MERGING POINT CLOUDS FROM MULTIPLE KINECTS

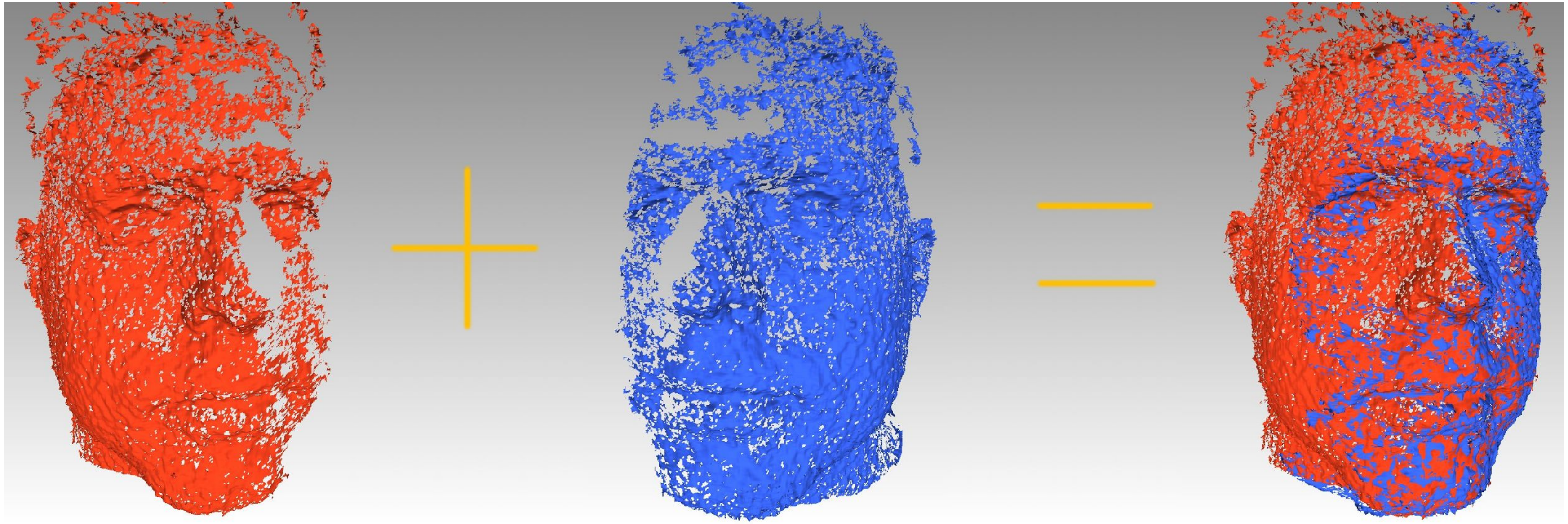
Nishant Rai
13th July, 2016
CARIS Lab
University of British Columbia

Introduction

What do we want to do? : Use information (point clouds) from multiple (2+) Kinects to gain better insight about the scene. Involves aligning the point clouds extracted from the Kinects.

Problem Statement : Given multiple Kinects and the corresponding point clouds, align the Point clouds to get a more 'complete' view of the scene.

Introduction (Cont.)



Use **Multi-View** Information to get the **complete** picture

Image taken from [3]

Introduction (Cont.)

Challenges Involved :

- Getting aligned Point Clouds from unaligned ones
- Difference in scale of the Point Clouds.
- Unaware of how the coordinates transform between each camera.

