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.

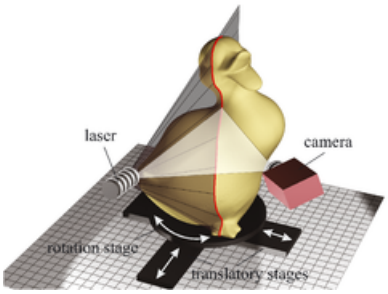

Figure 1.1: Using Laser Scanners

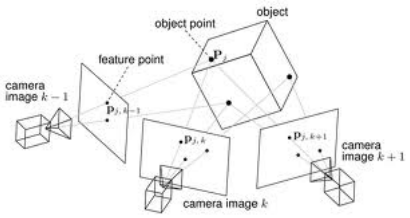

Figure 1.2: Structure From Motion

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

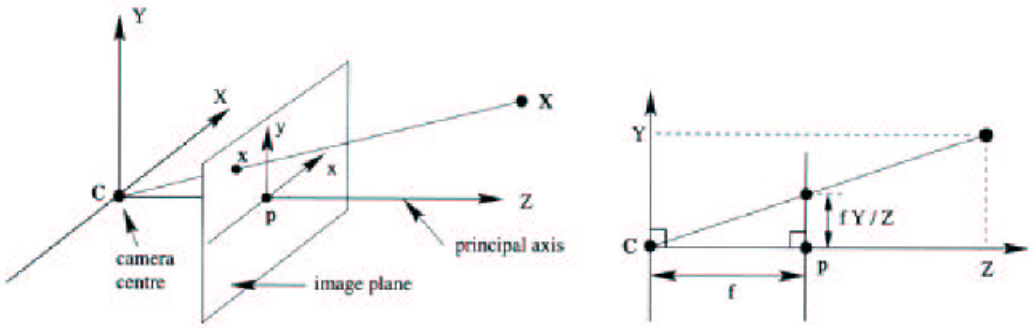

Figure 2.1: The pinhole camera model

   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 $P = K[R|t]$

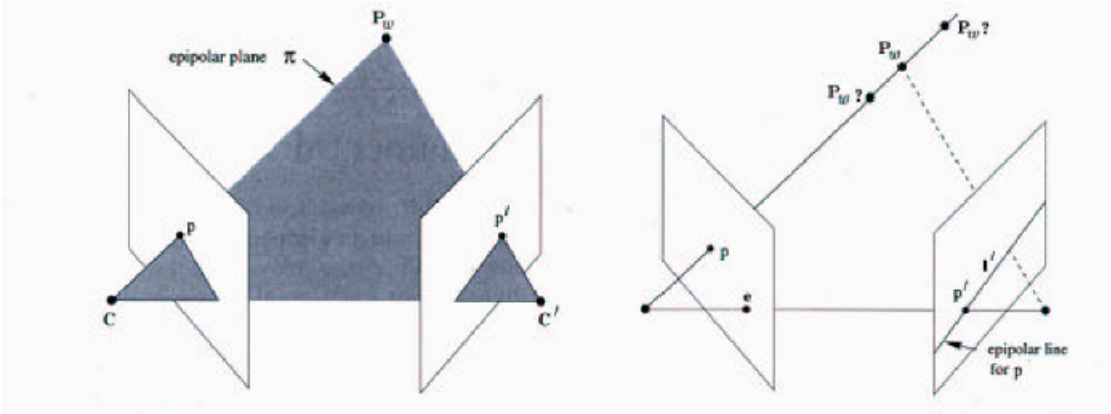

Figure 2.2: Epipolar Geometry

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

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

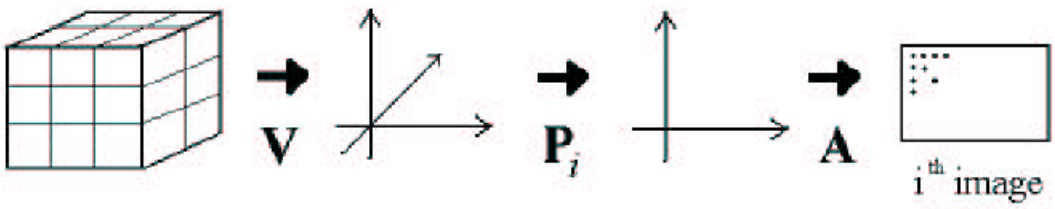

Figure 2.3: A flowchart of mappings from the Voxel Volume to Image Pixels

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

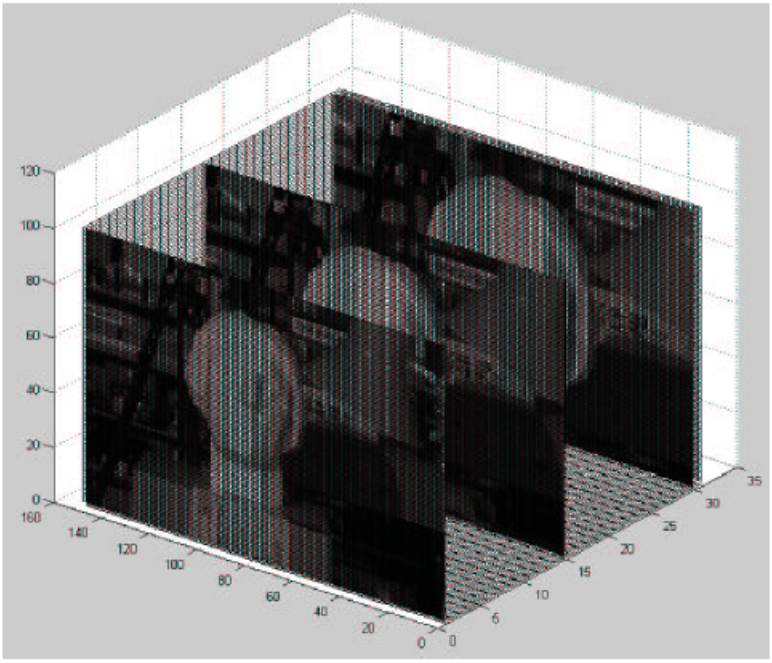

Figure 2.4: Projective volume projection for one of the image in the 'Head and Lamp' sequence

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 $x, yandz$ 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:

