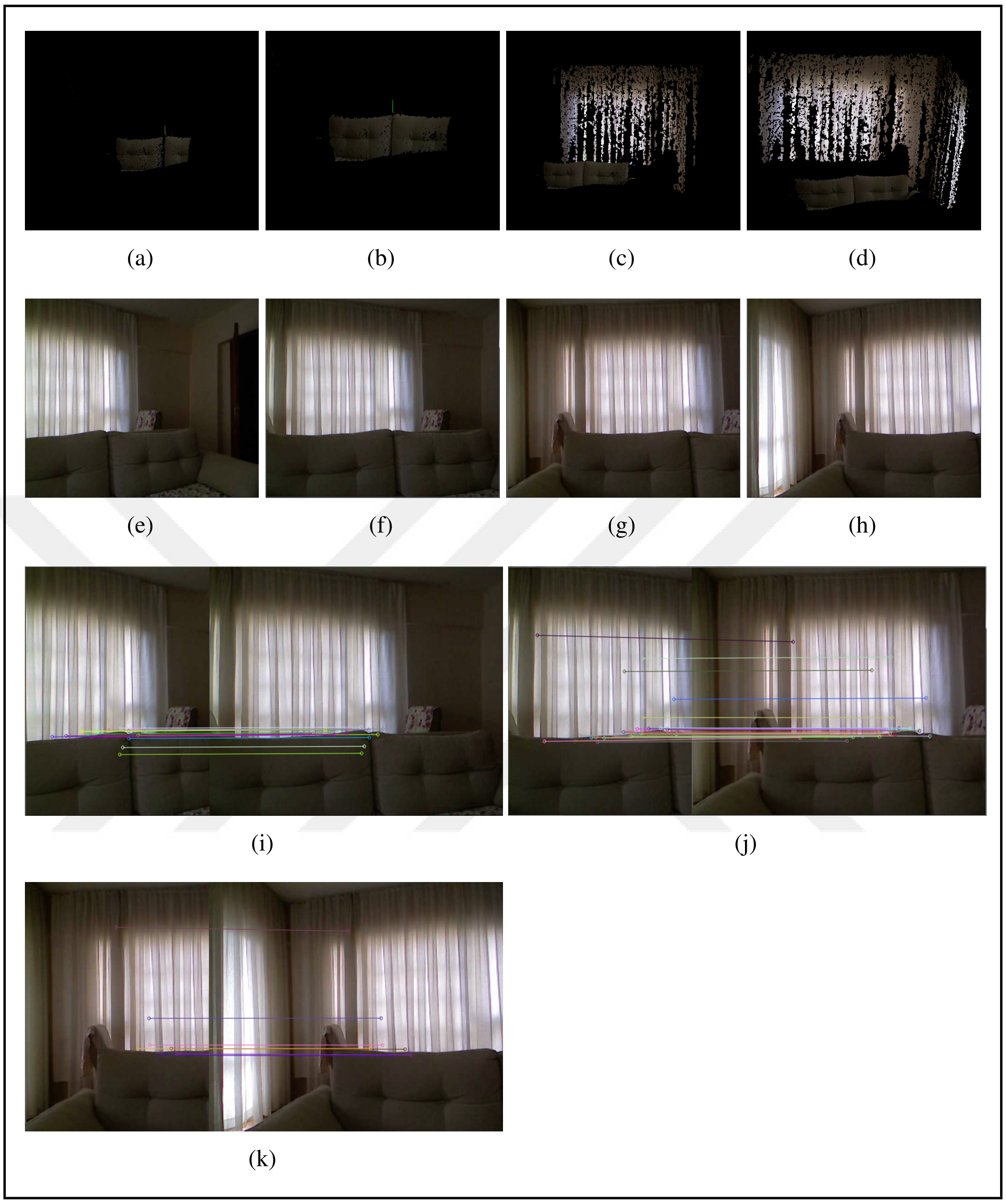


Figure A.14. Scene 14 images where (a)-(d) are 3D results, (e)-(k) are RGB images, (i)-(k) are matching visualization of the frames

