

# **A Real-time Implementation of Stereo Vision**

Honours and PGDip Projects 2009

**08602700**

**XINGJIE RUAN, XENOPHON**

## Table of content

1 Introduction.....	4
2 Literature review.....	5
2.1 Concept of Stereo vision.....	5
2.1.1 Human Stereo vision.....	5
2.1.2 Stereo vision in computer vision.....	5
2.1.3 Contradistinction between human stereo vision and computer stereo vision.....	5
2.2 Concept of Camera Calibration.....	5
2.3 Concept of Correspondence problem.....	5
3 Methodology.....	6
3.1 Camera Model.....	6
3.2 Capture Images .....	9
3.3 Camera Calibration.....	9
3.3.1 Hartley's algorithm.....	10
3.4 Correspondence.....	12
3.4.1 Stereo correspondence problem.....	12
1.3.4.2 Compute depth (Triangulation).....	12
3.4.3 Block Match Algorithm.....	12
4 Implementation.....	14
4.1 Capture Image.....	14
4.2 Camera Calibration.....	14
4.3 Find Correspondence.....	15
5 Result and Discussion.....	16
5.1 Result.....	16
5.1.1 Result of Calibration.....	16
5.1.2 Result of real time stereo vision.....	17
5.1.3 Good Cameras vs. Bad Cameras.....	19
5.2 Speed.....	21
5.2.1 History.....	21
2.5.2.1.1 Hardware.....	21
5.2.1.2 Software.....	24
7 Acknowledgements.....	26
8 References.....	27
9 Bibliography.....	29
10 Appendices.....	30
10.1 Camera Calibration .....	30



# A Real-time Implementation of Stereo Vision

XINGJIE RUAN, XENOPHON

*Institute of Information  
& Mathematical Sciences  
Massey University at Albany,  
Auckland, New Zealand  
Xingjie.Ruan.1@uni.massey.ac.nz*

This paper proposes the implementation of Stereo Vision. The author presents the concept of stereo vision, the implementation of stereovision. The advantage of using parallel in stereo vision and the parallel algorithms for stereo vision.

**Keywords:** stereovision; calibration; speed up; corresponds; parallel.

## 1 Introduction

Almost all the human beings got the stereo vision. People know objects are moving in the recognition of stereo vision. Nowadays, stereo vision becomes an important part of computer vision. Stereo vision used in many areas of computer vision. For example, stereo vision can be used in robots. The robot can use their stereo vision to detect the distance between itself and the target object. Stereo vision can be used in the security system, to join two or more images together, so that can increase the range of vision. In the other hand, how to build stereo vision in real-time becomes a big problem. Whatever in the case of robot or the case of security system, speed becomes more and more important. And also people can use parallel algorithms or parallel hardware to speed up.

It is possible to build a 3-D scene by given a series of 2-D images. By known that , given two images of a single scene we can get the depth of the scene. The human brain can handles this task easily. People have two eyes. There is a certain distance between two eyes, so there are some differences between two images from our eyes. Human's brain will detect this difference, and represent the 3-D scene.

In computer vision, people can build up a hardware environment to simulate human's eyes: By setting up two cameras in the same level. We can use some algorithms to compute the data from images to represent 3-D scene.

A complete stereo vision system can generally be divided into image acquisition, camera calibration, stereo matching, depth. Accurate Calibration of video cameras inside and outside the parameters of one and three-dimensional matching is a three-dimensional reconstruction of the most important and most difficult problems.

Open source computer vision library OpenCV (Open Source Computer Vision Library) by Intel's research lab in Russia developed, it is a set of freely available by a number of C functions and C + + class library consisting of used to achieve some common image processing and computer vision algorithms [2]. OpenCV and Intel Corporation developed by another image processing library IPL (Image Processing Library)-compatible, IPL is used to realize a number of low-level digital image processing, while OpenCV is mainly used to image a number of advanced processing, such as feature detection and tracking, motion analysis, object segmentation and recognition and 3D reconstruction. As the OpenCV source code is completely open, and the source code to prepare simple but efficient, in particular, most of which have been compiled function optimization, in order to enable efficient and full use of Intel's chip design system for dealing with family for the Pentium MMX, Pentium, Pentium III and Pentium IV in terms of these processors, OpenCV code execution efficiency is very high, so in recent years abroad in the relevant fields of image processing is widely used as a popular image processing software. OpenCV in the camera calibration module provides users with a good interface, also supports Windows, Linux platforms, effectively improving development efficiency, and implement fast and has good cross-platform portability, it can be well applied to engineering practice them.[6]

## 2 Literature review

### 2.1 Concept of Stereo vision

“Stereo vision (Stereopsis) is the process in visual perception leading to the sensation of depth from the two slightly different projections of the world onto the retinas of the two eyes.” [1]

#### 2.1.1 Human Stereo vision

Human has two eyes and one head, each eye can capture one image in a really short time. And the eyes send this two images to out brain. After the computation in the brain, the brain will show us what we saw. There is one way to confirm the images captured by eyes are different: close left eye to capture the image from the right eye , then close the right eye to capture the image from left eye. According to the contradistinction, there are some slightly differences between two images. By the result of the contradistinction, human knows that the brain is not just only mix two images together, but mix them by some methods.

