Kinstruct: Simultaneous Localization And Mapping Of Indoor Environments using Kinect

Author's name

Date

Acknowledgements

Contents

List of Figures

Chapter 1

Introduction

1.1 Overview

The problem of using cameras to build 3D models of real life environments is an extremely interesting problem that has been under investigation for many years, the reason for that is that many applications can be built in several fields using such a technology, also advances in both cameras and hardware equipments encourage researchers and developers to take the problem to new levels building models more accurately and faster

In our project, we use the commerically well-known and available Kinect to build Simultaneous Localization and Mapping system known as SLAM, this system allows a user to build 3D models of an indoor environment using a hand-held Kinect.

1.2 History

2D images obtained from ordinary cameras have been widely used to build 3D models of real life objects using the well known pipeline of Structure from Motion (SFM), but challenges exist in constructing models with accurate real distances maintained.

Anyway the introduction of laser scanners has provided more accurate information about the nature of scenes and objects especially surfaces with complex structure, the complexity of laser scanners and their high cost that can be up to tens of thousands of dollars make them only suitable for professionals.

Between the ordinary 2D camers and the expensive laser scanners, new cheap cameras (costs only hundreds of dollars) named RGB-D cameras have emerged in the market starting from 2009 such as the Kinect camera that was named initialy Project Natal, and although the Kinect was targeting the gaming industry, a wave of hacks exploiting the capabilities of Kinect in different fields such as 3D reconstruction has proved that despite the low sepecifications of the device it still can be used to produce results are useful for many users.

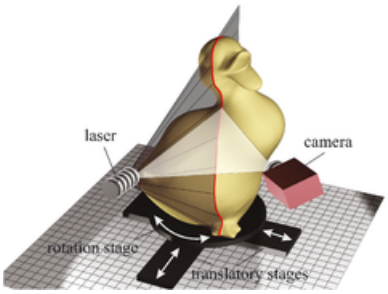

Figure 1.1: Using Laser Scanners

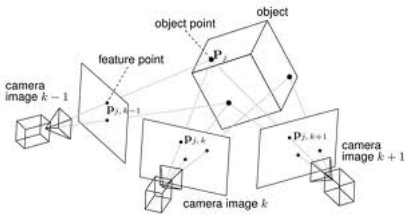

Figure 1.2: Structure From Motion

1.3 Organization of this Book

This book is divided into the following 11 chapters:

Chapter 1: Introduction - introduces the problem we are trying to solve and brief glimpse of related work

Chapter 2: Background - provides a formal definition of SLAM problem and gives an overview on the suggested pipeline explained in detail in later chapters

Chapter 3: System Analysis and Design - explains the top level analysis of the system and how components integrate using UML diagrams

Chapter 4: Using Kinect - explains our choice of Kinect as the scanning camera and details its sepecifications and history

Chapter 5: Feature Extraction and Matching - compares two different methods that we used to track features from consecutive frames

Chapter 6: Camera Tranformation Calculation - explains the methods used to calculate a transformation between the correspondences obtained from feature matching

Chapter 7: Loop Closure and Global Optimization - explains how we solved the problem of scene revisiting and accumlative error handling

Chapter 8: Refinement and Surface construction - discusses the different techniques that are used in enhancing the build 3D model

Chapter 9: Segmentation - demonstrates applying segmentation can be used to perform editing on the built 3D models

Chapter 10: User Interface - describes the User Interface that a user uses to perform the different tasks the system provides

Chapter 11: Conclusions and Future Work - explains our conclusions and how this project can develop further in the future

Chapter 2

Background

2.1 Motivation

3D scene reconstruction from 2D images has been an old and challenging problem. The task of computer vision and image processing is to be able to bring sight to the computer and provide it with vision analysis.

Being able to restore the depth information of an image and recreate the Original 3D scene from images alone has many applications in computer vision.

While reconstruction of 3D scenes can also be accomplished through the use of specialized hardware such as laser scanners, our focus will be on reconstructing 3D scene using images and photographs captured from KINECT. We will give a more detailed description for KINECT later in a separate chapter.

The main idea that was used in previous similar projects was to retrieve the lost third dimension from the 2D dimension multiple photos from different prospective and positions to build the 3D model for the required objects from this set of 2D photos.

The reconstruction of a dynamic, complex 3D scene from multiple images has been a fundamental problem in the field of computer vision. Given a set of images of a 3D scene, in order to recover the lost third dimension, depth, it is necessary to compute the relationship between images through correspondence. By finding corresponding primitives such as points, edges or regions between the images, such that the matching image points all originate from the same 3D scene point, knowledge of the camera geometry can be combined in order

to reconstruct the original 3D surface.

2.2 Applications

- 3D face recognition and construction from set of 2D images for a certain person from different perspectives.
- Teleconferencing requires a complete 3D world to be reconstructed.
- Interactive visualization of remote environments by a virtual camera
- Virtual modification of a real scene for augmented reality tasks
- Building 3D maps for the locations that the robot exists in to help in robot navigation
- Multimedia computing to generate new virtual views of scenes
- Virtual reality
- Simulation of show cases in crimes.
- Advertising and easy building of 3D models for different products.
- Indoor environment construction that helps in the field of decoration and furniture arrangements inside buildings.
- Games that depends on dynamic recognizing of the surrounding environment.
- Building panorama images for tourism issues.

2.3 Previous work

In this section we will introduce the methods that were being used and are used now for 3D reconstruction from stereo algorithms that can operate on building 3D scenes using 2D images.

The main problem as we mentioned before is to get the 3D dimension z from multiple 2D images (x, y) for the same scene. The algorithms introduced here help the researches in the field of 3D

construction to do that using special cameras like laser cameras or other high quality cameras.

While stereo algorithms require scene elements to be mostly visible from both cameras, volumetric methods can handle multiple views where very few scene elements are visible from every camera. A survey of methods for volumetric scene reconstruction from multiple images is presented in this section.

All the methods described here for building 3D scene models assume accurately calibrated cameras or images that are taken at known viewpoints. This is necessary in order to recover the absolute relationship between points in space and visual rays, so that Voxels in the object scene space can be projected to its corresponding pixels in each image. Image calibration and the computation of the projection matrix is itself a very challenging problem and a large number of literatures have been devoted to the recovery of camera geometry.

Also, it is assumed that the surfaces reflect light equally in all directions, such that the radiance observed of a 3D point is independent of its viewing direction.

A common approach to stereo reconstruction is the optimization of a cost function, computed by solving the correspondence problem between the set of input images. The matching problem involves establishing correspondences between the views available and is usually solved by setting up a matching functional for which one then tries to find the extreme. By identifying the matching pixels in the two images as being the projection of the same scene point, the 3D point can then be reconstructed by triangulation, intersecting the corresponding optical rays.

The proposed method to recover the 3D structure is a combination of volumetric scene reconstruction techniques and energy minimization techniques. Assuming a pinhole camera model, the 3D voxel volume is created by projecting all of the images into a 3D polyhedron, such that each voxel contains a feature vector of all the information contained in each camera view. A feature vector can include, for example, the RGB values of the voxel's projection into each camera image, the gradient of the image or information relating to the projected pixel's neighborhood.

The used reconstruction algorithm can be split into two separate modules and the rest of this chapter will be devoted to a detailed

description of each of these modules:

1. The first module is the computation of the projective Voxel volume. Camera calibration and the recovery of the camera geometry is required in order to determine the projection camera matrices for each camera necessary for projection. (3D Voxel Volume)

2. The second module involves the computation of the metric volume. (Metric Volume)

Here are the details of the two modules:

1. 3D Voxel Volume (Camera Calibration and the P-Matrix) :

   Consider the projection of a 3D point in world space onto an image. Assuming a pinhole camera model and setting center of projection as the origin of a Euclidean coordinate system with the image plane placed at $z = f$, we obtain the configuration in figure 2.1. By similar triangles, we can see that a 3D point $P = (x, y, z)^T$ is mapped onto the image at image coordinate p $(x, y, z)^T \rightarrow (fx/z, fy/z)^T$

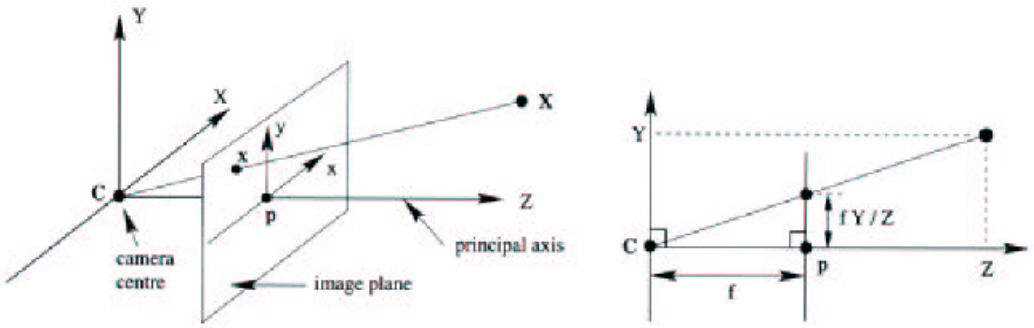

Figure 2.1: The pinhole camera model

   Represented as homogeneous vectors, the mapping from Euclidean 3-space R3 to Euclidean 2-space R2 can be expressed

in matrix multiplication as

$$\begin{pmatrix} fx \\ fy \\ z \end{pmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

The above formulation assumes that the origin of coordinates in the image plane coincides with the principal point, to account for this offset, where the coordinate of the principal point occurs at $(x_0, y_0)^T$, the mapping

$$(x, y, z)^T \rightarrow (fx/z + x_0, fy/z + y_0)^T$$

can be rewritten as

$$\begin{pmatrix} fx + zx_0 \\ fy + zy_0 \\ z \end{pmatrix} = \begin{bmatrix} f & 0 & x_0 & 0 \\ 0 & f & y_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

If we define a matrix K, known as the intrinsic camera matrix since it describes the internal camera parameters,

$$K = \begin{bmatrix} f & 0 & x_0 \\ 1 & f & y_0 \\ 1 & 0 & 1 \end{bmatrix}$$

Then the mapping from R3 to R2 can be written as

$$p = K[I|0]P$$

Furthermore in general, the camera coordinate system is embedded inside a world coordinate frame and the origin of the camera center, C, does not necessary coincides with the world coordinate origin. We realize the the P that we have been referring so far is expressed with respect to the camera coordinate system, and computations are generally performed with respect to the world coordinate system. The 3D point P relates to the world coordinate system by $P = R(P_w - C)$, where R is the rotation matrix representing the orientation of the camera coordinate frame. In matrix form this would become

$$P = \begin{bmatrix} R & -RC \\ 0 & 1 \end{bmatrix} P_w$$

Combining this with the intrinsic camera matrix, we obtain

$$P = KR[I| - C]P_w$$

The term $KR[I| - C]$ is the camera projection matrix. It is however more convenient not to explicitly describe the camera center and represent the transformation between the coordinate system as a rotation followed by a translation, giving rise to the more common form of the projection matrix P $P = K[R|t]$

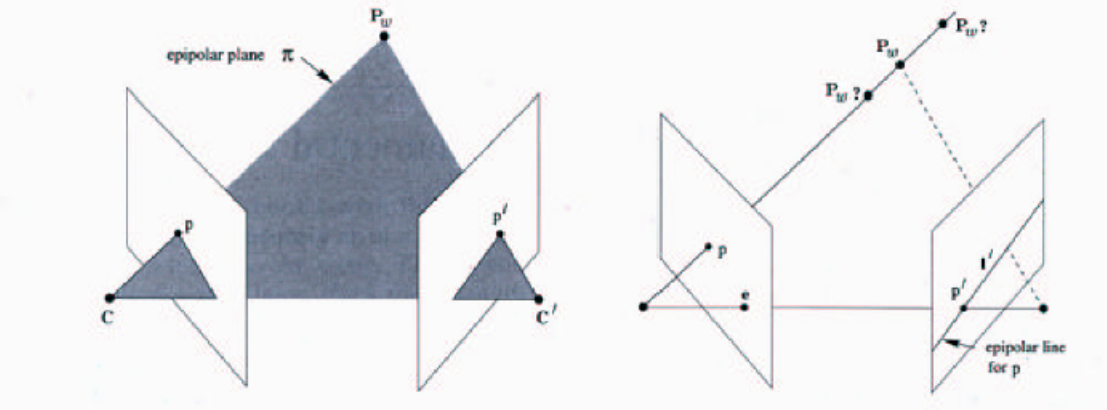

Figure 2.2: Epipolar Geometry

Here is a flowchart of mappings from the Voxel Volume to Image Pixels

In our formulation, we will assume a pinhole camera model and that all surfaces are Lambertian (i.e. the radiance observed of a 3D point is independent of viewing direction). The projective coordinate of a 3D point Pw in world space is expressed with homogeneous coordinates as

$$P_w = \begin{bmatrix} x_w & y_w & z_w \end{bmatrix}^T$$

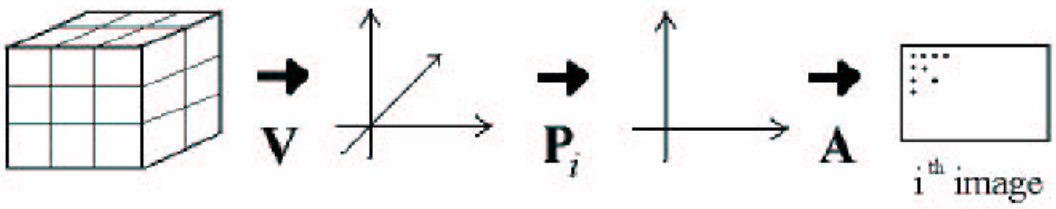

Figure 2.3: A flowchart of mappings from the Voxel Volume to Image Pixels

while the projective image coordinate of a pixel in image i is

$$p_i = \begin{bmatrix} x_i & y_i & z_i & 1 \end{bmatrix}^T$$

such that the corresponding pixel coordinate p'i of the projected point pi can be obtained by applying a homogenising function H where

$$H(\begin{bmatrix} x \\ y \\ z \end{bmatrix}) = \begin{bmatrix} x/z \\ y/z \end{bmatrix}$$