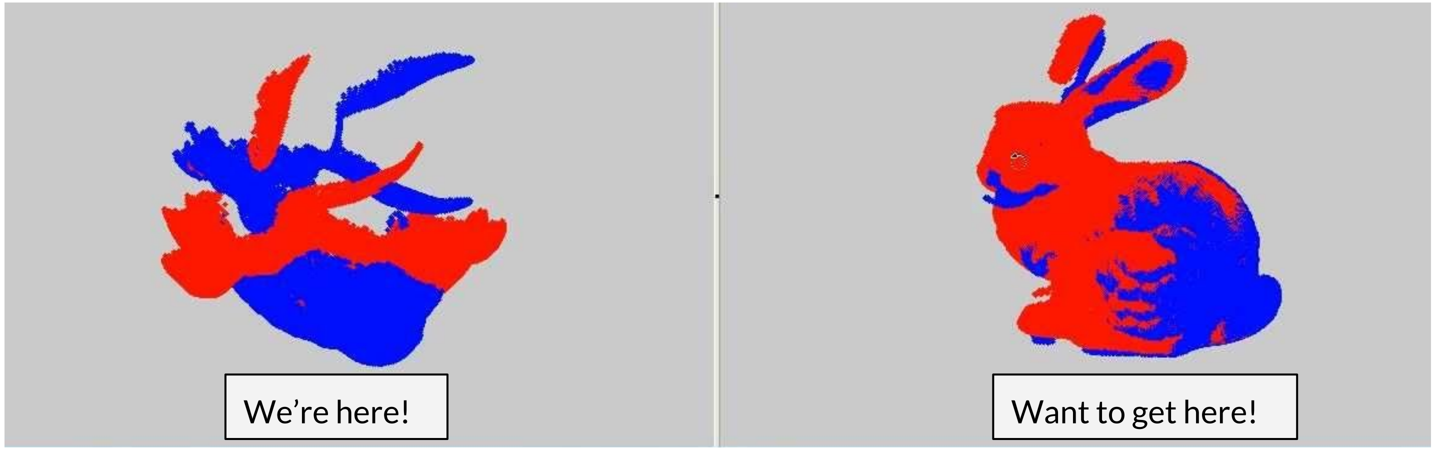


Image taken from [7]

Pinhole Camera Model

A pinhole camera is a simple camera without a lens and with a single small aperture.

Light rays pass through the aperture and project an inverted image on the opposite side of the camera.

Tip: Think of the virtual image plane as being in front of the camera and containing the upright image of the scene.

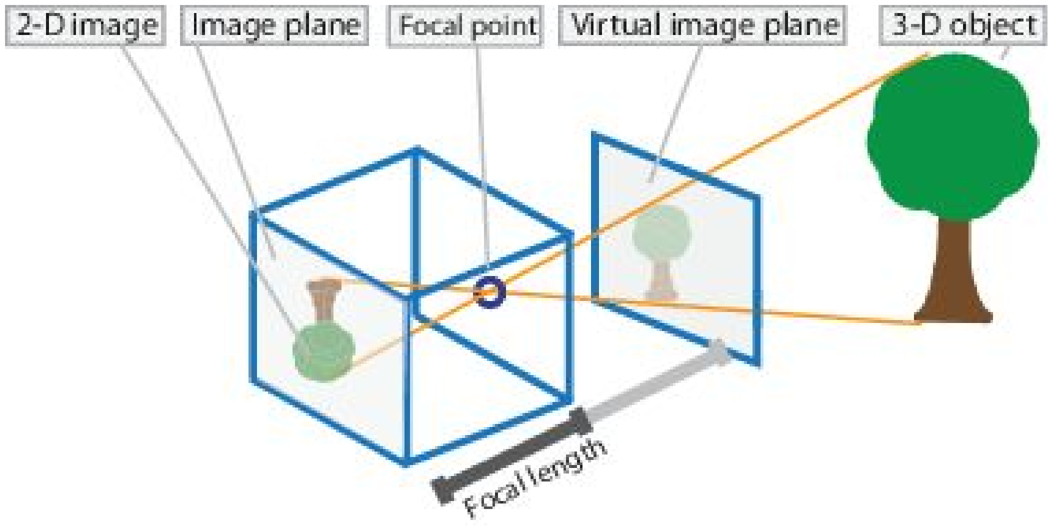
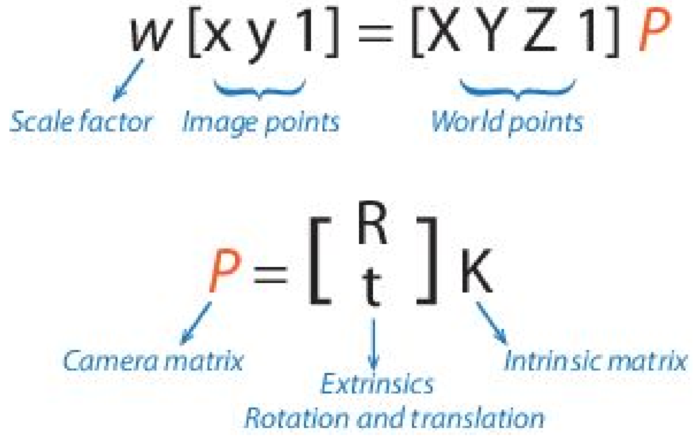


Image taken from [4].
Content inspired from [4]

Pinhole Camera Model (Cont.)