Human are able to deduce depth information from differences of a scene as seen by left and right eye. For example, if object one is moving on object two, object two is seen by human but some parts of object is not seen by human (if object two is bigger than object one). Then human know that: object one is closer.

#### 2.1.2 Stereo vision in computer vision

Stereo vision in computer vision is almost the same as stereo vision in human. Stereo vision in computer vision and stereo vision in human both work on the same principle. We can set up two camera as people's eyes, one computer as people's brain. Each camera can capture one image and send them to the "brain" , which is the computer. Then the computer need some methods to mix the images together. Then the result will present on the screen.

#### 2.1.3 Contradistinction between human stereo vision and computer stereo vision

Human stereo vision and computer stereo vision is almost the same. That's why stereo vision can be presented by computer. But they are not totally the same. Because the computer do not know what people 's brain do. So it is necessary to "tell" the computer, which methods human used.

### 2.2 Concept of Camera Calibration

“Camera Calibration (Camera resectioning) is the process of finding the true parameters of the camera that produced a given photograph or video.” [2]

Nothing is perfect in the world. Same as cameras. Because of the mirror in the camera are not the same, in some case, a beeline in the real world will present as the curve in the image. In this situation, it is necessary to calibrate the cameras. From the result of calibrating the cameras, they should present the same beeline in two images.

### 2.3 Concept of Correspondence problem

“ Given two or more images of the same 3D scene, taken from different points of view, the correspondence problem is to find a set of points in one image which can be identified as the same points in another image.”[3]

## 3 Methodology

### 3.1 Camera Model

Vision begins with the detection of light from the world. That light come from some light source. (e.g. a bulb or the sun), when the light travels and strike some object. Some of the colours will go into human’s eyes. Some of them are not. That can explain why different got different colours. And also , that can explain why people cannot see anything without the light.

The simplest but most useful model is the pinhole model. There are two planes and one light source in the model. Such as Figure 1.

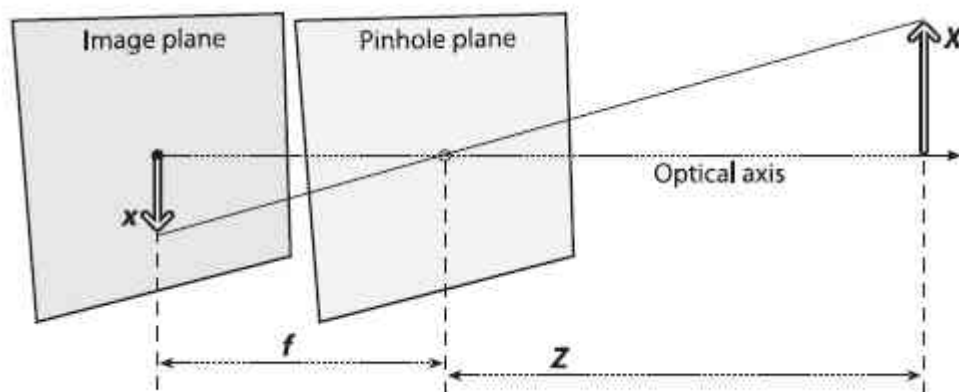


Figure 1 [6]

Figure 1 is the model of pinhole camera. Light is from the scene or a distant object though the pinhole to “project” to the Image plane.  $X$  is the length of the object ,  $x$  is the object’s image on the imaging plane.  $Z$  is the distance from the camera to the object,  $f$  is the focal length of the camera. From the Figure1. We can see by one similar triangles :

$$-x = f \frac{X}{Z}$$

From the Figure1 we can know about that : In the pinhole model , everything are reversed. If we swap the pinhole and the image plane. The object image now is rightside up. Such as Figure2.