Results:

2. Metric Volume:

   After the projective volume is determined, a metric volume condensing the feature vectors of each voxel into a meaning measure or matching functional is required. While it is very important for the camera calibration process to determine the correct camera projection matrices so that the projective volume can be correct constructed, it is also very important to select an appropriate metric such that the correct measure can be computed. Accurate projections provide the necessary information for each voxel, so that each voxel can confidently locate the pixels from which it is back-projected to. Given all the information, feature vectors, it is up to the metric volume stage to analyze the obtained information and to decide whether a

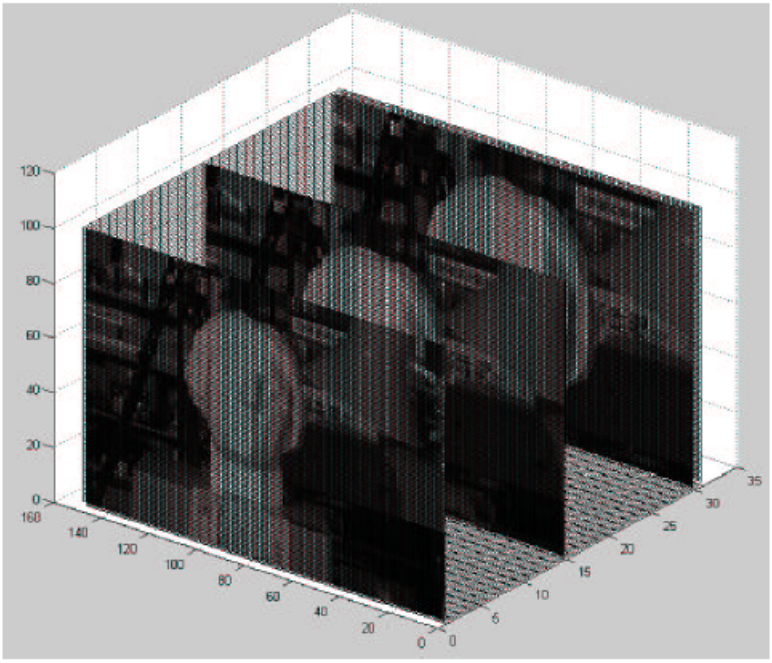

Figure 2.4: Projective volume projection for one of the image in the 'Head and Lamp' sequence

voxel is part of the 3D object scene. Volumetric scene reconstruction techniques recover depth information by labeling a voxel with either transparent or opaque, or in the case of a ternary decision, as either transparent, opaque or unseen. In the end, the most important task of the reconstruction is this decision. Thus the metric measure has the important task of making this decision through the analysis of the information available. Although our proposed reconstruction technique delays this decision making till the segmentation stage, since segmentation is accomplished through minimization of an energy function, it is still extremely important for the metric volume to derive a meaning measure of cost so that the 3D true scene surface is the global minima.

2.4 Similar Projects

2.5 What is different with our project?

Chapter 3

System Analysis and Design

In this chapter we explain the system requirements, we also provide an analysis of the system architecture along with a description of each module.

3.1 System Requirements

Our system should build a 3D model of an indoor environment scanned by a user holding Kinect camera, the system must work in real-time meaning that the model will be constructed while the user is scanning the environment, this will help the user see what parts of the environment are missing and identify any mistakes if any.

Our main focus in this system is for it to be available for ordinary users that may not have access to expensive hardware and may not be experienced in 3D scanning. For this reason, we used techniques that give good results while not needing much processing, we also handle any mistakes commited by users while moving the Kinect such as moving fast.

To prove the usefulness of the system, we enable the user to do simple editing on the constructed model, this simple editing includes moving and deleting some objects from the scene.

3.2 Main pipeline and architecture

SLAM systems can be described as iterative systems as the second frame is aligned to the first frame, then the operation is repeated for each new frame.

We explain briefly the main steps of the pipeline, each step is explained in more details in following chapters:

3.2.1 Frame Acquisition

We use the OpenNI library to obtain the current viewed scene by Kinect, this scene is represented as a 3D point cloud, for each point we have the x, y and z coordinates in meters along with the RGB components of the pixel color. The captured point cloud is then converted into two images; one representing the colored scene and another containing the depth information for each pixel, this step is done so that we can use these images later for feature tracking.

3.2.2 Feature Tracking

To align any two consecutive frames, we need to obtain the transformation between between the two frames, this can be achieved by tracking feature points from one frame to another, then use transformation calculations to obtain the best transformation between the two set of points. In chapter 5 we talk in details about two different methods for feature tracking and compare their results.

3.2.3 Transformation Calculation

After having two sets of correspondence between the two consecutive frames, we calculate the transformation using both closed form method known as Horn's method followed by Iterative Closes Point procedure to enhance the results and get a more accurate transformation.

3.2.4 Concatenation and Presenting Results

Having obtained the required transformation between the two frames, the second is transformed and concatenated to the global point cloud that is viewed by the user. The results of construction can be also saved using the popular .ply format as a mesh or a point cloud.

3.2.5 Loop Closure and Global Optimization

At the end of the scanning operation, it is likely that the user will return to the starting point, theoritcally the first and last frames should concide over each other, but due to both cumulative errors in alignment and numerical errors the alignment of the first and last frame will not be exact, so we need to detect a loop in the scanning and try to make optimize the global scene to distribute the error.

The following figure summarizes the general pipeline:

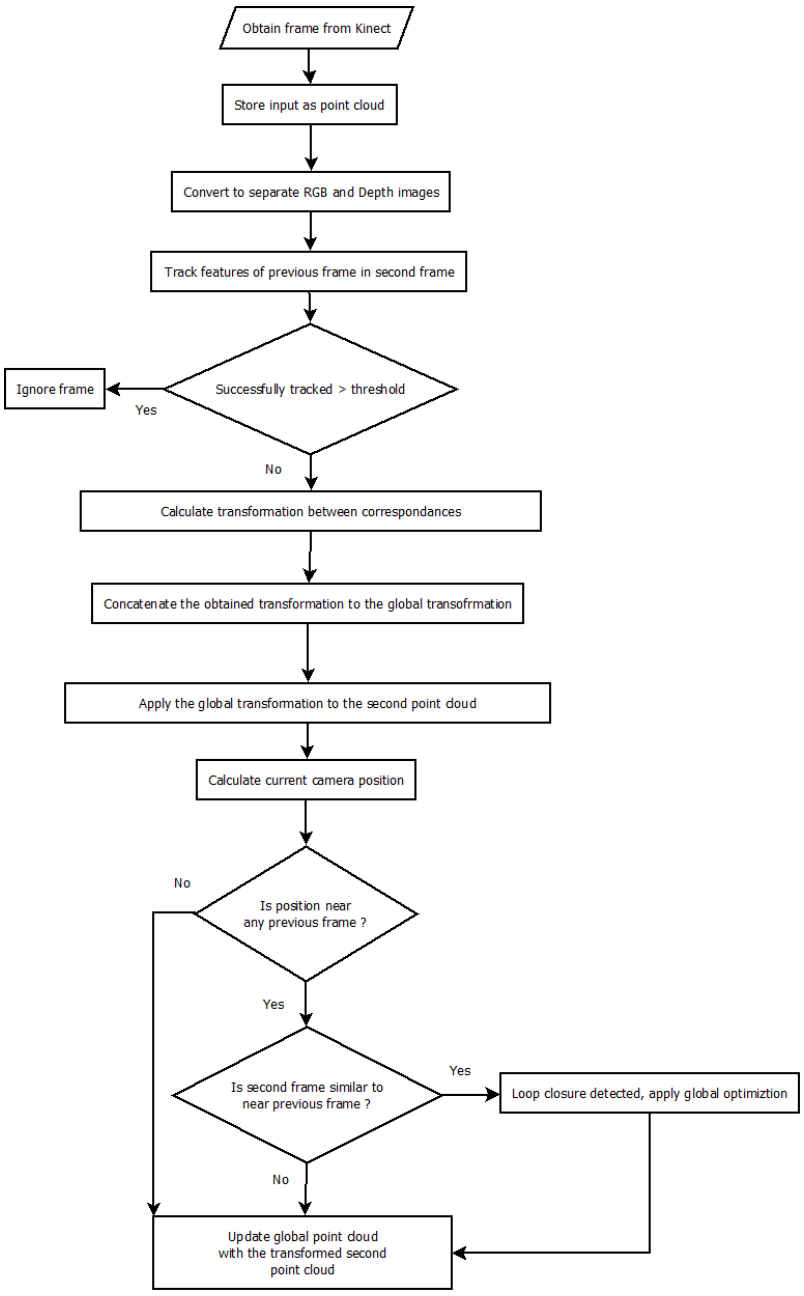

Figure 3.1: System Pipeline

3.3 System components

We divide our system to a number of components, each component is responsible for doing only one function, we tried to make the design as cohesive as possible and reduce the coupling between components as possible.

Each component is implemented in a class that offers methods that are used by other components, here is a list of classes and their descriptions:

- Transformation: a data class that represents a 3D transformation.
- HornMethod: this class contains the code needed to calculate the best transformation between two sets of 3D points.
- Tracker: this class provides feature tracking using optical flow and SURF feature detection and matching.
- Alignment: this class has functions that use the previous components to obtain an initial transformation between two consecutive frames.
- SurfaceConstruction:
- Segmentation:

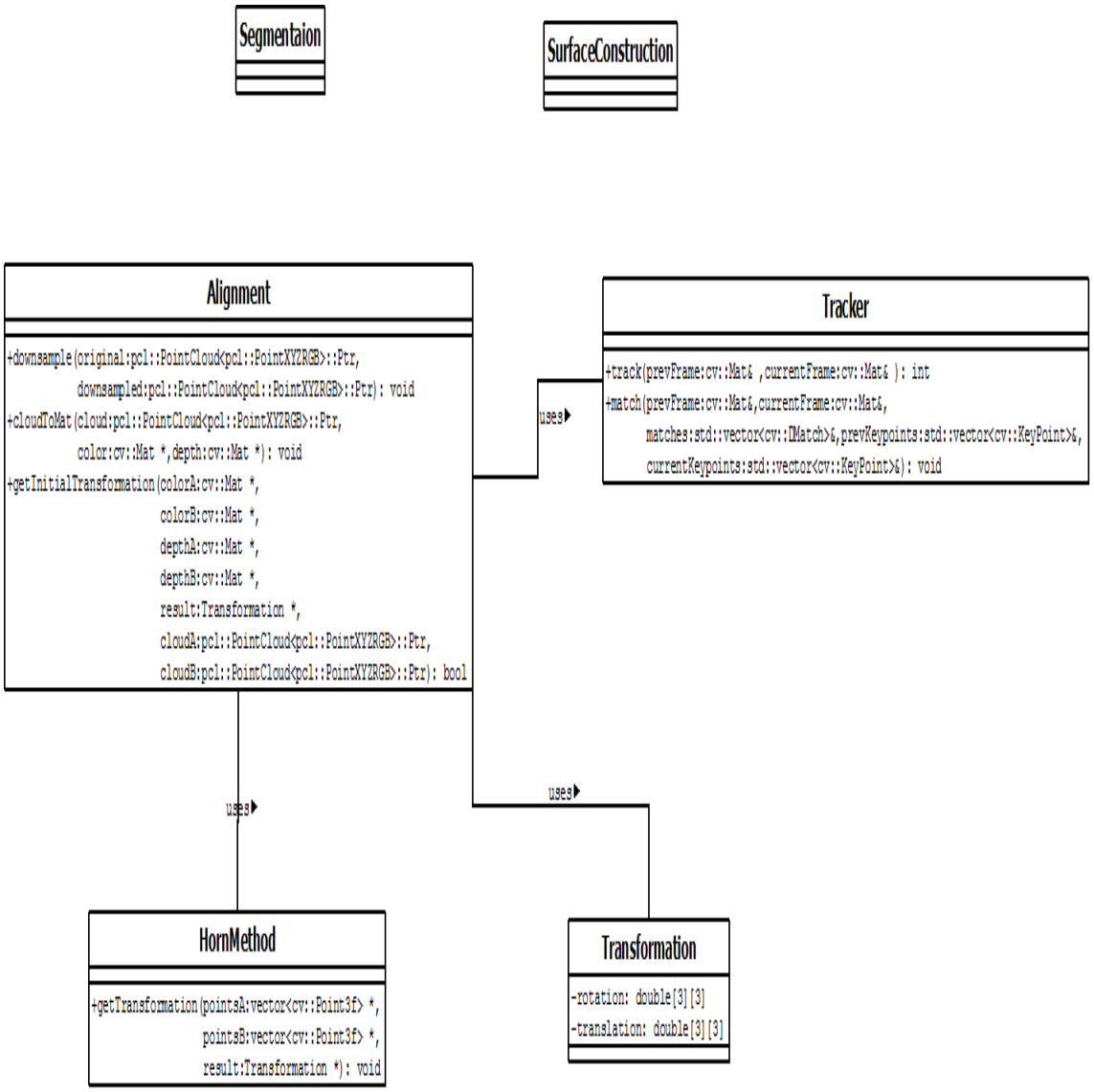

Figure 3.2: Class Diagram

3.4 Libraries and tools

It is well known that SLAM systems are expensive both in computation and memory usage, this is why it was necessary to choose a development path that will achieve both. We have chosen C++ as our programming language as it is the fastest high level programming language and is supported by many projects such as OpenCV and Point Cloud Library (PCL).