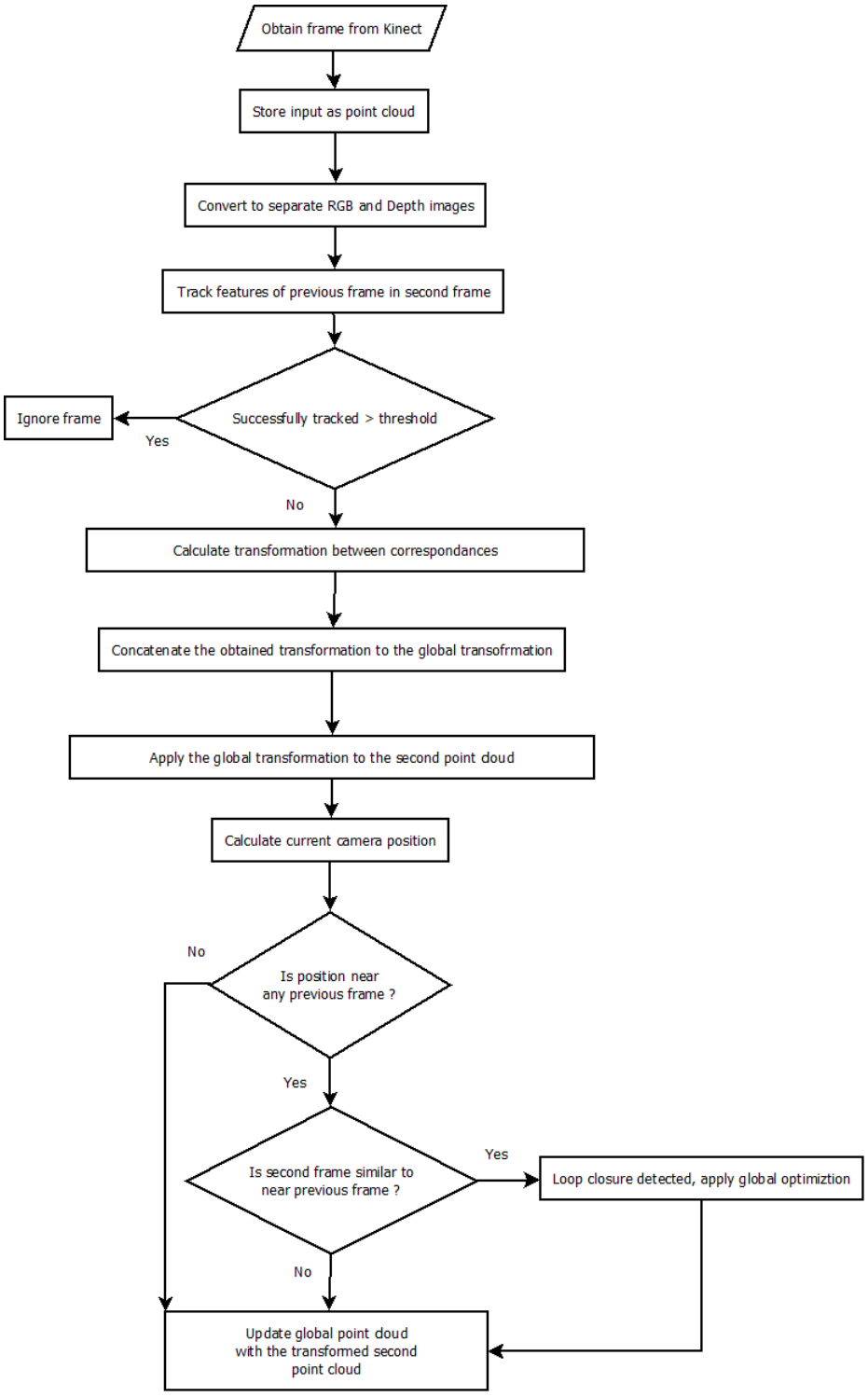

Figure 3.1: System Pipeline

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 uses the previous components to obtain an initial transformation
- SurfaceConstruction:
- Segmentation:

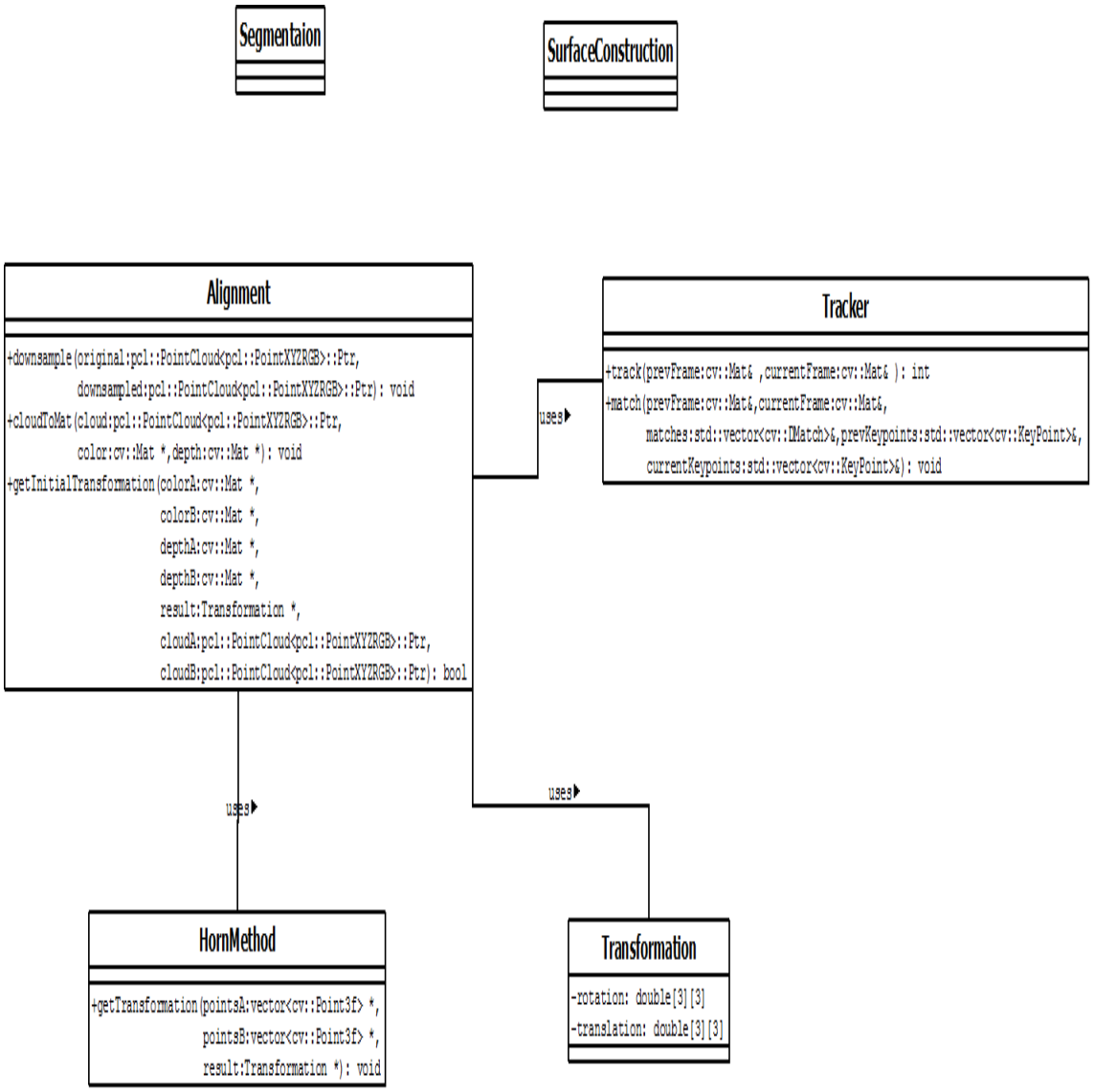

Figure 3.2: Class Diagram

3.3.1 Libraries and tools

Chapter 4

Camera Transformation Calculation

4.1 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.

4.2 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 $F_{AB} = [R|t]$

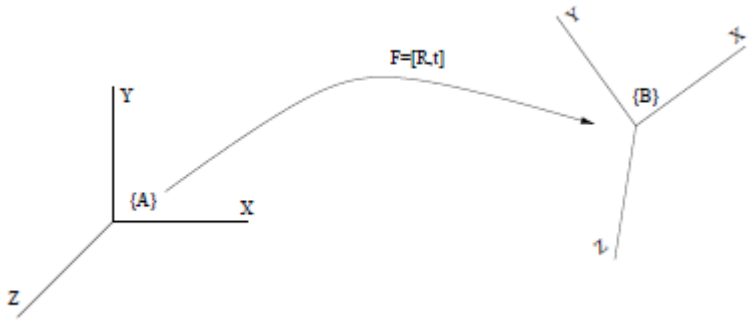

Figure 4.1: One coordinate system transformed to the other

4.2.1 Quaternions

Quaternions are a mathematical object of the form

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

where s, x, y, z $\in$ 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 4.1: 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.