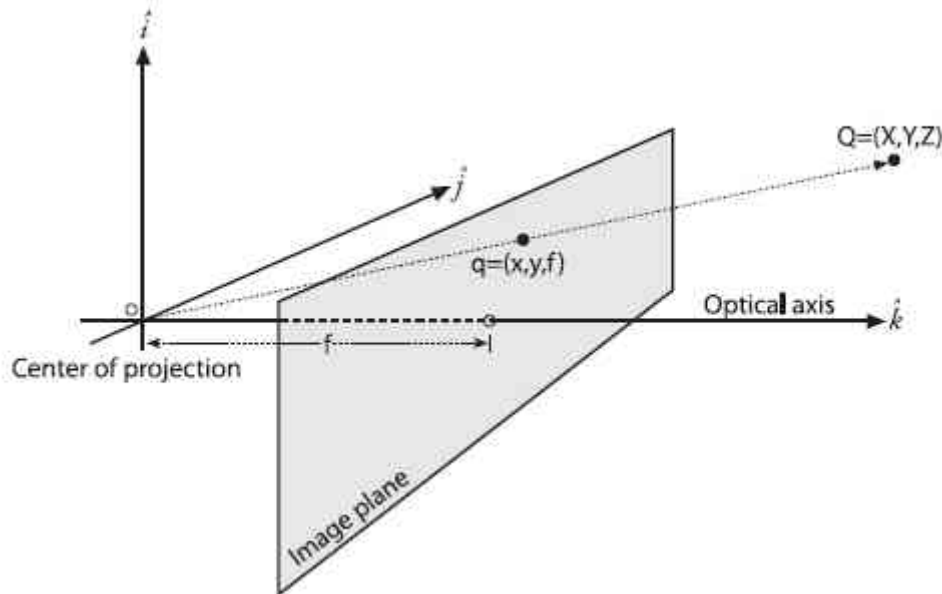


Figure2[6]

O is the Center of projection. In author's opinion , o is the center of projector, the light source. The focal length is f. From Figure2 we can see that if the distance from the image plane to the center of projection is f , then we got the point q(x,y,f) on the image plane; but if we try to increase the distance to point k , then the point Q(X,Y,Z) or Q(X,Y,k). That means if the distance become longer , then the object on the image plane become bigger. From the Figure2 we can see that: (Figure3)

$$x_{\text{screen}} = f_x \left( \frac{X}{Z} \right) + c_x, \quad y_{\text{screen}} = f_y \left( \frac{Y}{Z} \right) + c_y$$

Figure3

Because the center of the chip is usually not on the optical axis , so there introduce two new parameter , Cx and Cy. Those two parameters used to model a possible displacement of the center of coordinates on the projection screen.

Nowadays people usually use a camera to catch scene in the real world. The hardware architecture of camera is simply. It is basic on the pinhole model (Figure4).

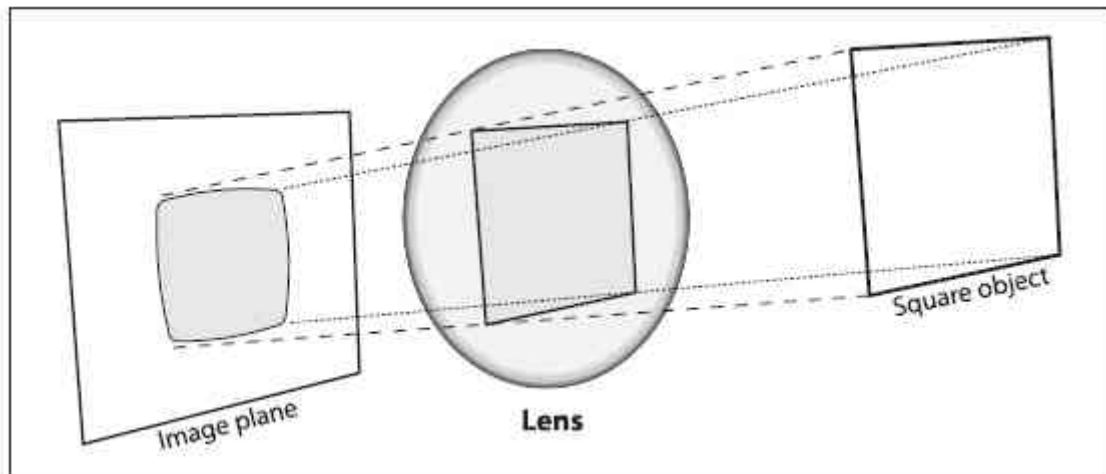


Figure 4 [6]

From the Figure4 we can see that , there is a lens between the image plane and the object. From Figure4 we can see that the object image on the image plane is no longer a square. It is called “barrel” or “Fish-eye” effect. Figure5 is the Radial distortion plot for a particular camera lens. And Figure6 is the Tangential distortion plot for a particular camera lens.