3.4.1 OpenCV

OpenCV is a library of programming functions mainly aimed at real time computer vision, developed by Intel and now supported by Willow Garage. It is free for use under the open source BSD license. The library is cross-platform. It focuses mainly on real-time image processing. If the library finds Intel's Integrated Performance Primitives on the system, it will use these proprietary optimized routines to accelerate itself.

We use the C++ implementaion of OpenCV extensively in our project, we use it for storing images in data structures, feature extraction and tracking, RANSAC operations and image viewing.

3.4.2 PCL

PCL is a large scale, open project for 3D point cloud processing. The PCL framework contains numerous state-of-the art algorithms including filtering, feature estimation, surface reconstruction, registration, model fitting and segmentation. These algorithms can be used, for example, to filter outliers from noisy data, stitch 3D point clouds together, segment relevant parts of a scene, extract keypoints and compute descriptors to recognize objects in the world based on their geometric appearance, and create surfaces from point clouds and visualize them – to name a few.

We use PCL to store and view 3D point clouds, use its Iterative Closest Point implementation, use its Explicit Loop Closing Heuristic, also it contains useful data structures such as k-d trees.

Chapter 4

Kinect Device

4.1 Introduction

Kinect is a motion sensing input device by Microsoft for the Xbox 360 video game console and Windows PCs. Based around a webcam-style add-on peripheral for the Xbox 360 console, it enables users to control and interact with the Xbox 360 without the need to touch a game controller, through a natural user interface using gestures and spoken commands. Kinect was launched in North America on November 4, 2010, in Europe on November 10, 2010, in Australia, New Zealand and Singapore on November 18, 2010, and in Japan on November 20, 2010. Microsoft released Kinect software development kit for Windows 7 on June 16, 2011. This SDK allows developers to write Kinecting apps in C++/CLI, C# or Visual Basic .NET

KINECT
for XBOX 360.

Figure 4.1: Kinect for Xbox 360

4.2 Technology

Kinect builds on software technology developed internally by Rare, a subsidiary of Microsoft Game Studios owned by Microsoft, and on range camera technology, which developed a system that can interpret specific gestures, making completely hands-free control of electronic devices possible by using an infrared projector and camera and a special microchip to track the movement of objects and individuals in three dimension. This 3D scanner system called Light Coding employs a variant of image-based 3D reconstruction. The depth sensor consists of an infrared laser projector combined with a monochrome CMOS sensor, which captures video data in 3D under any ambient light conditions. The sensing range of the depth sensor is adjustable, and the Kinect software is capable of automatically calibrating the sensor based on gameplay and the player's physical environment, accommodating for the presence of furniture or other obstacles.

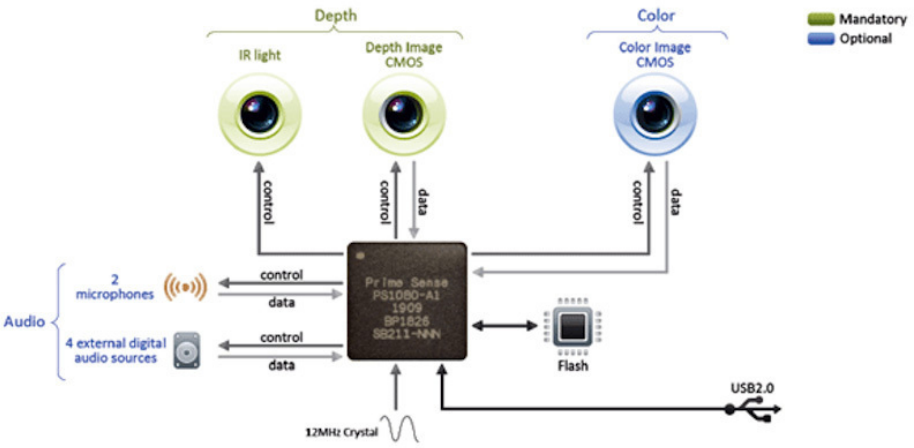

Figure 4.2: Underlying technology of Kinect

4.3 Depth Image

Kinect captures a sequence of two types of synchronized frames, an RGB Image and a depth Image, the synchronization between the two images ensures that each RGB image has a correspondent depth image. Using information obtained from each two synchronized frames, we can obtain for each pixel:

- RGB color information
- x ,y and z co-ordinates

RGB image is a simple 640×480 pixels image Depth image is a 640×480 pixels grayscale image where:

- Bright colors represent close objects
- Dark colors represent far objects
- Black segments represent objects out of range "too close or too far"

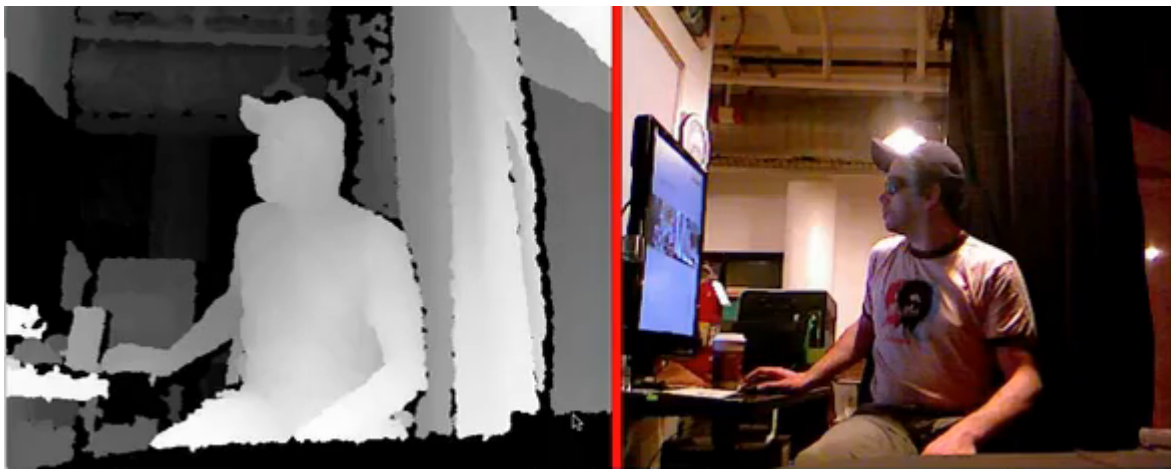

Figure 4.3: A figure showing depth color on the left with its correspondent RGB image, the brighter the pixel the closer it is, while out of range pixels are represented in black

4.4 Device Specifications

Sensor	Colour and depth-sensing lenses Voice microphone array Tilt motor for sensor adjustment Fully compatible with existing Xbox 360 consoles
Field of View	Horizontal field of view: 57 degrees Vertical field of view: 43 degrees Physical tilt range: 27 degrees Depth sensor range: 1.2m - 3.5m
Skeletal Tracking System	Tracks up to 6 people, including 2 active players Tracks 20 joints per active player Ability to map active players to LIVE Avatars
Audio System	LIVE party chat and in-game voice chat Echo cancellation system enhances voice input Speech recognition in multiple
Camera resolution	640 * 480 320 * 240
Frame rate	30 frames per second

Table 4.1: Kinect Specifications

4.5 Why Kinect?

4.5.1 Capturing the depth

As mentioned earlier Kinstruct is all about capturing a sequence of frames for a scene and building a 3D model from the processing of those frames, if we have information about x ,y and z co-ordinates for each pixel in a frame in the real world, then it would be much more easier to align frames and distinguish between overlapping objects.

4.5.2 Price and Availability

The preference of the Kinect sensor comes from its affordable price according to other distance detectors that uses different technologies, also the Kinect functions as both a camera and a depth sensor, so there's no need to afford two different hardware devices.

Device	Kinect	Laser sensor
Cost	$150	$45000 - $60000
Availability	Electronics store	Need to order
Programmable interface	High level	Low level

Table 4.2: Kinect Specifications

4.6 Precision of the Kinect sensor

Because the Kinect is essentially a stereo camera, the expected error on its depth measurements is proportional to the distance squared.

The experimental data shown in the graph below confirm the expected error model. The graph was obtained by pointing the Kinect at a planar surface, fitting a plane (using RANSAC) through the measured point cloud, and checking the distance of the points in the point cloud to that plane. This means the plane represents the average distance measurement, and the graph shows how far off the Kinect measurements are from that average distance.

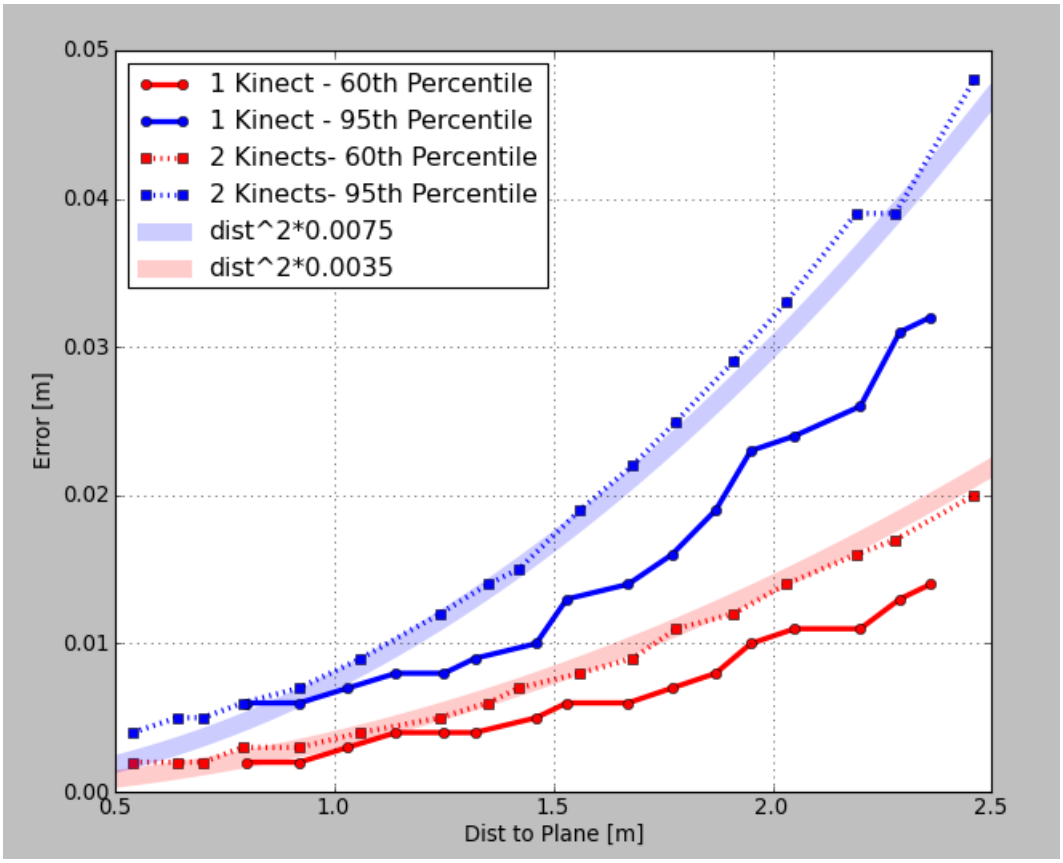

Figure 4.4: Precision of the Kinect measuring depth

4.7 Kinect SDKs

4.7.1 Kinect SDK for Windows

The Kinect for Windows SDK enables developers to create applications using C++, C# or Visual Basic, which support gesture and voice recognition using the Kinect for Windows sensor and a PC or embedded device.

It includes drivers for using Kinect sensor on a computer running Windows 7, Windows 8 Consumer Preview, and Windows Embedded Standard 7. In addition, the download includes application programming interfaces (APIs) and device interfaces.

4.7.2 Open Kinect

OpenKinect is an open community of people interested in making use of the Xbox Kinect hardware with our PCs and other devices. Working on free, open source libraries that will enable the Kinect to be used with Windows, Linux, and Mac.

4.7.3 Open NI

OpenNI or Open Natural Interaction is an industry-led, non-profit organization focused on certifying and improving interoperability of natural user interface and organic user interface for natural interaction devices, applications that use those devices and middleware that facilitates access and use of such devices.

The organisation has been created on November 2010, with the website going public on December 8th. One of the main members is PrimeSense, the company behind the technology used in the Kinect, a motion sensing input device by Microsoft for the Xbox 360 video game console.

In December 2010, PrimeSense, whose depth sensing reference design Kinect is based on, released their own open source drivers along with motion tracking middleware called NITE.PrimeSense later announced that it had teamed up with Asus to develop a PC-compatible device similar to Kinect, which will be called Wavi Xtion and is scheduled for release in the second quarter of 2012 Natural Interaction Devices or Natural Interfaces [6] are devices that capture body movements and sounds to allow for a more natural interaction

of users with computers in the context of a Natural user interface. The Kinect and Wavi X-tion are examples of such devices.