- The pinhole camera parameters are represented in a 4-by-3 matrix called the camera matrix. This matrix maps the 3D world scene into the image plane.
- The calibration algorithm calculates the camera matrix using the extrinsic and intrinsic parameters.
- The extrinsic parameters represent the location of the camera in the 3D scene.
- The intrinsic parameters represent the optical center and focal length of the camera

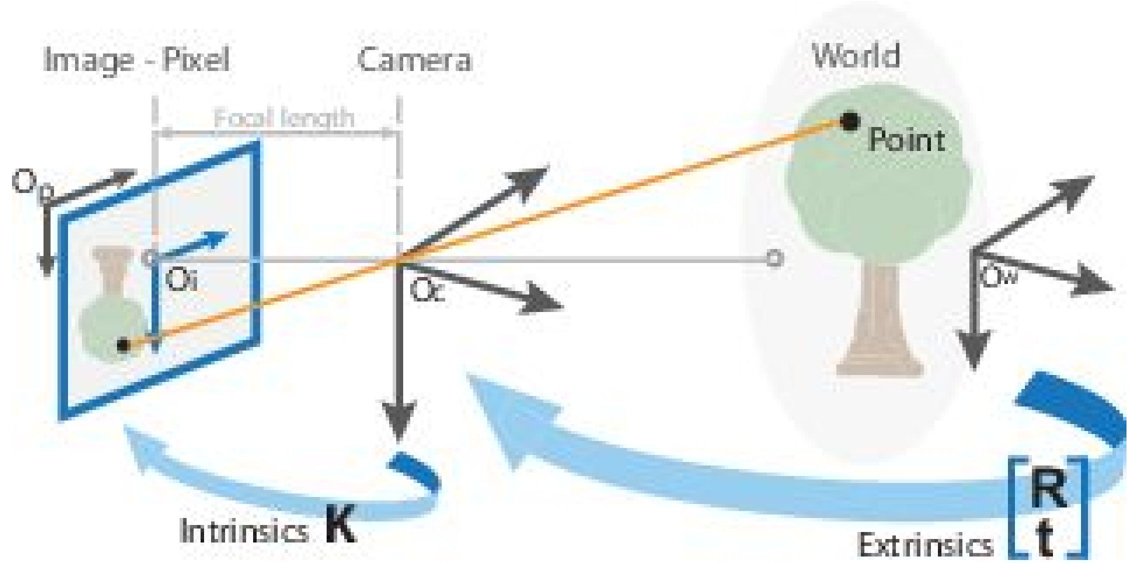


$$\underbrace{w}_{\text{Scale factor}}\,\underbrace{[x\ y\ 1]}_{\text{Image points}} = \underbrace{[X\ Y\ Z\ 1]}_{\text{World points}}\,P$$

$$\underbrace{P}_{\text{Camera matrix}} = \underbrace{\begin{bmatrix} R \\ t \end{bmatrix}}_{\substack{\text{Extrinsics} \\ \text{Rotation and translation}}} \underbrace{K}_{\text{Intrinsic matrix}}$$

Image taken from [4].
Content inspired from [4]

Pinhole Camera Model (Cont.)



The world points are transformed to camera coordinates using the extrinsics parameters.

The camera coordinates are mapped into the image plane using the intrinsics parameters.

Image taken from [4]

Basic Approach

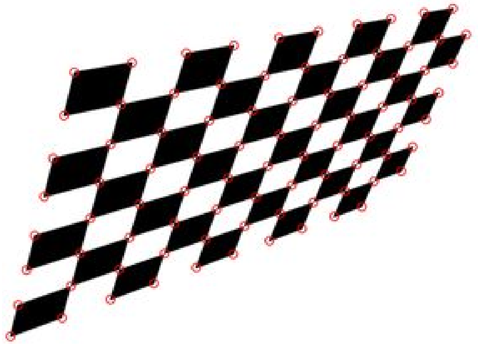
Steps Involved:

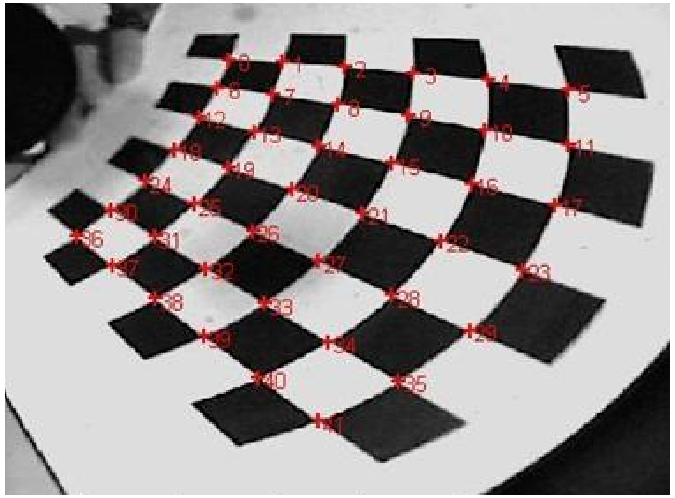
- Use a calibration target. Generally a chessboard (Why?).
- Detect the target in each view.
- Identify the relevant transform between the camera and the calibration model.
  - Compute Homography between model and image
  - Use intrinsics matrix to extract the extrinsics.
  - Construct the transformation.
- Use the computed transformation to link points in different views.

Chessboard Corner Detection

A variety of methods to perform this task, so not going into the details of the algorithms used.

Many libraries and functions for detecting the corners. Motivates us to choose a chessboard as the calibration target.





We use a 6x7 chessboard target.

Image taken from [2]

Homography Computation

Basic Notations:

- A 2D point is represented by $\mathbf{m} = [u, v]^T$. A 3D point is represented by $\mathtt{M} = [X, Y, Z]^T$.
- We create augmented vectors by adding 1 as the last element, denoted by,
  $\widetilde{\mathbf{m}} = [u, v, 1]^T$ and $\widetilde{\mathtt{M}} = [X, Y, Z, 1]^T$
- The relationship between a 3D point **M** and its image projection **m** is given by,
  $s\widetilde{\mathbf{m}} = \mathbf{A}[\mathbf{R} \quad \mathbf{t}]\widetilde{\mathtt{M}}$ (**s** is the scale parameter)

Homography Computation (Cont.)

Let's look at the relation between a model point and its image.

Without loss of generality, we assume the model plane is on Z = 0 of the world coordinate system.
We denote the ith column of the rotation matrix R by $\mathbf{r}_i$.

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{A}\begin{bmatrix}\mathbf{r}_1 & \mathbf{r}_2 & \mathbf{r}_3 & \mathbf{t}\end{bmatrix}\begin{bmatrix} X \\ Y \\ 0 \\ 1 \end{bmatrix}$$

$$= \mathbf{A}\begin{bmatrix}\mathbf{r}_1 & \mathbf{r}_2 & \mathbf{t}\end{bmatrix}\begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} .$$

$$s\widetilde{\mathbf{m}} = \mathbf{H}\widetilde{\mathtt{M}} \qquad \text{with} \quad \mathbf{H} = \mathbf{A}\begin{bmatrix}\mathbf{r}_1 & \mathbf{r}_2 & \mathbf{t}\end{bmatrix}$$

Credits to [1]

Homography Computation (Cont.)

Let, $H = \begin{bmatrix} P_0 & P_1 & P_2 \\ P_3 & P_4 & P_5 \\ P_6 & P_7 & P_8 \end{bmatrix} = \begin{bmatrix} -- & \bar{h}_1 & -- \\ -- & \bar{h}_2 & -- \\ -- & \bar{h}_3 & -- \end{bmatrix}$, and $x = \begin{bmatrix} \bar{h}_1^T \\ \bar{h}_2^T \\ \bar{h}_3^T \end{bmatrix}$.

Then, the equation $s\tilde{\mathbf{m}} = \mathbf{H}\tilde{\mathtt{M}}$ can be written as $\begin{bmatrix} \tilde{\mathtt{M}}^T & \mathbf{0}^T & -u\tilde{\mathtt{M}}^T \\ \mathbf{0}^T & \tilde{\mathtt{M}}^T & -v\tilde{\mathtt{M}}^T \end{bmatrix} \mathbf{x} = \mathbf{0}$

Stacking all the equations together (n equations for n chessboard points) we get matrix **L**.

$$L = \begin{bmatrix} \tilde{M}_1^T & 0^T & -u_1\tilde{M}_1^T \\ 0^T & \tilde{M}_1^T & -v_1\tilde{M}_1^T \\ \tilde{M}_2^T & 0^T & -u_2\tilde{M}_2^T \\ 0^T & \tilde{M}_2^T & -v_2\tilde{M}_2^T \\ & \vdots & \\ \tilde{M}_n^T & 0^T & -u_n\tilde{M}_n^T \\ 0^T & \tilde{M}_n^T & -v_n\tilde{M}_n^T \end{bmatrix}$$

This finally gives us the equation $\mathbf{Lx} = \mathbf{0}$.