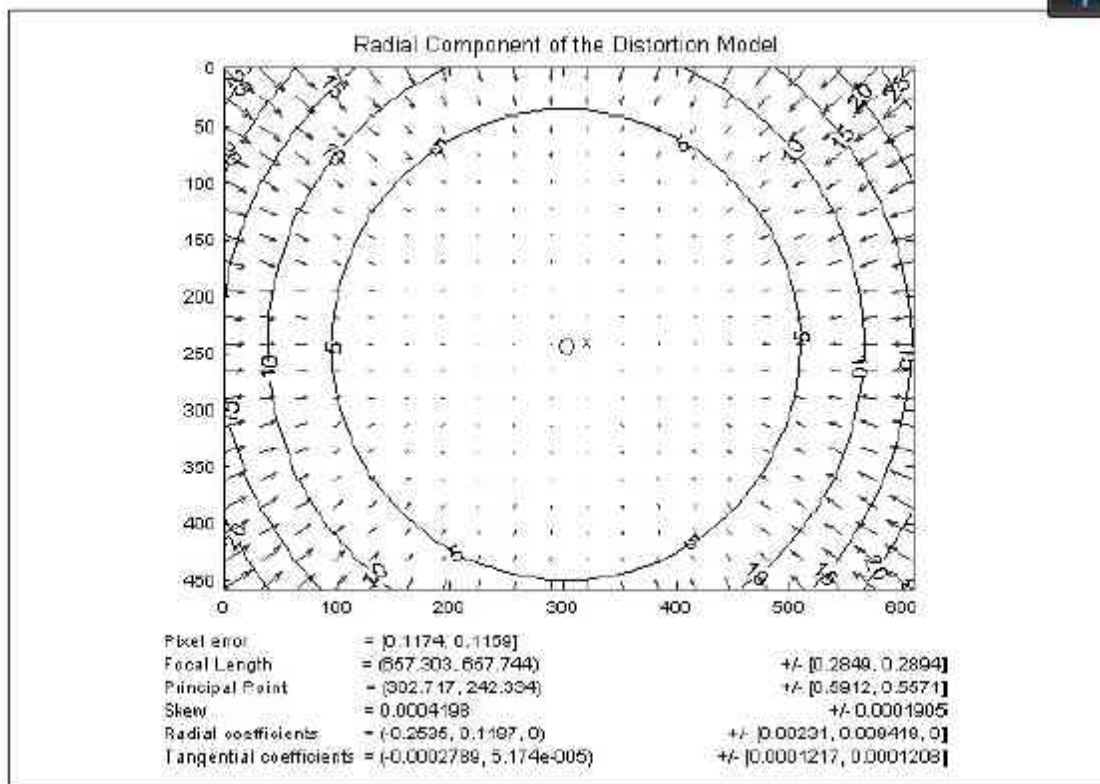


Figure5[6]

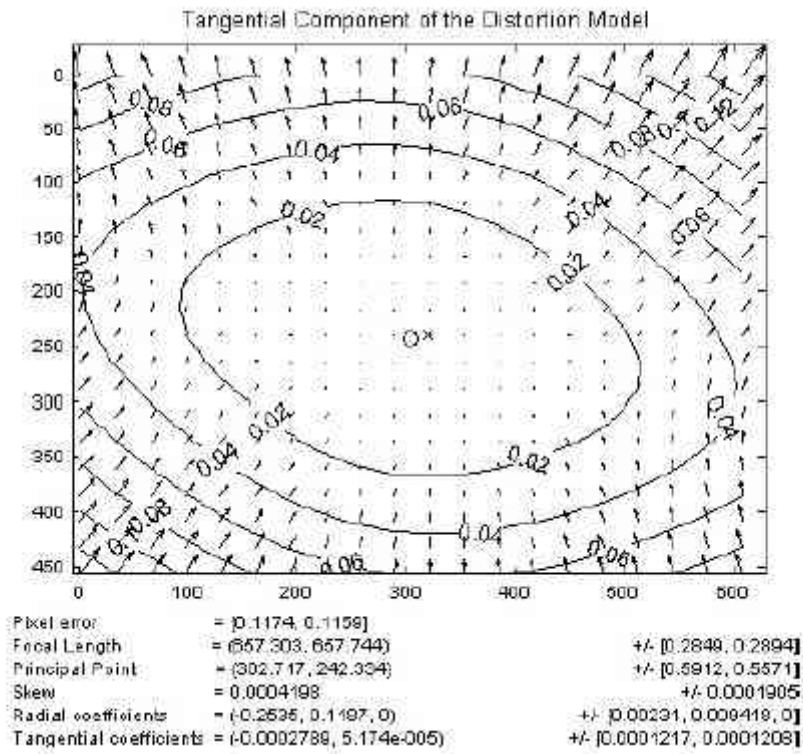


Figure6[6]

The second largest problem is tangential distortion. This distortion is due to manufacturing defects resulting from the lens not being exactly parallel to the imaging plane (Figure7)

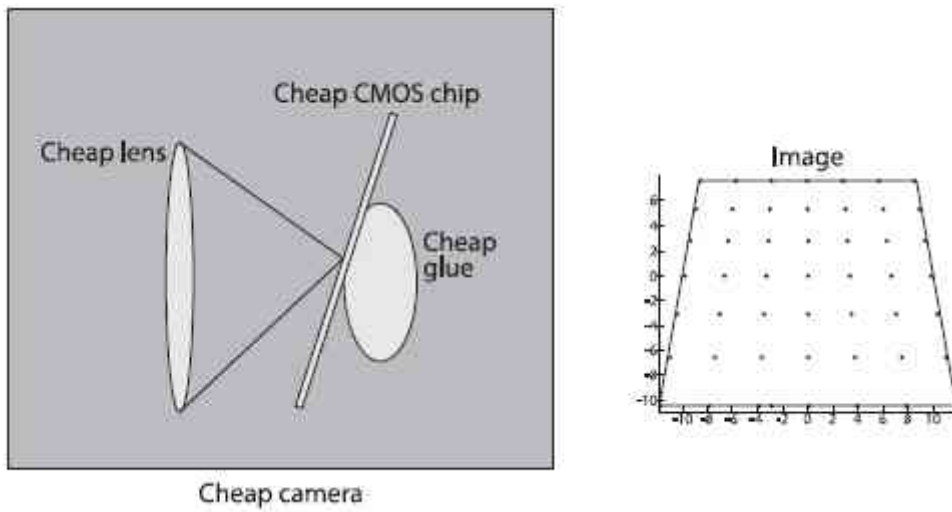


Figure7[6]

Because those distortion properties of the camera , there is necessary to place camera calibration on the image first , in order to get the correct image.