Chapter 5

Camera Transformation Calculation

5.1 Introduction

In the previous chapter, we explained how to extract feature points from a frame then find matches in the next frame. Now, we explain how we calculate a transformation between the two sets of feature points, this transformation is then applied on the second point cloud to align it with the global point cloud.

Tranformation calculation is a well known mathematical problem, where we calcluate a transformation matrix that when applied to the first set of points will result in the second set of points.

Some of the methods used for transformation calculation are iterative and others are closed form solutions, we have found that applying a closed form solution then using the results as a seed for the iterative solution gives better and faster results than applying only one method.

5.2 Horn's method

We Consider that problem of transforming the second point cloud to the first point cloud identical to the problem of transforming one coordinate system to another, this problem is named the Absolute Orientation Problem, we used the closed form solution proposed by Horn.

The transformation between the two coordinate systems is assumed to be rotation and translation, we denote it as $F_{AB} = [R|t]$

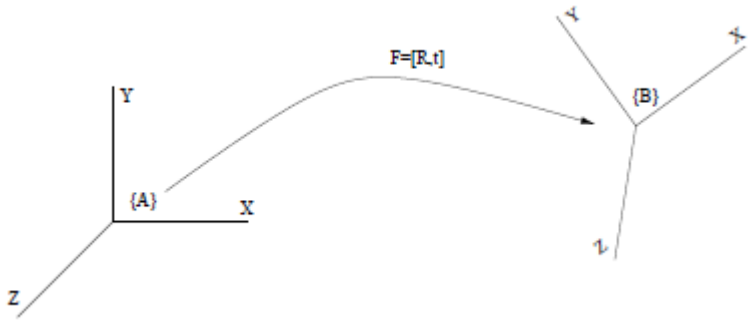

Figure 5.1: One coordinate system transformed to the other

5.2.1 Quaternions

Quaternions are a mathematical object of the form

$$Q = s + ix + jy + kz = s + v$$

where s, x, y, z ∈ R, and i, j, k are mutually orthogonal imaginary units with the following composition rule

	i	j	k
i	-1	k	-j
j	-k	-1	i
k	j	-i	-1

Table 5.1: i,j and k composition rule

We now define several basic operations on quaternions:

- Addition and Substraction: Given two quaternions $q_1 = (s_1, v_1)$ and $q_2 = (s2, v2)$ their addition/subtraction is defined as

$$q = q_1 + q_2 = (s_1 + s_2, v_1 + v_2)$$

- Multiplication: Given two quaternions $q_1 = (s_1, v_1)$ and $q_2 = (s_2, v_2)$ their multiplication is defined as

$$q_1 * q_2 = (s_1 + v_1) * (s_2 + v_2) = s_1 s_2 + s_1 v_2 + s_2 v_1 + v_1 * v_2$$

Then compute:

$$\begin{aligned} v_1 * v_2 &= (iv_{1x} + jv_{1y} + kv_{1z}) * (iv_{2x} + jv_{2y} + kv_{2z}) \\ &= (-v_{1x}v_{2x} - v_{1y}v_{2y} - v_{1z}v_{2z}) + \\ &\quad i(v_{1y}v_{2z} - v_{1z}v_{2y}) + \\ &\quad j(v_{1z}v_{2x} - v_{1x}v_{2z}) + \\ &\quad k(v_{1x}v_{2y} - v_{1y}v_{2x}) \\ &= -(v_1 \cdot v_2) + (v_1 \times v_2) \end{aligned}$$

Then

$$q = q_1 * q_2 = [(s_1 s_2 v_1 \cdot v_2); (s_1 v_2 + s_2 v_1 + v_1 \times v_2)]$$

We have just defined quaternions and their basic operations, let's explain how they work as rotation operators

Let's first explain how rotations are expressed using Euler angles then we talk about the problem of gimbal lock and how quaternions avoid it.

Euler angles represent a 3D rotation as a rotation around the 3D axes x, y and z, so a rotation is expressed as:

$$R_{AB} = R_z(\omega_z)R_y(\omega_y)R_x(\omega_x)$$

$$= \begin{bmatrix} \cos(\omega_z) & -\sin(\omega_z) & 0 \\ \sin(\omega_z) & \cos(\omega_z) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\omega_y) & 0 & \sin(\omega_y) \\ 0 & 1 & 0 \\ \sin(\omega_y) & 0 & \cos(\omega_y) \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\omega_x) & \sin(\omega_x) \\ 0 & \sin(\omega_x) & \cos(\omega_x) \end{bmatrix}$$

$$= \begin{bmatrix} \cos(\omega_z)\cos(\omega_y) & \cos(\omega_z)\sin(\omega_y)\sin(\omega_x) - \sin(\omega_z)\cos(\omega_x) & \cos(\omega_z)\sin(\omega_y)\cos(\omega_x) + \sin(\omega_z)\sin(\omega_x) \\ \sin(\omega_z)\cos(\omega_y) & \sin(\omega_z)\sin(\omega_y)\sin(\omega_x) + \cos(\omega_z)\cos(\omega_x) & \sin(\omega_z)\sin(\omega_y)\cos(\omega_x) - \sin(\omega_z)\sin(\omega_x) \\ -\sin(\omega_y) & \cos(\omega_y)\sin(\omega_x) & \cos(\omega_y)\cos(\omega_x) \end{bmatrix}$$

if we have a rotation matrix and want to get the rotation angles around the axes we use the following relations:

$$\omega_y = \text{atan2}(-r_{31}, \sqrt{r_{11}^2 + r_{21}^2})$$
$$\omega_z = \text{atan2}(r_{21}/\cos\omega_y, r_{11}/\cos\omega_y)$$
$$\omega_x = \text{atan2}(r_{32}/\cos\omega_y, r_{33}/\cos\omega_y)$$

the problem of gimbal lock happens when a rotation causes two of the axes are in the same plane, for instance, if $\omega_y = \pi/2$ then we will not be able to extract the angles in the previous relations.

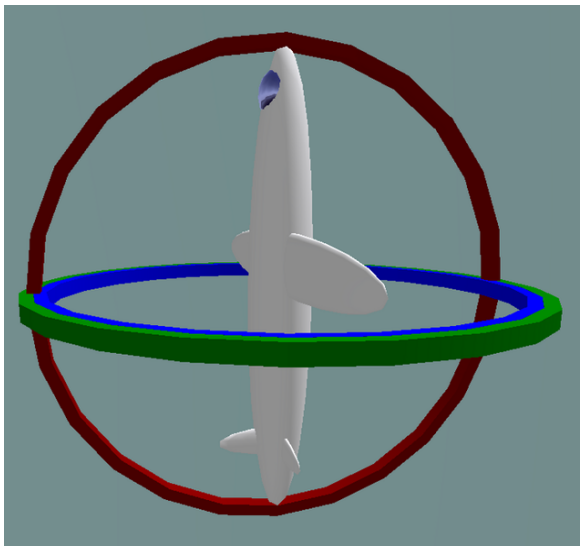

Figure 5.2: Gimbal lock happens when two rotation axes are in the same plane

this happens because Euler angles defines a 3D rotation as 3 rotations around the three axes, which causes a loss of degree of freedoms when two axes are in the same plane, this doesn't happen when we use quaternions because they define rotation around a unit vector not around the three axes x, y and z

The conjugate norm and inverse are given by the following three equations:

$$\bar{q} = [s, -v]$$
$$N(q) = \|q\| = \sqrt{q * \bar{q}} = \sqrt{s^2 + |v|^2}$$
$$q^{-1} = \frac{1}{q} = \frac{1}{q} * \frac{\bar{q}}{\bar{q}} = \frac{\bar{q}}{N(q)^2}$$

if $N(q) = 1$ then the quaternion is referred to as a unit quaternion and $\bar{q} = q^{-1}$

5.2.2 Quaternions as rotational operators

Given two vectors a and b and a unit quaternion of the form $q = [\cos\frac{\theta}{2}, n\sin\frac{\theta}{2}]$ the general rotation of a into b about an arbitrary unit axis n by θ radians is given by the equation:

$$b = q * a * q^{-1}$$

It can be proved that the previous equation is equivalent

5.2.3 Closed Form Registration

Given two sets of corresponding points in two different coordinate systems, we want to compute the transformation between the coordinate systems. This solution assumes we have at least three correspondences between the two coordinate systems.

For any two corresponding points p_l and p_r, the error in the tranformation can be defined as:

$$e_i = p_r - R(p_l) - t$$

We want to minimize the sum of square errors over all points:

$$\sum_{i=1}^{n} \|e_i\|^2$$

We will refer all measurements to the centroids given by:

$$\mu_l = \frac{1}{n}\sum_{i=1}^{n} p_{l,i}$$

$$\mu_r = \frac{1}{n}\sum_{i=1}^{n} p_{r,i}$$

The new coordinates are now:

$$p'_{l,i} = p_{l,i} - \mu_l$$

$$p'_{r,i} = p_{r,i} - \mu_r$$

the error term using the new coordinates becomes

$$e_i = p'_r - R(p'_l) - t'$$

where

$$t' = t - \mu_r + R\mu_l$$

The sum of square errors becomes

$$\sum_{i=1}^{n} \|e_i\|^2 = \sum_{i=1}^{n} \|p'_r - R(p'_l) - t'\|^2$$

$$= \sum_{i=1}^{n} \|p'_r - R(p'_l)\|^2 - 2t' \cdot \sum_{i=1}^{n} [p'_{r,i} - Rp'_{l,i}] + n\|t'\|^2$$

Because $\sum_{i=1}^{n} p'_{r,i} = \sum_{i=1}^{n} p'_{l,i} = 0$ we notice that the middle term in the previous equation equals zero, we minimize the last term by making it equal to zero then

$$t' = 0$$

$$\downarrow$$

$$t = \mu_r - R\mu_l$$

We have now to minimize the first term

$$\sum_{i=1}^{n} \|p'_r - R(p'_l)\|^2 = \sum_{i=1}^{n} \|p'_{r,i}\|^2 - 2\sum_{i=1}^{n} p'_{r,i} \cdot Rp'_{l,i} + \sum_{i=1}^{n} \|Rp'_{l,i}\|^2$$

The first and third terms of the previous equation are constants independent of R because rotations preserve the vector norm. We minimize the error by maximizing the second term.

This can be expressed in the following figure:

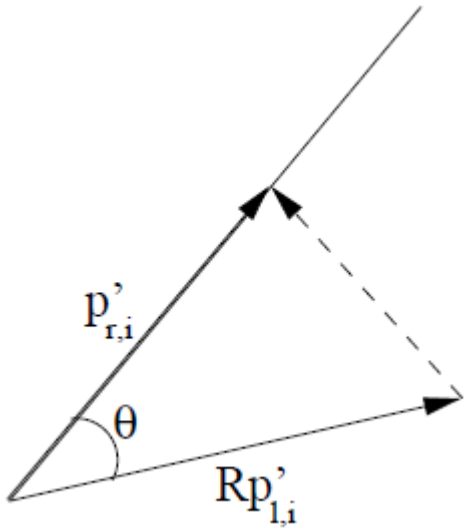

Figure 5.3: Maixmizing the sum $\sum i = 1^n p'_{r,i} \cdot Rp'_{l,i}$ is equivalent to maximizing $\sum i = 1^n |p'_{r,i}||Rp'_{l,i}| \cos\theta$. This sum is maximized when $\cos\theta = 1$, $\theta = 0$. Gemoetrically we compute the rotation which minimizes the angle between the two vectors

We express the equation using unit quaternions:

$$\sum i = 1^n (q * p'_{l,i} * \bar{q}) \cdot p'_{r,i}$$

$$= \sum i = 1^n (q * p'_{l,i}) \cdot (p'_{r,i} * q)$$

We use the matrix representation:

$$p'_{r,i} * q = \begin{bmatrix} 0 & -x'_{r,i} & -y'_{r,i} & -z'_{r,i} \\ x'_{r,i} & 0 & -z'_{r,i} & y'_{r,i} \\ y'_{r,i} & z'_{r,i} & 0 & -x'_{r,i} \\ z'_{r,i} & -y'_{r,i} & x'_{r,i} & 0 \end{bmatrix} q = \Re_{r,i} q$$

and

$$q * p'_{l,i} = \begin{bmatrix} 0 & -x'_{l,i} & -y'_{l,i} & -z'_{l,i} \\ x'_{l,i} & 0 & z'_{l,i} & -y'_{l,i} \\ y'_{l,i} & -z'_{l,i} & 0 & x'_{l,i} \\ z'_{l,i} & y'_{l,i} & -x'_{l,i} & 0 \end{bmatrix} q = \bar{\Re}_{l,i} q$$

So we have:

$$\sum i = 1^n (\bar{\Re}_{l,i} q) \cdot (\Re_{r,i} q)$$

$$\Downarrow$$

$$\sum i = 1^n q^T \bar{\Re}_{l,i}{}^T \Re_{r,i} q$$

$$\Downarrow$$

$$q^T (\sum i = 1^n \bar{\Re}_{l,i}{}^T \Re_{r,i}) q$$

$$\Downarrow$$

$$q^T (\sum i = 1^n N_i) q$$

$$\Downarrow$$

$$q^T N q$$

The matrix N is symmetric as it is a sum of symmetric matrices. The vector q which maximizes $q^T N q$ is the eigenvector corresponding to the most positive eigenvalue of the matrix N.