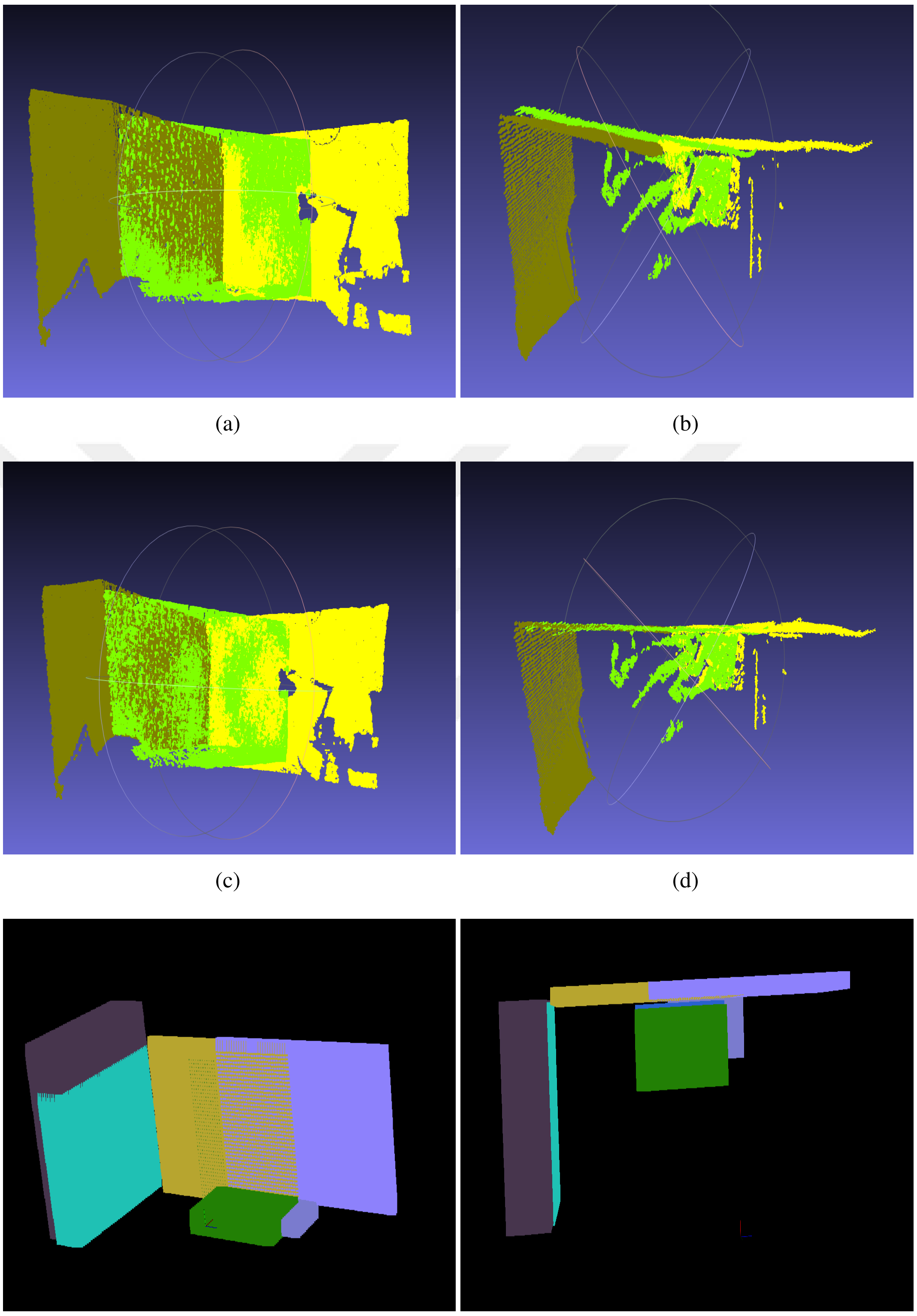

(a) (b) (c) (d) (e) (f)

Figure A.15. Scene 15 images where (a)-(b) are crude registrations, (c)-(d) are fine registrations, (e)-(f) are plane representations of the frames