Figure8[6] is the sample of distortion and after un-distortion image



Figure8[6]

### 3.2 Capture Images

Capture Images is the basis for stereo vision. There are many ways to capture images, depending on the occasion and purpose of the application, but also take into account differences in viewpoint , illumination conditions , camera performance and features, characteristics and other factors, in order to facilitate stereo vision calculation. Stereoscopic vision requires two or more images, in order to accurately determine a three-dimensional environment.

In the real world, it is necessary to have a reference point to determine the distance from the observer to the static object. But for the moving objects, it is different. For example , there are two same size objects moving around in front of the observer, one is red , the other is green . Once two moving objects overlap, people can know that if they can see the red object , the red one should be closer to the observer then the green one. In the case of static objects , if two same-size objects shown in the image. One is bigger and the other is smaller. The bigger one should be closer to the observer than the smaller one. According to that , how to capture images is very important for the stereo vision.

Human have two eyes, so the best simulation solution is to build a hardware system with two cameras. In this project the author would like to set up two USB web cameras connected one next to another. The cameras should have capability to capture 640X480 images.

### 3.3 Camera Calibration

The normal scene seen by people is formed by projecting the 3D points in the real-world into the image plane, by using a perspective transformation.

Such like this:

$$s \quad m' = A[R|t]M'$$

or

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

The (X,Y,Z) stand for the world coordinate space. (u,v) are the coordinates of projection points. A is camera matrix, (cx,cy) is the image center. The (fx,fy) stand for the units of focal lengths. If the camera is scale, all of these parameters should be scaled by the same factor. [R|t] stand for matrix of extrinsic parameters. It is used to describe the static scene.

And the parameter can be transform by following formula.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = R \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + t$$

$$x' = x/z$$

$$y' = y/z$$

$$u = fx * x' + cx$$

$$v = fy * y' + cy$$

Nothing is the same in the real world, and nothing is perfect in the real world. So there are usually have some distortions. The model can be extend as

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = R \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + t$$

$$x' = x/z$$

$$y' = y/z$$

$$x'' = x'(1 + k_1r^2 + k_2r^4 + k_3r^6) + 2p_1x'y' + p_2(r^2 + 2x'^2)$$

$$y'' = y'(1 + k_1r^2 + k_2r^4 + k_3r^6) + p_1(r^2 + 2y'^2) + 2p_2x'y'$$

where  $r^2 = x'^2 + y'^2$

$$u = fx * x'' + cx$$

$$v = fy * y'' + cy$$

(k1,k2,k3) are the radial distortion coefficients.(p1,p2) are tangential distortion coefficients. If a image of 640X480 resolution from a calibrated camera, the same distortion coefficients can be used for images of 1280X960 resolution from the same calibrated camera, but fx,fy,cx,cy need to be scaled appropriately.

In principle, any appropriately characterized object can be used as the calibration object. But the most common object in stereo vision is the chessboard. We can hold the chessboard like Figure9[6] to provide enough information to complete the camera calibration.