We construct the matrix N by:

$$M = \sum i = 1^n p'_{l,i} p' T_{r,i}$$

$$= \sum i = 1^n [p_{l,i} p_{r,i}^T] - \mu_l \mu_r^T$$

$$= \begin{bmatrix} S_{xx} & S_{xy} & S_{xz} \\ S_{yx} & S_{yy} & S_{yz} \\ S_{zx} & S_{zy} & S_{zz} \end{bmatrix}$$

where

$$S_{xx} \sum i = 1^n x'_{l,i} x'_{r,i}$$

$$S_{xy} \sum i = 1^n x'_{l,i} y'_{r,i}$$

and so on. The matrix contains all the information needed to construct the matrix N

$$N = \begin{bmatrix} trace(M) & \Delta^T \\ \Delta & M + M^t - trace(M) I_3 \end{bmatrix}$$

where

$$N = \begin{bmatrix} (M - M^T)_{23} \\ (M - M^T)_{31} \\ (M - M^T)_{12} \end{bmatrix}$$

5.2.4 Results of Horn's method

We have successfully implemented the previously explained method, and have reached results that (measure time of horn) We noticed that this method works well when:

- The two frames are rich in features, meaning they have a high (specific) number of matches.
- The features must also be well distributed over the image space, meaning that features are not only focused in one place in the two images.

if one or both of these conditions fail to exist, the results will not be correct and what happens is that the transformation calculated will be biased towards the feature points used in the calculations, this means that these features will be aligned well but the rest of the scene may or may not align in the right place.

This problem occurs a lot when the scanned scene contains a large continous space of the same surface like walls or floors, these surfaces don't contribute in the feature tracking thus are not considered in the transformation calculations, this is explained in the following images:

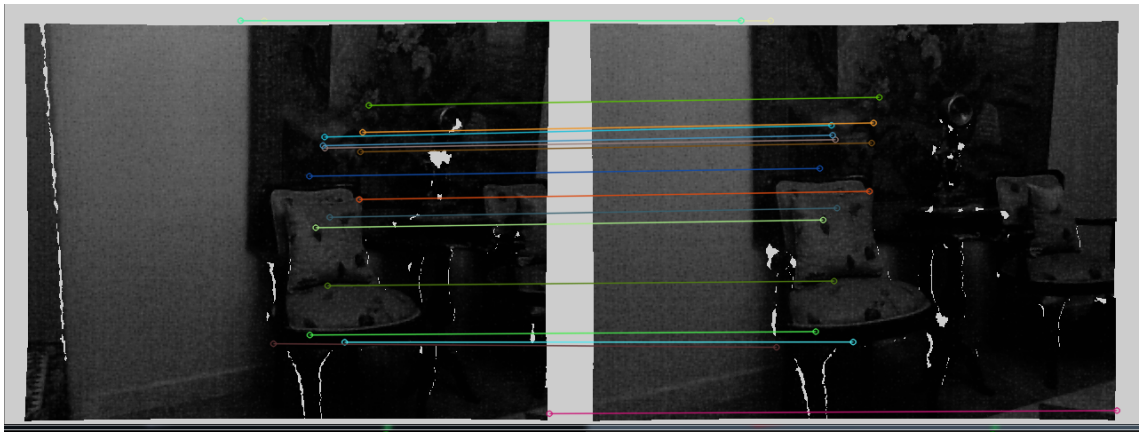

Figure 5.4: We notice that most features are concentrated in the parts that contains chairs while the wall on the right has no features matched

this causes the transformation to be baised towards the chairs while neglecting the alignment of the walls

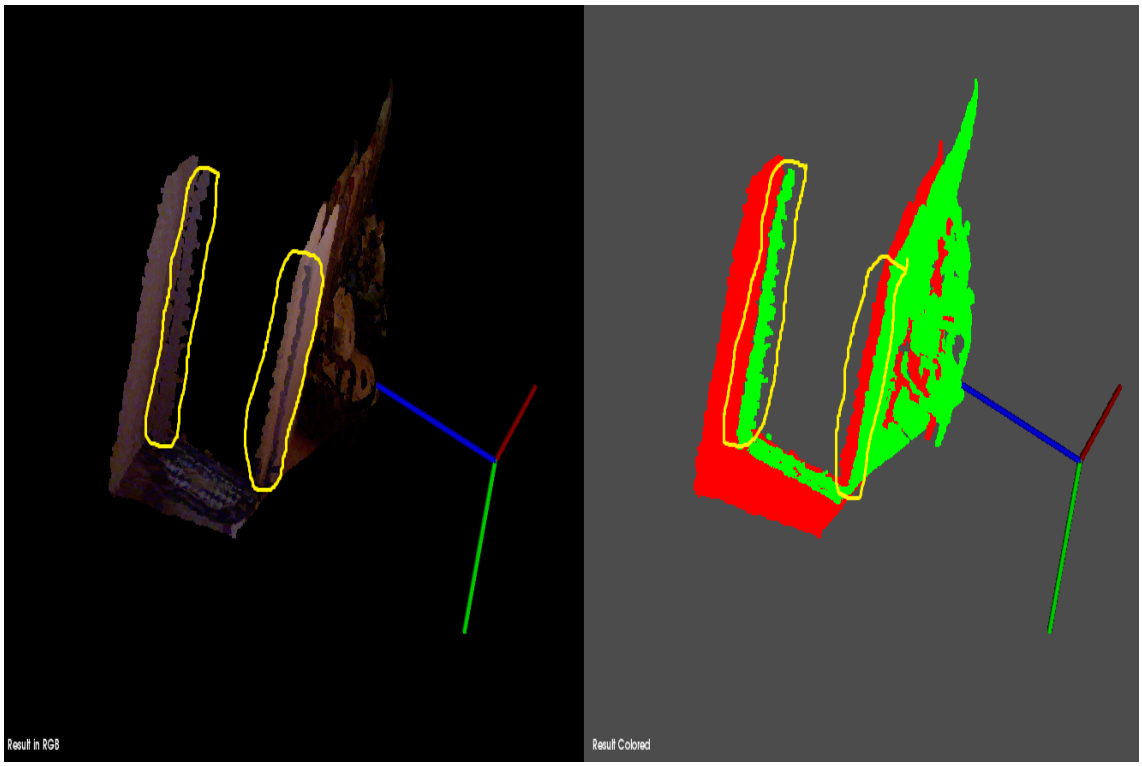

Figure 5.5: Areas marked with yellow show how alignment fails in featureless areas

As a result of the previous conclusion we used a different transformation estimation algorithm known as Iterative Closest Point (ICP), the following section explains how the algorithm works and the results we have reached.

5.3 Iterative Closes Point

5.3.1 Definition

The Horn's method explained in the previous section was a closed form solution, here we explain ICP which is an iterative algorithm that works as follows:

1. Find closest neighbour to every point
2. Compute transformation that minimizes the error over all pairs

of points

3. Apply transformation and update point cloud
4. Go back to step 1 till stopping criteria is met

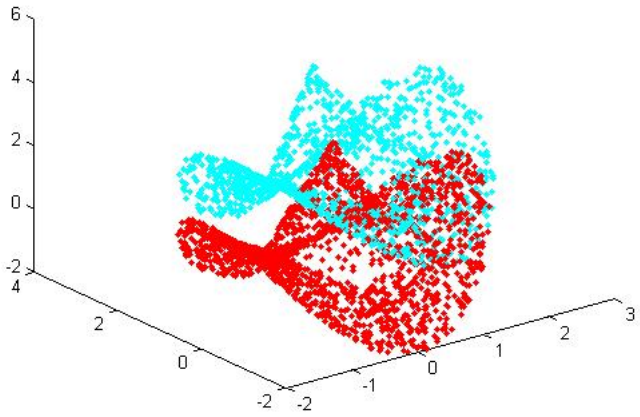

Figure 5.6: Iterative closest point used to align two point clouds

5.3.2 Motivation

ICP can be used to align two point clouds by iteratively computing the transformation that minimizes the sum of square errors, the problem is that ICP is suffers from:

- ICP may not converge so we need to set a stopping criteria.
- ICP may converge to a local minimum, meaning that we may get a transformation that is good but not the best one.
- ICP is extremely slow as it uses all the points of the point clouds, while Horn's method uses much less number of points, and can do with only three.

As stated in the previous chapter, the problem with the closed form solution is that it considers only feature points in transformation calculation, so we decided to use ICP as a second step after

obtaining the closed form solution, and we use the obtained transformation as an initial guess to the ICP, this way the ICP step takes less time and converges better, we also consider downsampling the input point clouds to reduce the input size of the ICP making it more faster.

5.3.3 Downsampling

Downsampling is used to reduce the number of points in a point cloud without losing much details.

We use voxel grids to downsample the input point clouds, voxel grids work by dividing the input cloud into small 3D boxes and then replacing every box with the average of all points inside it, the dimensions of the 3D boxes used determine the downsampling ratio, so if we use large boxes we get fewer points and vice versa.

We use a voxel grid of size $(2.5)^3$ cm^3, this allows us to reduce the number of points processed in ICP while not losing much of the details of the scene.

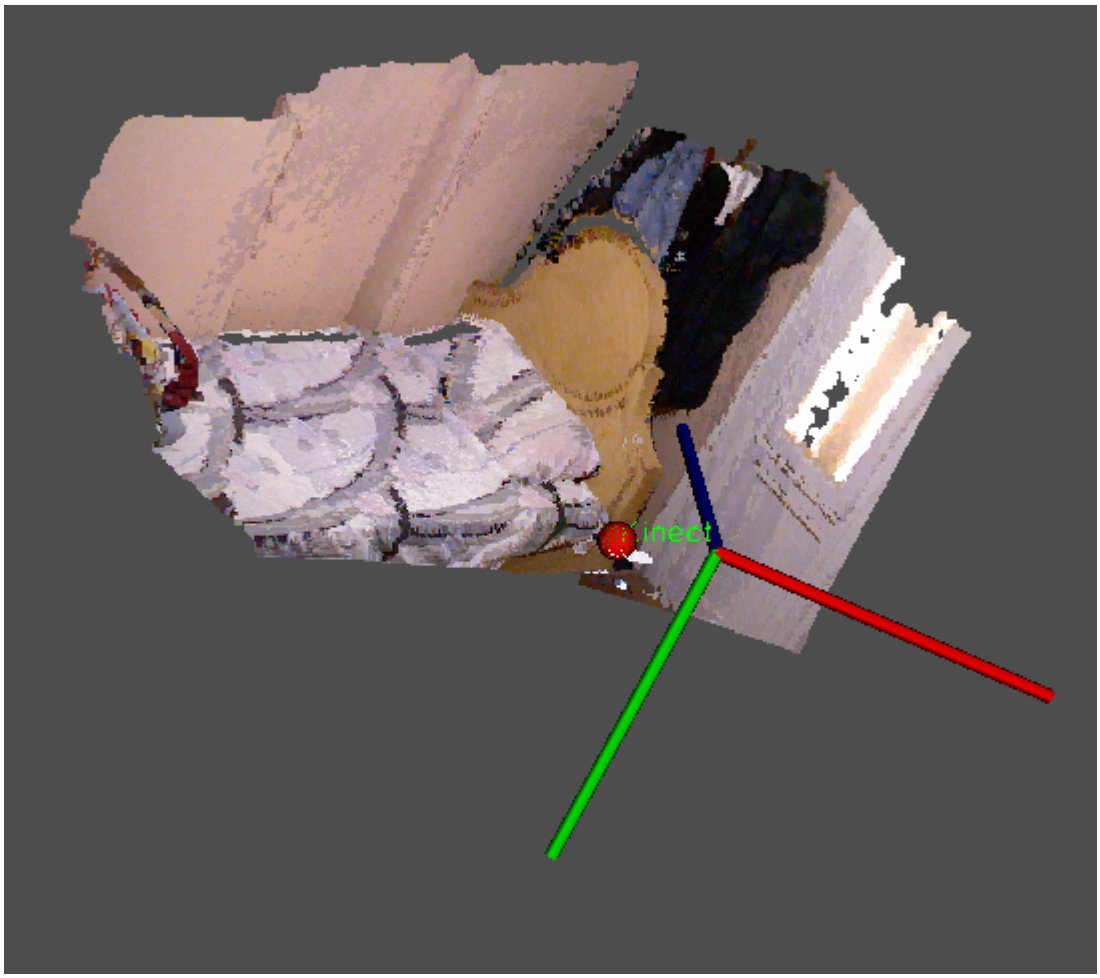

Figure 5.7: A QVGA scene of a bed without downsampling

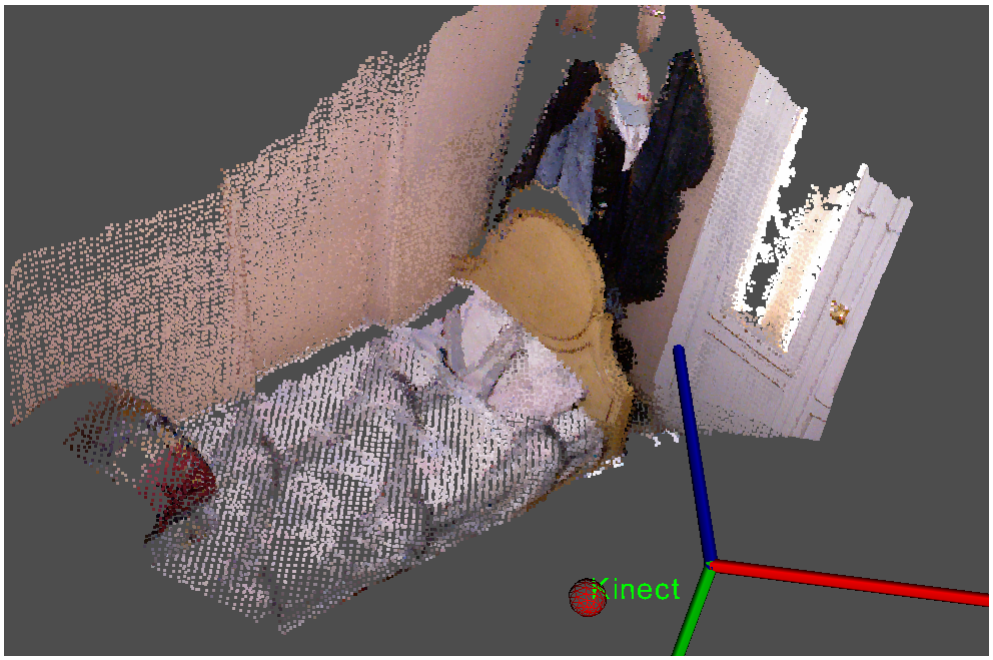

Figure 5.8: A QVGA scene of a bed after downsampling

5.3.4 Results of applying ICP

After using Horn's method transformation as an initial guess for the ICP procedure on the two downsampled point clouds we have obtained great results because the ICP takes all points in consideration.