So, we solve the system $\mathbf{Lx} = \mathbf{0}$, with **x** as the variable.

The solution to this equation can easily be found using **S**ingular **V**alue **D**ecomposition.

We reconstruct **H** using the computed **x**.

Credits to [1] and [5]

Extracting Extrinsics

Constructing the Transformation

Estimating the rotation and translation matrices.

$$R = \begin{bmatrix} | & | & | \\ r_1 & r_2 & r_3 \\ | & | & | \end{bmatrix}, t = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix}$$

Use the homography and the intrinsics matrix **A** to recover the extrinsics.
The intrinsics (**A**) of the Kinects have already been computed and are directly used.

$$H = \begin{bmatrix} | & | & | \\ h_1 & h_2 & h_3 \\ | & | & | \end{bmatrix}$$

$$\mathbf{H} = \mathbf{A}\begin{bmatrix}\mathbf{r}_1 & \mathbf{r}_2 & \mathbf{t}\end{bmatrix}$$

$$\lambda_{r_1} = \frac{1}{\|A^{-1}h_1\|}$$

$$\lambda_{r_2} = \frac{1}{\|A^{-1}h_2\|}$$

$$\lambda_{r_3} = \frac{\lambda_{r_1} + \lambda_{r_2}}{2}$$

$$r_1 = \lambda_{r_1} A^{-1} h_1$$

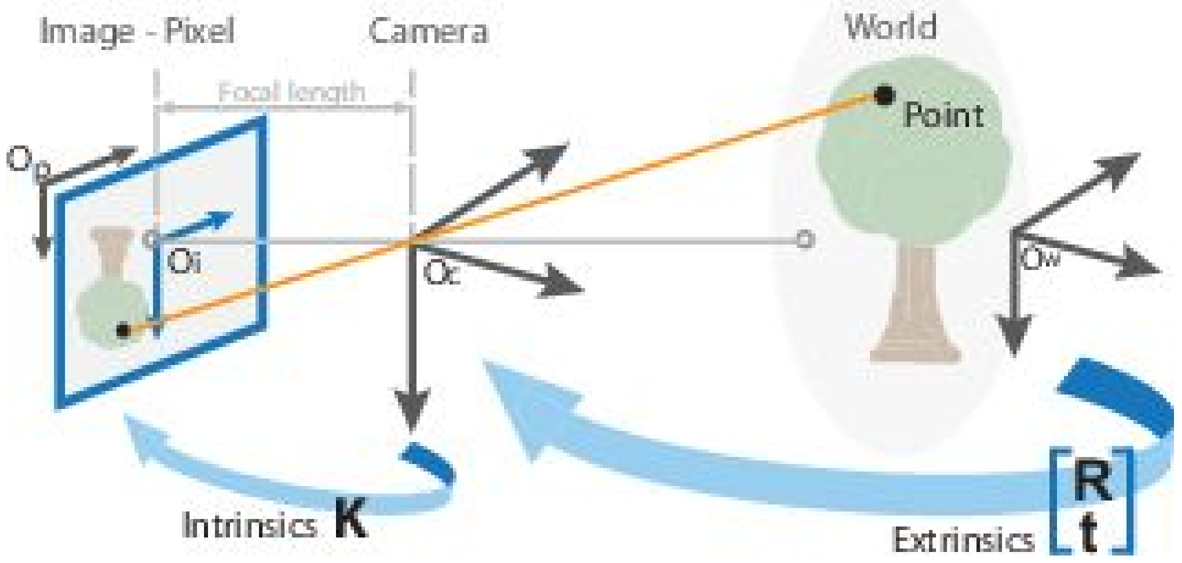
$$r_2 = \lambda_{r_2} A^{-1} h_2$$

Careful! $\Rightarrow$ $$r_3 = r_1 \times r_2$$

$$t = \lambda_{r_3} A^{-1} h_3$$

Extracting Extrinsics

Constructing the Transformation (Cont.)



We have this..

But we want this!

Image taken from [4]

Constructing the Transformation

Once we have matrix **E** = [**R**, **t**]. We get the desired transformation by adding [0, 0, 0, 1] as the last row to **E**. Let's call this transformation matrix **M**.