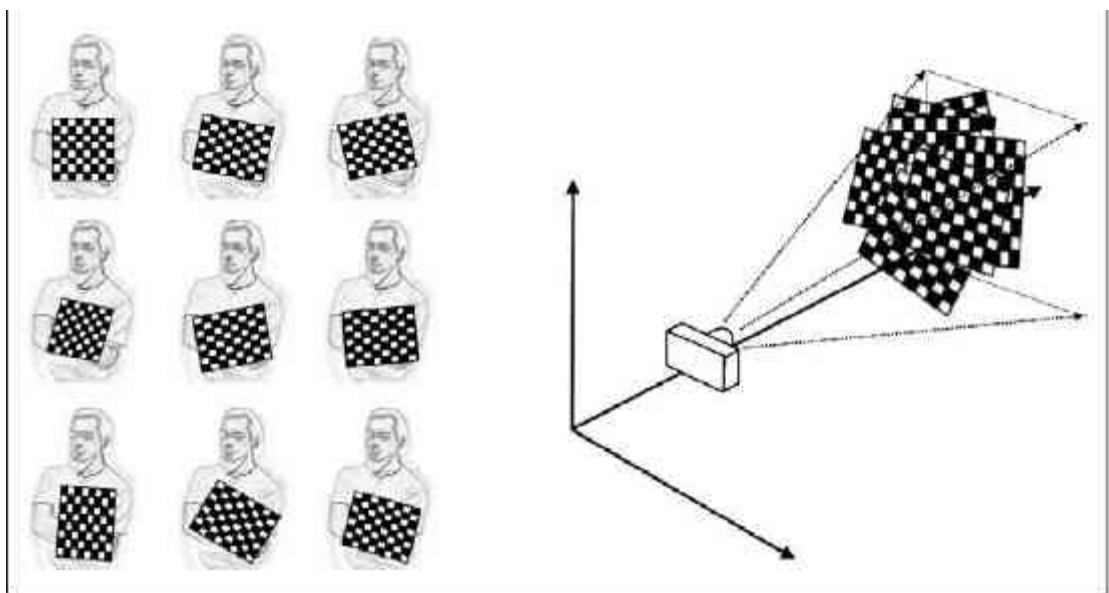


Figure9[6]

Figure10[6] is the sample of the stereo calibration chessboard with cornersX=9 cornersY=6.

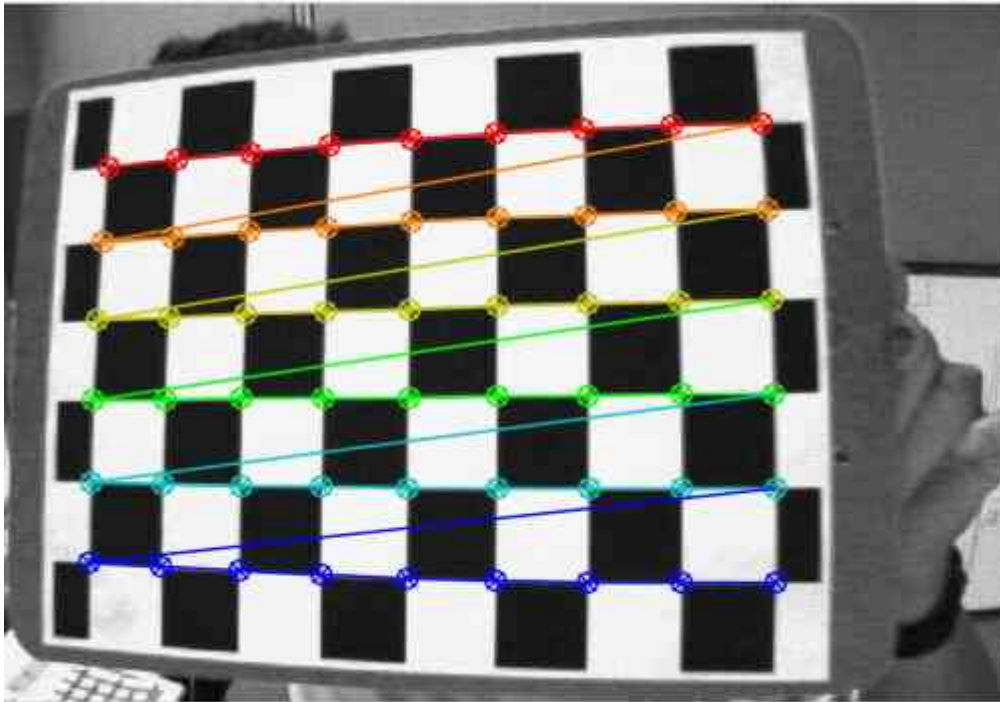


Figure10[6]

### 3.3.1 Hartley's algorithm

Hartley's algorithm used to reduce the computation of compute disparities between the two stereo images when we want to find out the homographies that map the epipoles to infinity. It is not necessary to compute the camera intrinsic information of two cameras because that information is not contained in the match points. It only need to compute the fundamental matrix which can be computed from `cvStereoCalibrate()`;

### 3.4 Correspondence

#### 3.4.1 Stereo correspondence problem

Stereo correspondence means how to find a similar point from image one from image two. Maybe there are some points in image two similar to the point in image one. It is not necessary to search all the points in image two. It is better just search the same area in image two to get it faster. Or can just compute only the overlap areas in the views of two cameras.

If we know all the physical parameter of the cameras we can compute the distance by using triangulated disparity. If not, we can compute depth up to a scale factor. And also we have Hartley's algorithm if we don't have the camera intrinsic.