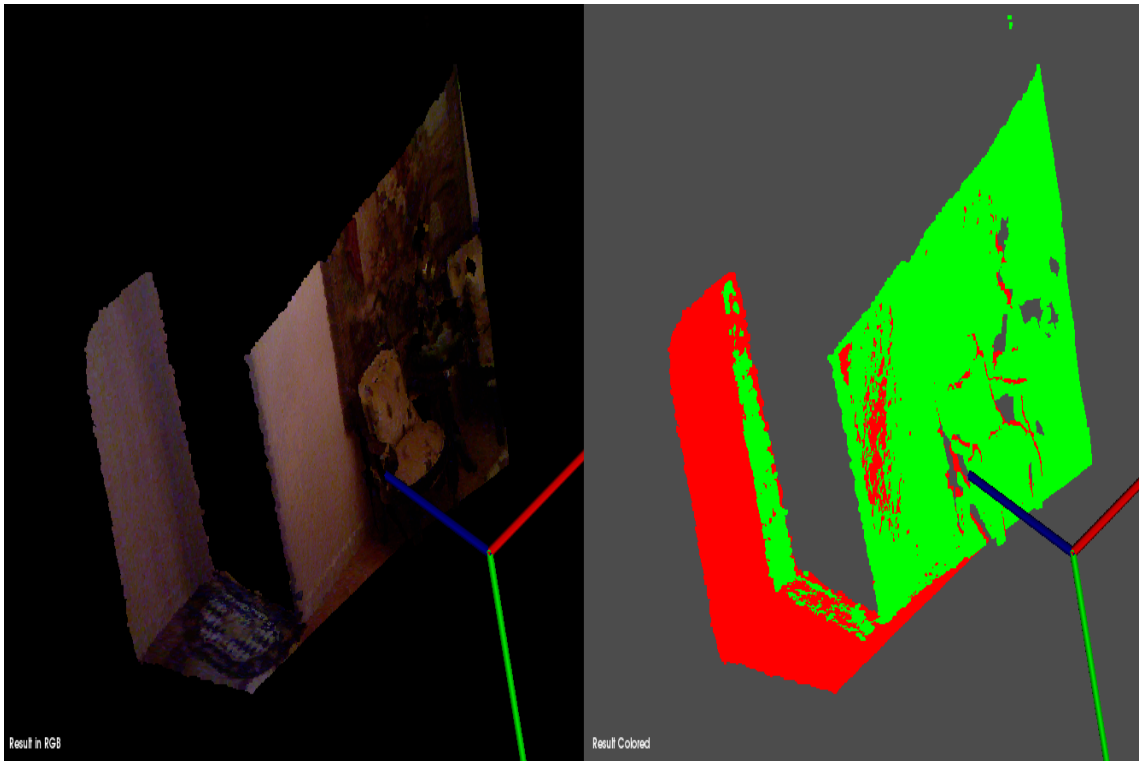

Figure 5.9: Applying ICP after using Horn's method result as an initial guess, the results are clearly better than the one obtained only with Horn's method in Figure 5.5

The previous figure clearly shows how the results have imporved as we see the two point clouds are well aligned with no gaps, and we notice that the problem that was happening for the wall has disappeared.

5.4 Results and performance

Our system needs is real-time systsm as it needs to provide the constructed 3D model while the user is scanning the environment, this constranis that the alignment takes no more than 2 seconds so that the user will feel no lag in the scanning process.

The performance of the methods we have proposed before satisfy this time constraing as the whole alignment process starting from feature detection to that concatenation of aligned two point clouds takes a maximun of one second for two frames of QVGA resolution. The average time for the alignment is 0.1428 seconds. [need to do samples and stuff]

We provide here the results obtained from aligning 21 QVGA frames:

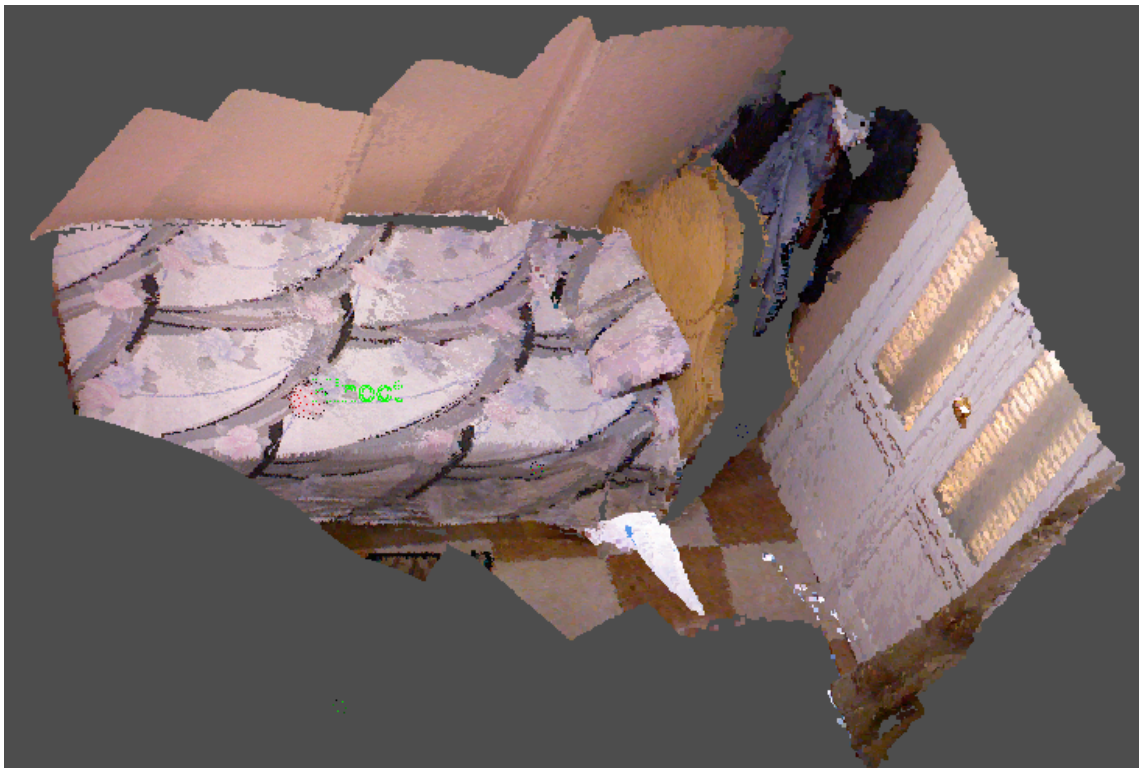

Figure 5.10: Constructed scene from two different viewpoints

More accurate results can be obtained by using VGA resolution instead of QVGA, this enhances every step in the alignment process, as the feature detection and tracking is richer, also the ICP operates on more points that describe the scene in more details.

The difference can be clearly noticed in the two following images that show the alignment result of 14 VGA frames in 7 seconds.

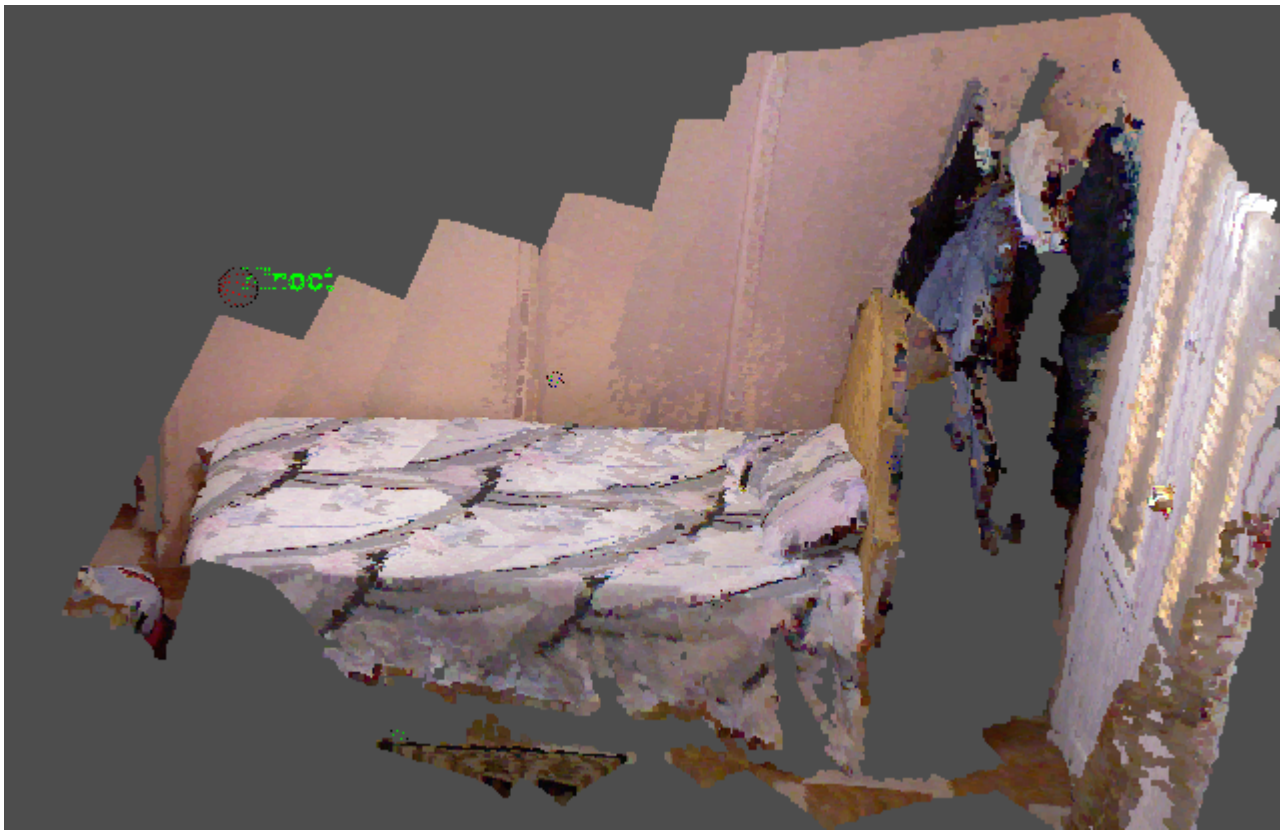

Figure 5.11: Constructed scene from two different viewpoints

Chapter 6

Loop Closure and Global Optimization

6.1 Introduction

As explained before, the alignment we are implementing is a pairwise alignment which means that evey frame is aligned with the previous frame and so on, although this method gives good results, it will face a problem in case a loop happens in the scanning.

We explain this problem with a simple example: Assume the user scanned a scene so five frames were aligned namely: A, B, C, D, E, If the user has returned to the starting point A then there will be overlapping between the first and last frames A and E respectively, the problem is that even if our alignment calculations are accurate, there are always small errors and false matches that accumlate all over the scanning process which will cause overlapping areas of A and E not to align perfectly as we would want, because we align E according to D only without considering the overlapping between the two frames A and E.

We present here a visual explanation of the problem and examples of scanning processes that include loops.

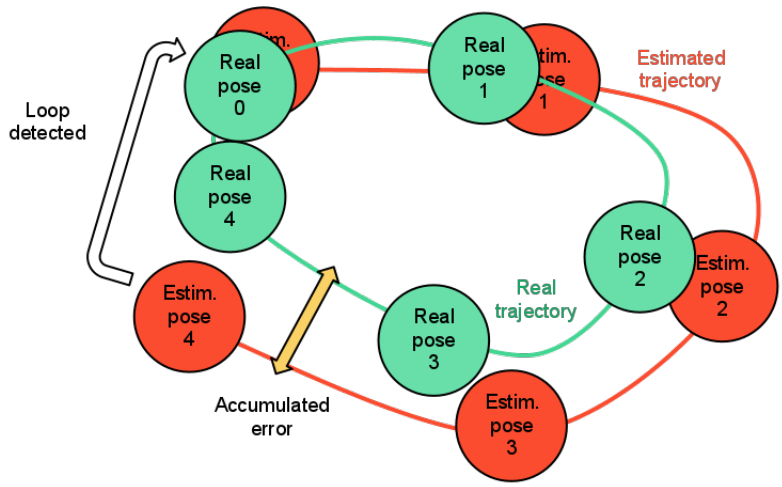

Figure 6.1: Because of accumlating alignment errors, the last estimated pose is far from where the real pose is

Figure 6.2: Highlighted areas are duplicate objects that are not aligned as they should be because of accumlative numerical errors and false matches

Now that we have defined the problem, we introduce a solution that involves two steps; the first step is to detect that a loop has occured (user has returned to a previously visited scene), for the second step we try to realign the frames to remove the error in alignment between the first and last frames in the loop.