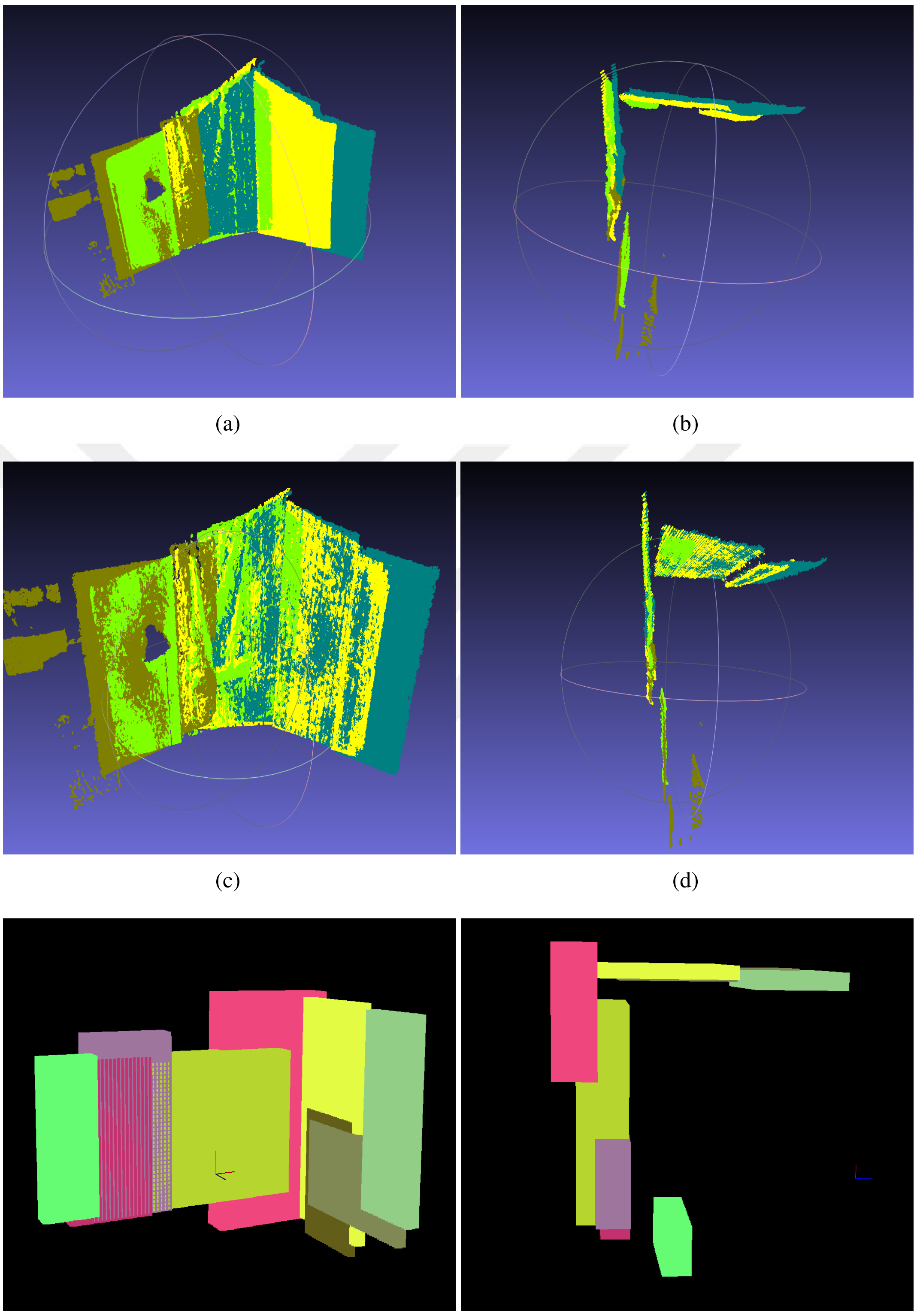

(a) (b) (c) (d) (e) (f)

Figure A.16. Scene 16 images where (a)-(b) are crude registrations, (c)-(d) are fine registrations, (e)-(f) are plane representations of the frames