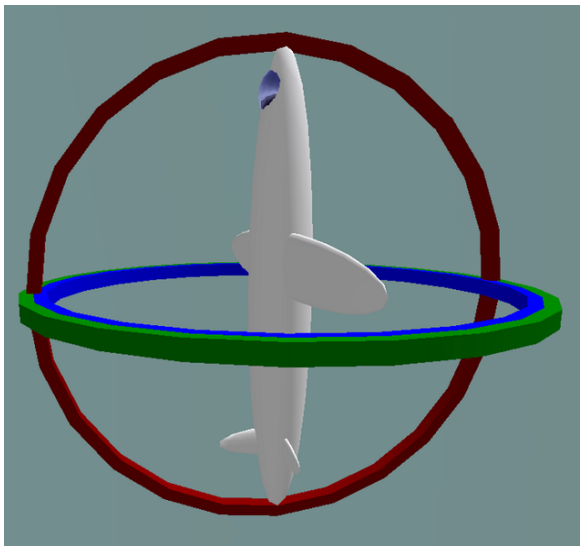

Figure 4.2: Gimbal lock happens when two rotation axes are in the same plane

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 $\bar{q} = q^{-1}$

4.2.2 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

4.2.3 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:

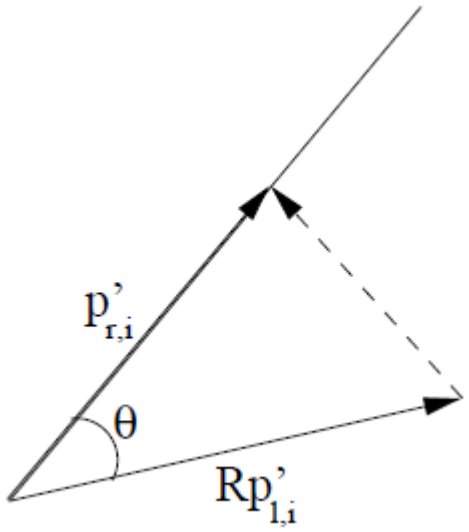

Figure 4.3: Maixmizing the sum $\sum i = 1^n p'_{r,i} \cdot Rp'_{l,i}$ is equivalent to maximizing $\sum i = 1^n |p'_{r,i}||Rp'_{l,i}| \cos\theta$. This sum is maximized when $\cos\theta = 1$, $\theta = 0$. Gemoetrically we compute the rotation which minimizes the angle between the two vectors