6.2 Loop Detection

The brute force approach for loop detection is to just perform feature matching between the current frame and all previous frames and use a threshold to determine if a loop happened, this approach has two drawbacks:

- The overhead incurred for matching a frame with all previous frames is extremely high and is not acceptable for long scans especially that the system we are building is real-time system.
- This approach is not going to be conclusive in the case of similar scenes such as floors or repeated patterns because all the scanned frames will have high values for matching and we can't determine where the loop exactly was formed.

We enhanced the previous approach by performing feature matching only on frames that are spatially close to the current frame, this enhancement is achieved using a k-d tree holding all the previous pose estimations, then every new aligned frame is tested against only those frames that are within a certain distance from the current pose estimation.

6.2.1 k-d Trees

The data structure used to store the pose estimations must be optimized for spatial search, so we can find points that are in a certain range from another point, this is why we chose the k-d tree to store the pose estimations.

k-d is a space-partitioning data structure for organizing points in a k-dimensional space. k-d trees are a useful data structure for several applications, such as searches involving a multidimensional search key.

We explain here how a new point is inserted in a k-d tree, for simplicity we consider the case when k = 2 (each point has x and y coordinates), in this case we do insertion just like a binary tree but as we have two different values (x and y coordinates) at each node then we alternate between them in comparison; so we compare at the root according to x, then at depth one we compare according to y, then at depth two we compare according to x and so on ..

The following algorithm inserts a new node in a k-d tree where k = 2:
procedure **Insert(T, N, depth=0)** takes T the root of the tree, N the node to be inserted and depth holding the value of current depth.

```
if T is NULL
    then T = N
else
    if depth is even
        if N.x > T.x
            then Insert(T.right, N, depth + 1)
        else Insert(T.left, N, depth + 1)
    else
        if N.y > T.Y
            Insert(T.right, N, depth + 1)
        else Insert(T.left, N, depth + 1)
```

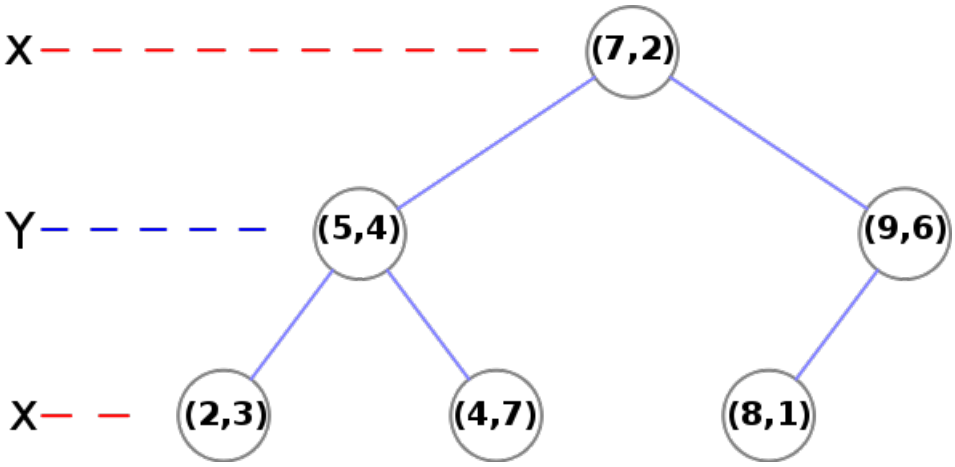

Figure 6.3: k-d tree with d = 2, this tree was built by inserting (7,2), (9,6), (5,4), (2,3), (4,7) and (8,1)

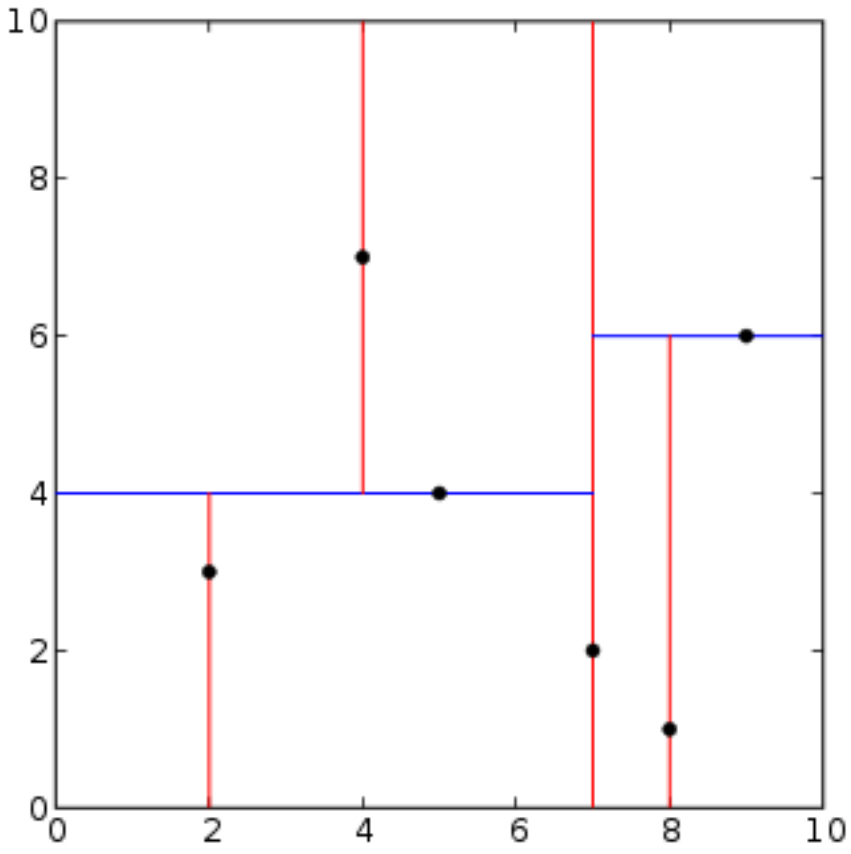

Figure 6.4: The decompsition of the k-d tree in fig 6.3

The previous procedure for building the k-d tree can be generalized fot k = 3 by alternating between x, y and z and each level of the tree.

After we have built the k-d tree containing all pose estimations, we need to find all the previous estimations that fall in a certain range from the current pose estimation.

Range search operations in k-d tree depend on the fact that we partition the space into different parts, this is clear in the following figure:

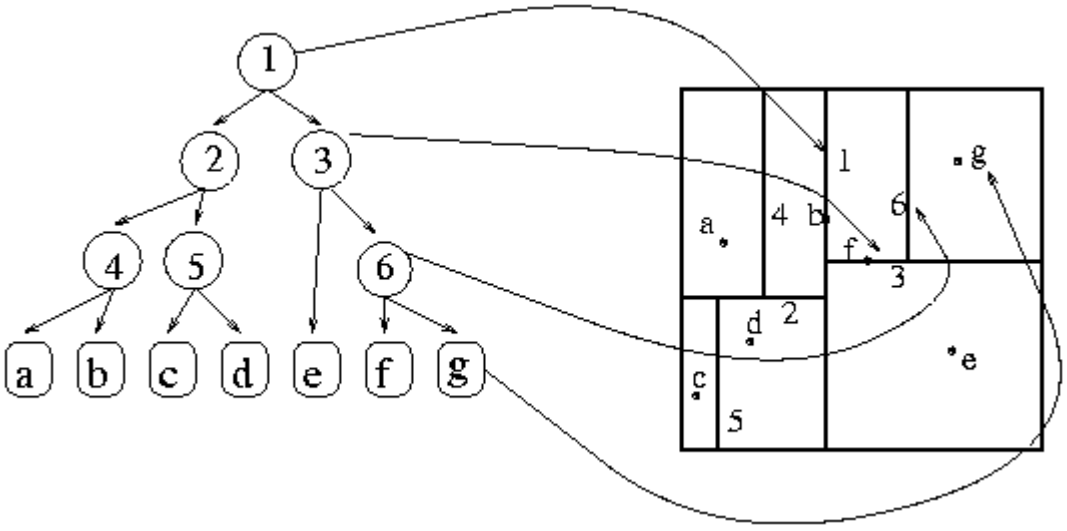

Figure 6.5: The root of the tree which is node 1 splits the space vertically into two parts; one containing all points on the left of point 1 and one containing all points on the right of point 1, the same is for node 3 which splites the space horizontally and node 6 that splits the space vertically

The range search in k-d trees starts from the root and tests if any of its subtrees is fully contained inside the region then all nodes in the subtree are returned, if the subtree only intersects with the region then we call the search procedure on the subtree, but if the subtree is not contained and doesn't intersect with the region then we just ignore it.

The following algorithm explains how search is performed:
procedure **SearchKdTree(v, R)** takes v the root of the tree, and R the range to search inside.

SearchKdTree(v, R)

```
if v lies in R
    then Report the point stored at v
else
    if region( v.left ) is fully contained in R
        then ReportSubtree( v.left )
    else
        if region( v.left ) intersects R
            then SearchKdTree( v.left, R )
    if region( v.right ) is fully contained in R
        then ReportSubtree( v.right )
    else
        if region( v.right ) intersects R
            then SearchKdTree( v.right, R )
```

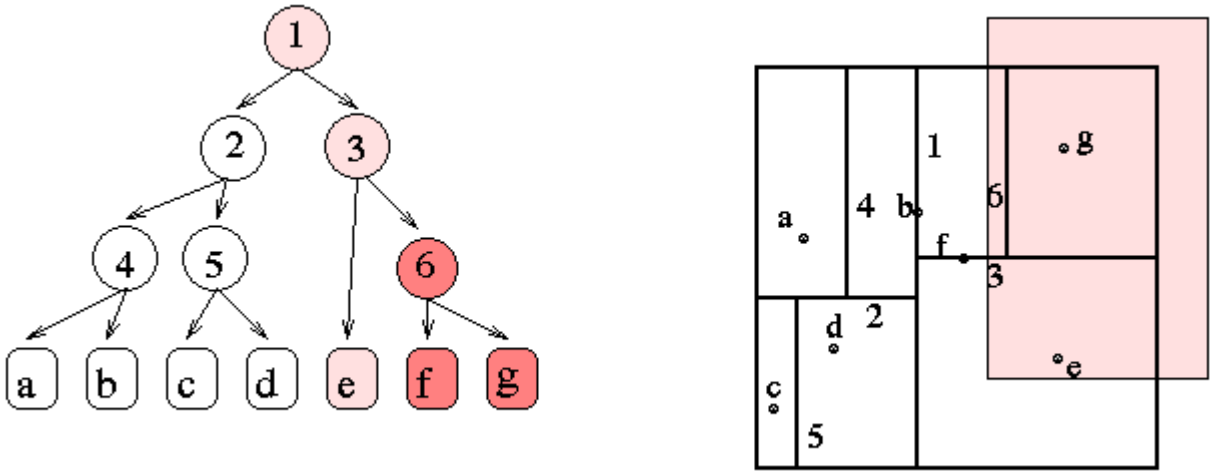

Figure 6.6: Here we search for any nodes that fall in the pink rectangluar, It is clear that the region rectangle intersects with the right side of point 1 and not with the left side of point 1, so we investigate only the right subtree starting at node 3, there we find that e the point below point 3 falls into the region and the subtree starting also intersects with the region so we investigate it and find that only the point g falls in the region

6.2.2 Feature matching

After finding all previous pose estimations that fall in a certain range from the current previous estimation, we preform a visual test to make sure that the current pose and the previous one are both close to each other and are very similar visually.

We choose the closest pose estimation that is still far enough not to be just the previous one, and we use the feature matching explained before in chapter 4.

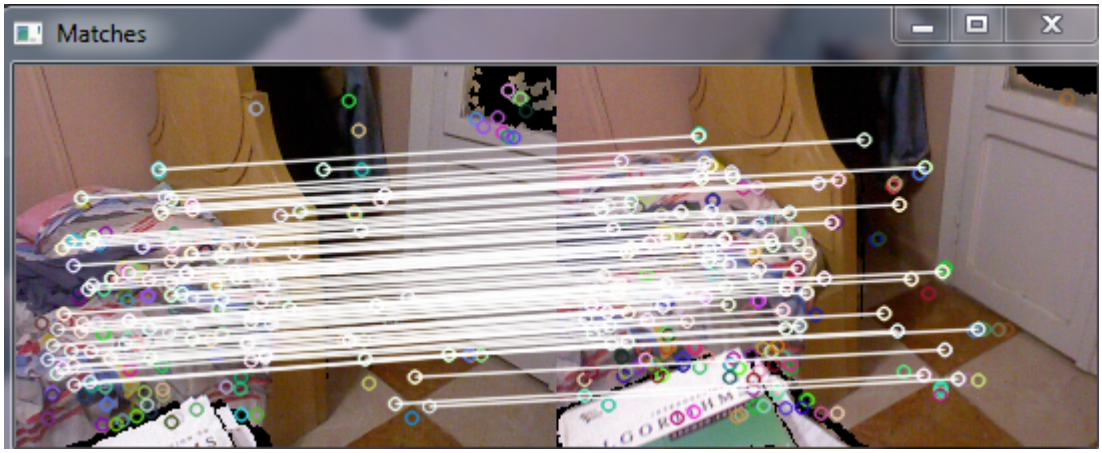

Figure 6.7: Feature matching between the first and last frame in the scanning

6.2.3 Loop detection Results

We present here three shots; the first shot represents the first frame of the scanning process and the first pose estimation indicated with the red sphere is at (0,0,0), the second shot shows the scene after building and the last frame is spatially close and visually similar to the first one so a loop is detected, the third shot shows us the loop in 3D space starting from (0,0,0) and going in anti clock-wise direction.