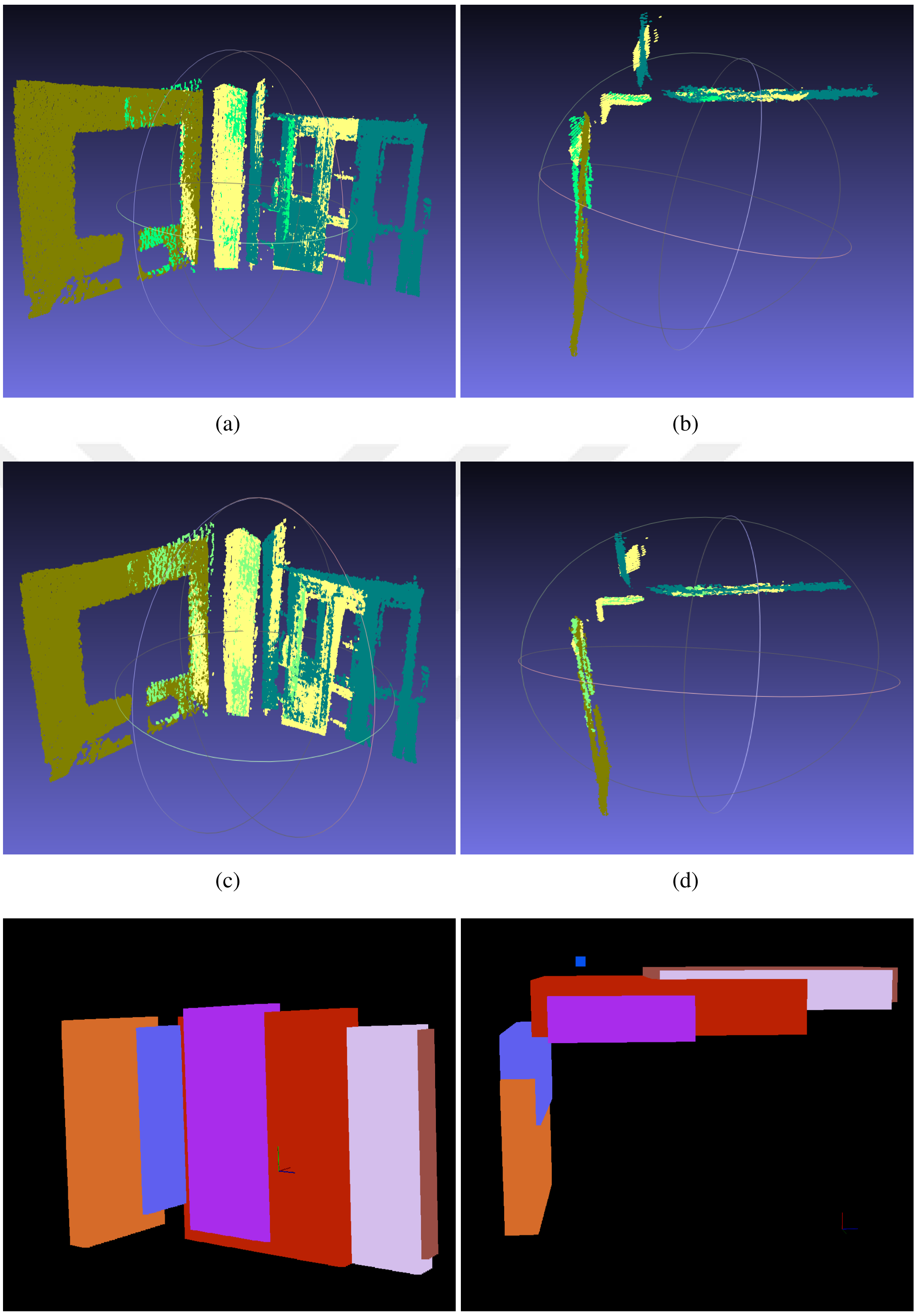

(a)
(b)
(c)
(d)
(e)
(f)

Figure A.17. Scene 17 images where (a)-(b) are crude registrations, (c)-(d) are fine registrations, (e)-(f) are plane representations of the frames