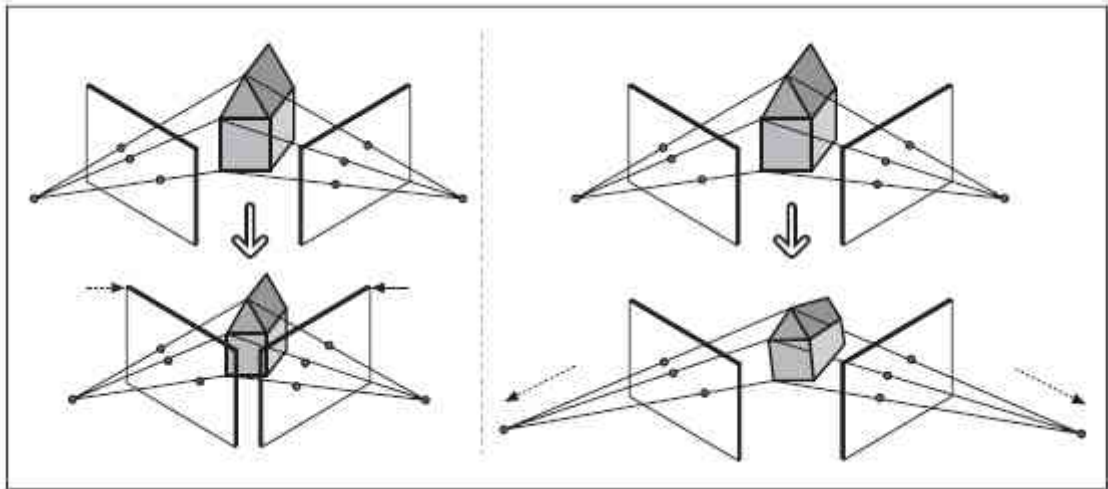


Figure11 [6]

Figure11 shows that the sample of correspondence problem. There are three points in two images. If we try to change the angle of the objects in the images. The correspondence points are totally different. Therefore, in author's opinion, it is not necessary to search the whole image. The point1 in the image one should be in some small area of image2. If we need to speed up the real-time stereo vision, we should use some parallel method here.

#### 1. 3.4.2 Compute depth (Triangulation)

The methodology used for compute depth is show as below.

## Computing $Z$ from the disparity

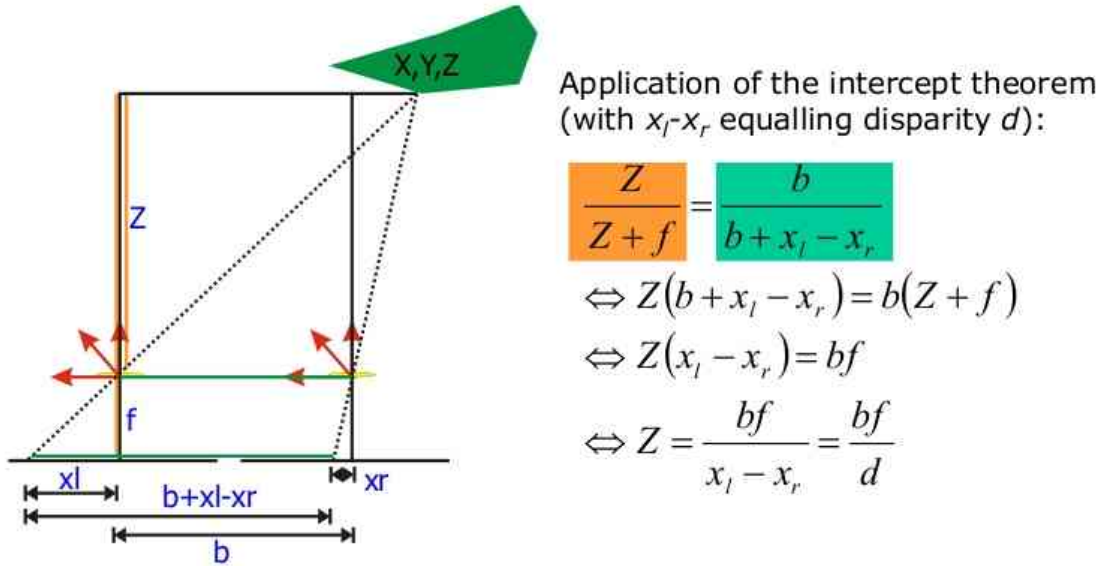


Figure 12: Compute depth [5]

From the Figure12 [5], all the variables are known when build up the environment. We can get depth by using the formula in the Figure1.