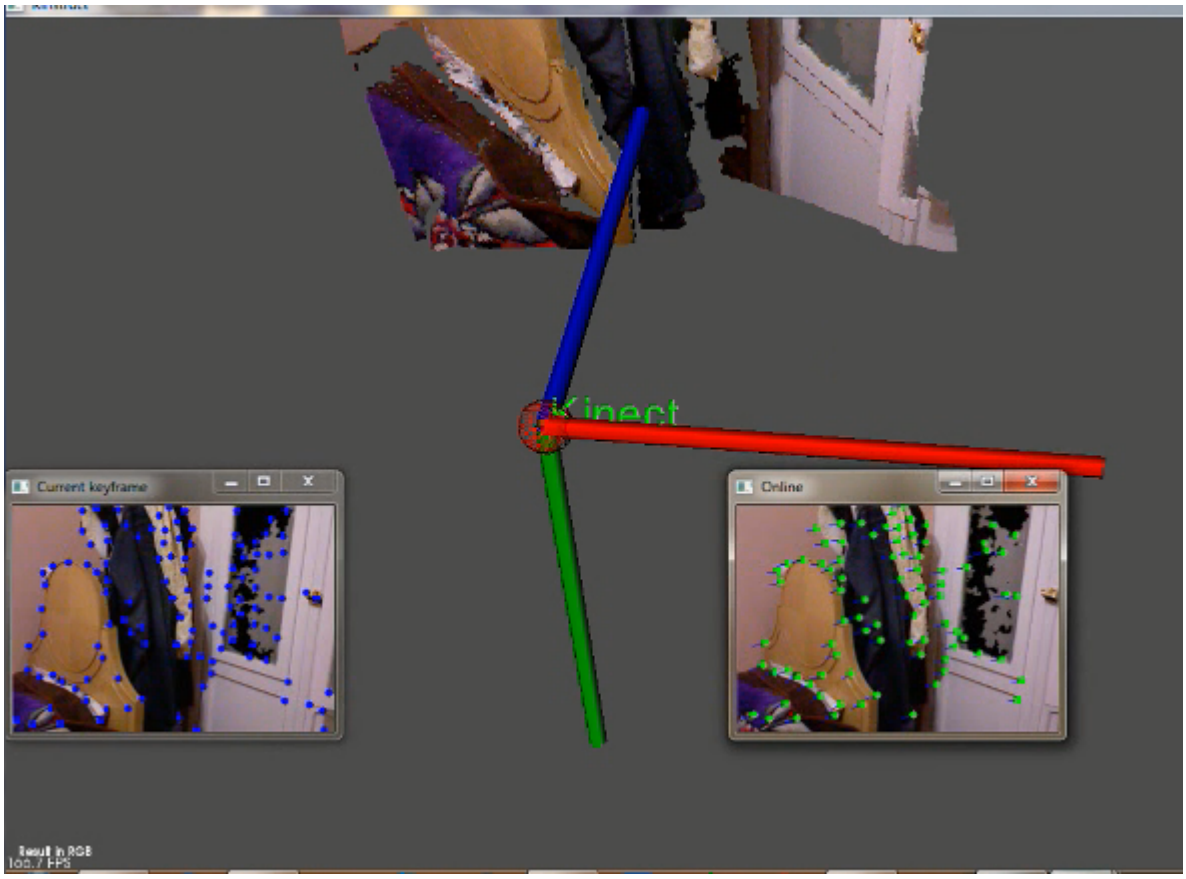

Figure 6.8: This is the first frame in a scanning process

Figure 6.9: This is the last frame in scanning process, we notice that the last frame is similar to the first frame

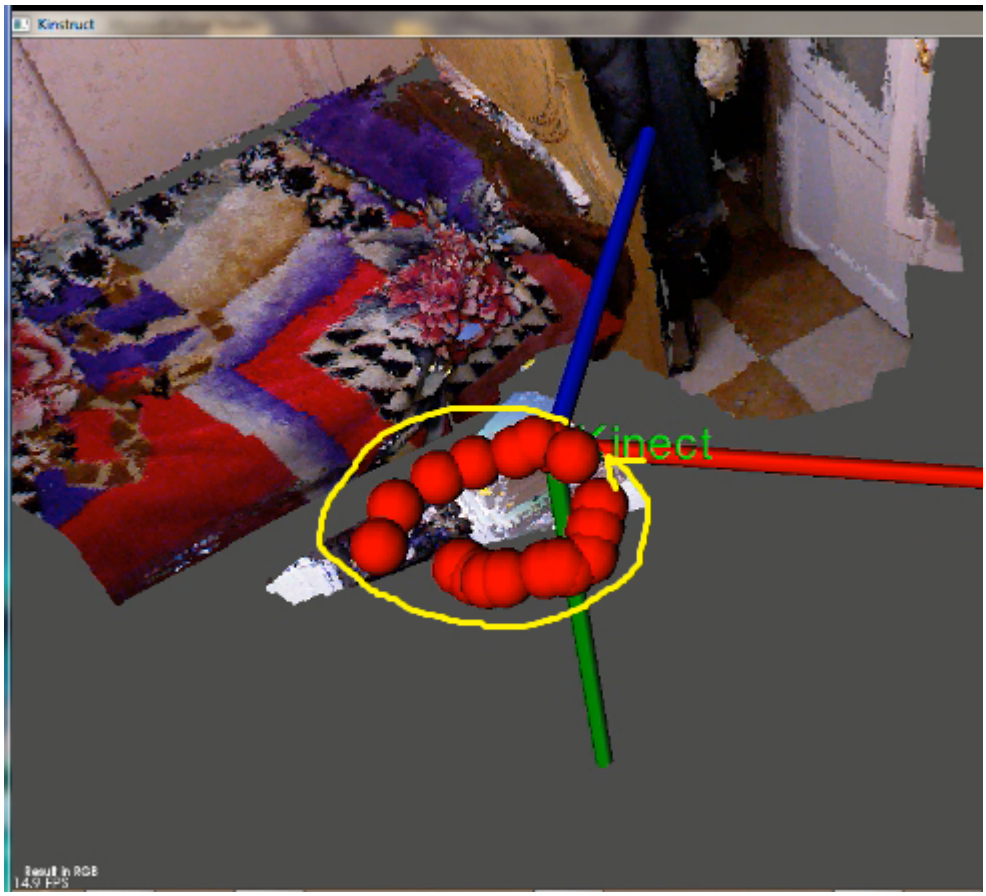

Figure 6.10: The red spheres represent the pose estimations that form the loop

6.3 Global Optimization

Now that we have been able to detect that a loop has occurred during the scan process, we need to find a way that will make the error in the loop has less effect on the constructed scene.

In SLAM graphs pose estimations are represented by nodes while edges represent constratints (transformations) between the estimations, the purpose of global optimization systesm is to minimize the global error in the graph by changing the constraints between the poses.

We have used the newly published method named Explicit Loop Closing heuristic, this method is different from other systems such as TORO that it is not an iterative method, the main idea of the algorithm is to dissociates the last scan of a sequence of acquired scans, reassociates it to the map, built so far by scan registration, and distributes the difference in the pose error over the SLAM graph.

We used the implementation of which is provided as a part of the Point Cloud Library (PCL), the next section shows the results of loop detection followed by global optimization.

6.4 Implementaion and Results

We have successfully used the techniques explained in previous sections to perform loop closure and global optimization, we have used a threshold of at most 15 cm between two frames to be considered spatially the same, then we require at least 10 SURF feature matches between the two frames to be visually the same.

We present here two shots of the same constructed scene before and after the applying ELCH to remove noise and duplication resulting from loop closure and accumlating error, the fixing operation took about 16 seconds for 18 aligned frames with QVGA quality.

Figure 6.11: Highlighted areas are duplicate objects that are not aligned as they should be because of accumlative numerical errors and false matches

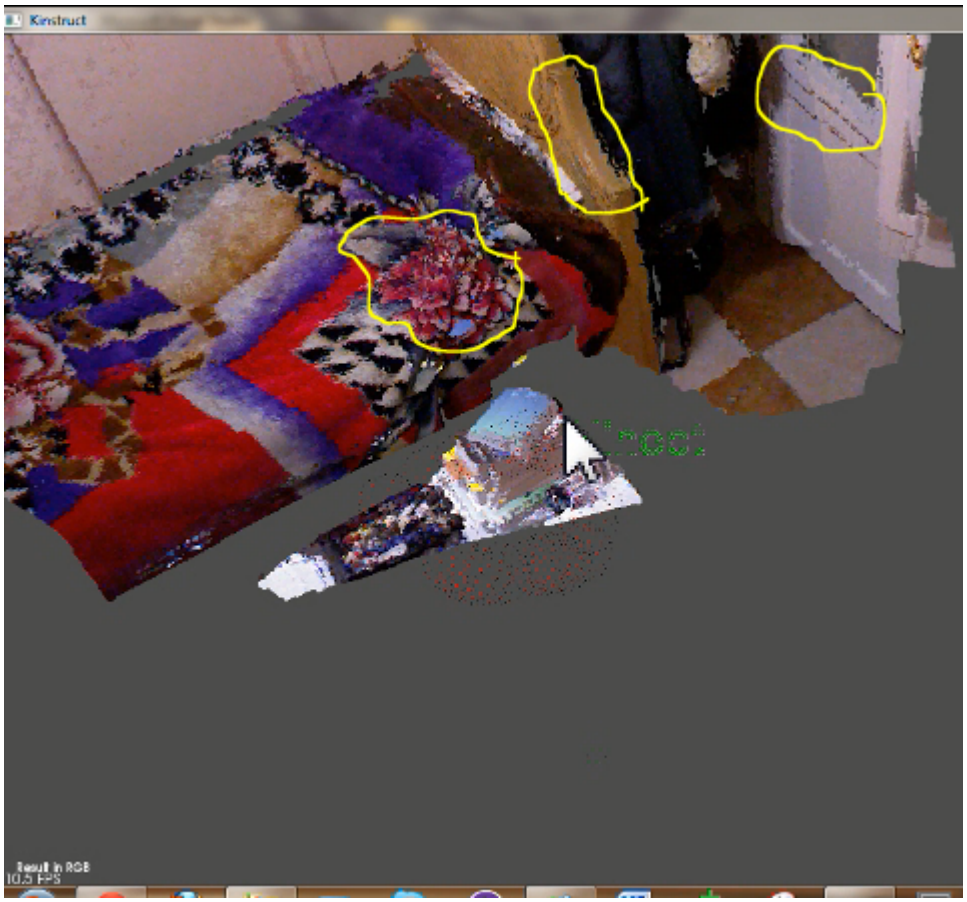

Figure 6.12: After applying ELCH, we notice how noise and duplication from the marked area compared with the constructed scene before applying ELCH in fig 6.11

Chapter 7

Conclusions and Future Work

7.1 Conclusions

7.2 Future Work

Although 3D scanning is relatively an old problem, it is still an important research area, and will continue to be like this because of the continuous improvement in cameras and hardware, which makes it possible to produce results in a faster and more accurate manner.

7.2.1 Kinect Development

Kinect is a very hot supported by Microsoft, this means that Kinect will continue to develop in the future, so it is expected to have better hardware providing better images, it is likely also it will be supported officially on more platforms, also its price is likely to go cheaper making it more and more available. It's also clear now that Kinect can be used heavily outside the gaming industry, this is why Microsoft launched Kinect for Windows which is a Kinect camera adjusted to work better with Windows.

Figure 7.1: Kinect for Windows

Figure 7.2: Kinect for Xbox

7.2.2 Mobile

No one can deny how important mobile platforms are in today's world, the mobile market is growing with more tablets and smartphones pushed to the market all over the world, and these mobile devices have considerable computational power and capable hardware.

Smart phones and tablets now have extremely powerfull cameras, we have seen the HTC EVO 3D which is capable of recording and viewing 3D videos, and the new Nokia 808 Pure View which has a 41 mega pixel camera that can take photos of superb quality.

Figure 7.3: HTC EVO 3D uses two cameras to record 3D videos

Figure 7.4: Nokia 808 Pure View that can take shots up to 41 mega pixels

All these new mobile devices show us that at some point in the future we can build SLAM technology inside mobiles, this will open

new doors to 3D construction, it will be possible for you to have a 3D model of your vacation beach or hotel room instead of recording a video or taking a photo. The SLAM technology being in the hands of the ordinary user will make it more desirable and demanded which will make researchers and companies work more and more on the technology to prefect it.

7.2.3 Walk through

Our system can be used for advertising for apartments, and it will be a nice feature if the user can actually have a virtual walk through inside the constructed model.

7.2.4 Moving Object

A possible improvement of our system is to add