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

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

4.2.4 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:

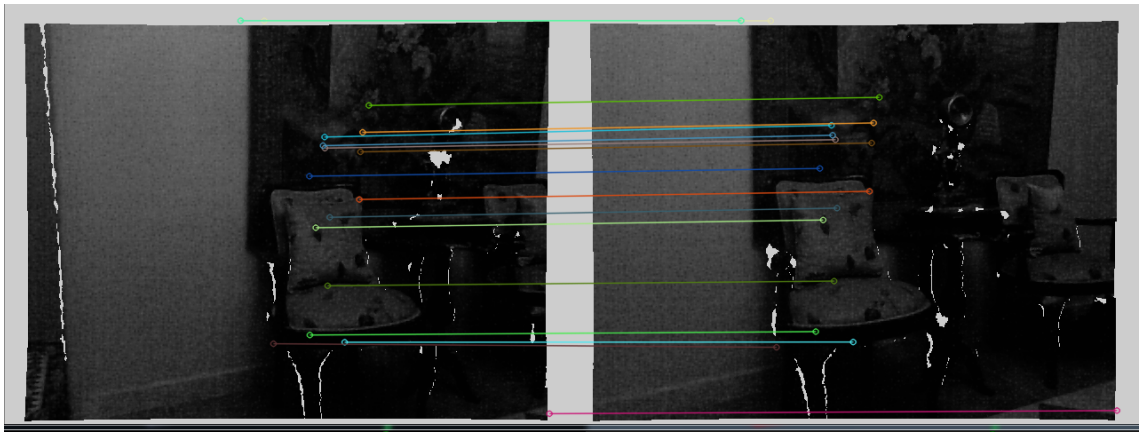

Figure 4.4: We notice that most features are concentrated in the parts that contains chairs while the wall on the right has no features matched

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

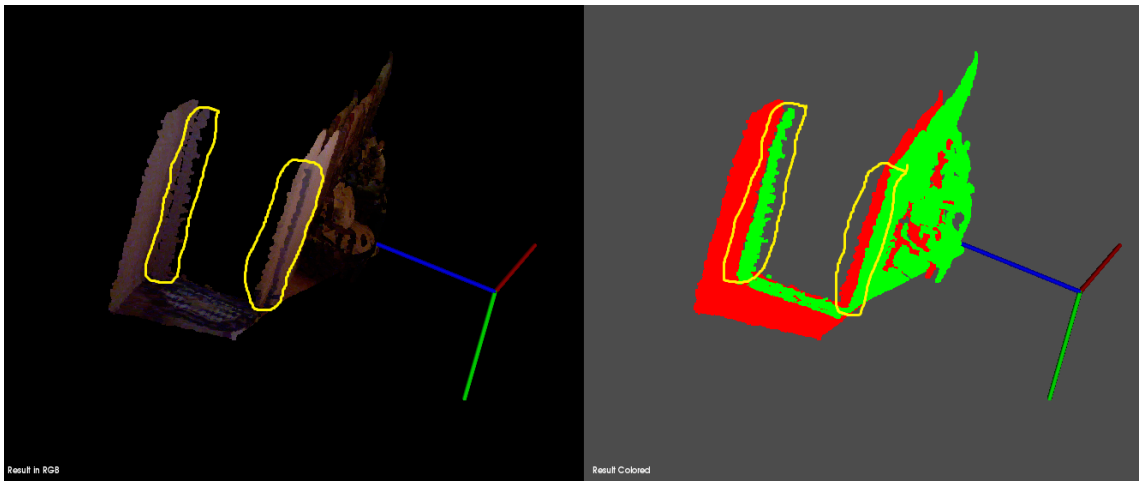

Figure 4.5: Areas marked with yellow show how alignment fails in featureless areas