Note:

- There are actually two possible ways to construct **R**.

  $$r_3 = r_1 \times r_2 \qquad \text{and} \qquad r_3 = r_2 \times r_1$$

- But only one is correct! How do we find the correct solution?
- Hit and Trial
- Choose a 3D point which should be in front of the camera. Multiply it with **R** and check if it still remains in the front. If it does, then that's our solution.

Merging Information from Kinects

Now that we have computed the transformation matrix (M_i) for each camera i. Let's focus on the final task of merging the information from different Kinects.

Assume that we only have two cameras for now.
We need to get the transformation between them (say camera 1 and 1)
Simply, invert M_1 and multiply by M_2 to get M_{12} (i.e. $M_1^{-1}M_2$).

This can be repeated for other pairs too. Giving us a way to use information from all the cameras simultaneously!

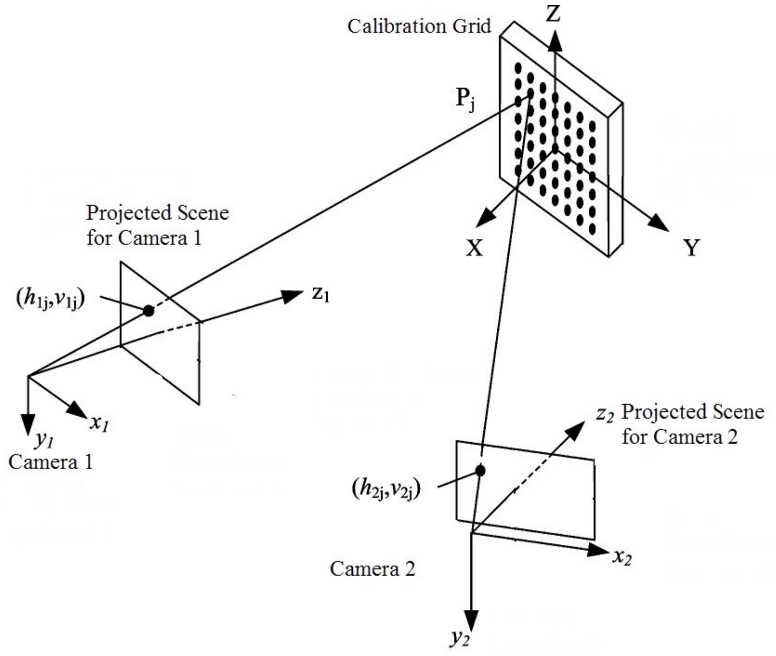


Image taken from [6]

From theory to implementation (Challenges)

Fixing Noisy Rotation Matrices:

Why is R not a correct Rotation matrix? : Mainly due to the real world. Lots of noise in our data, causes the computations to be slightly off.

What do we mean by 'fixing' a Rotation matrix? : They're supposed to follow certain rules, namely, rotation matrices should be orthogonal i.e. $AA^T = I$

How do we fix it? : A possible fix involves Singular Value Decomposition.

Quick Summary

All you have to do is..

- Print a pattern and attach it to a planar surface.
- Detect the feature points in the images (Can use chessboard corners, lots of libraries and functions available for the same)
- Estimate the extrinsic parameters using the method discussed earlier.
- Compute the transformation matrix (M_i) for each camera i.
- To get a transformation between two cameras (say 1 and 2), invert M_1 and multiply by M_2 to get M_{12} (i.e. $M_1^{-1}M_2$).

References

1. Z. Zhang, "A flexible new technique for camera calibration : http://research.microsoft.com/en-us/um/people/zhang/Papers/TR98-71.pdf
2. Chessboard Corner Detection : https://en.wikipedia.org/wiki/Chessboard_detection
3. Identity recognition using 4D Facial Dynamics : http://dynface4d.isr.uc.pt/database.php
4. What is Camera Calibration (MathWorks) : http://www.mathworks.com/help/vision/ug/camera-calibration.html
5. Zhang's Camera Calibration : http://webserver2.tecgraf.puc-rio.br/~mgattass/calibration/zhang_latex/zhang.pdf
6. Chris Walker's Blog, Stereo Vision : http://chriswalkertechblog.blogspot.ca/2014/03/stereo-vision-basics.html
7. Registration Technique for Aligning 3D Point Clouds : https://www.youtube.com/watch?v=Tn0JOVpFWzM
8. Distortion in Cameras : http://www.photocritic.org/articles/everything-about-camera-lenses

Thank You!
Questions?