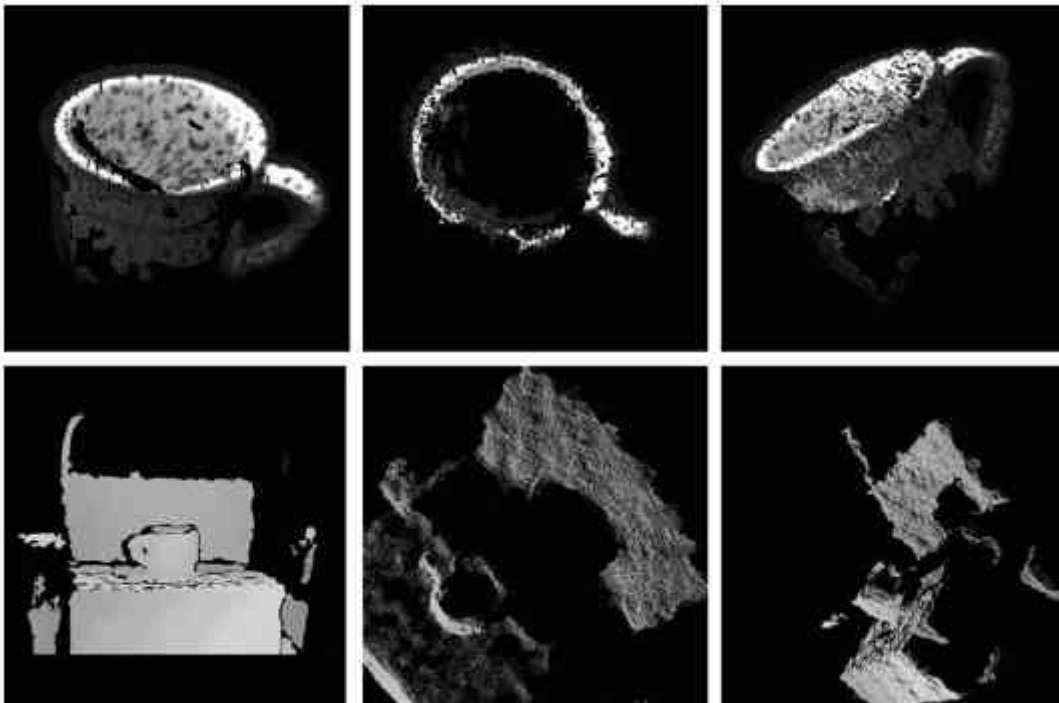


Figure 13[6]

Figure13 [6] is a sample of output (depth map). It shows the distance between camera to the object. The lighter pixels show us that part of object is more near than the darker part. Once we know that information, it can be used to rebuild the 3-D scene.

### **3.4.3 Block Match Algorithm**

A Block Matching Algorithm (BMA) is a way of locating matching blocks in a sequence of digital video frames for the purposes of motion estimation.[4]

Block Matching Algorithm slide a block taken from one image over another image, to approach stereo disparity calculations and motion compensation. Finding the position from one image is similar to another image at a possible offset. But there have a small dynamic range compared to the actual distances to the objects in real world.

## 4 Implementation

### 4.1 Capture Image



Image1

Image2

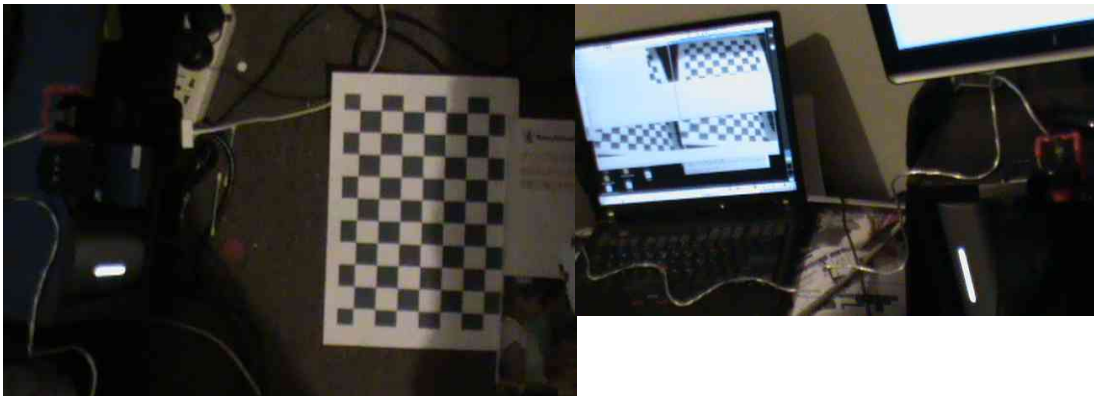


Image3

Image4

### 4.2 Camera Calibration

OpenCV provides a ready-to-use Calibration algorithm. [7]

```
void cvStereoCalibrate(const CvMat* object_points, const CvMat* image_points1, const CvMat*
    image_points2, const CvMat* point_counts, CvMat* camera_matrix1,
    CvMat* dist_coeffs1, CvMat* camera_matrix2, CvMat* dist_coeffs2, CvSize
    image_size, CvMat* R, CvMat* T, CvMat* E=0, CvMat* F=0,
    CvTermCriteria term_crit=cvTermCriteria( CV_TERMCRIT_ITER+CV_TERMCRIT_EPS, 30, 1e-
    6), int flags=CV_CALIB_FIX_INTRINSIC)¶
```

Parameters:

- **object\_points** – The joint matrix of object points, 3xN or Nx3, where N is the total number of points in all views.

- ***image\_points1*** – The joint matrix of corresponding image points in the views from the 1st camera,  $2 \times N$  or  $N \times 2$ , where  $N$  is the total number of points in all views.
- ***image\_points2*** – The joint matrix of corresponding image points in the views from the 2nd camera,  $2 \times N$  or  $N \times 2$ , where  $N$  is the total number of points in all views.
- ***point\_counts*** – Vector containing numbers of points in each view,  $1 \times M$  or  $M \times 1$ , where  $M$  is the number of views.
- ***dist\_coeffs1*, *dist\_coeffs2*** – The input/output vectors of distortion coefficients for each camera, `bgroup({# Pinhole Camera Model, Distortion})bgroup({4x1, 1x4, 5x1 or 1x5.})`
- ***image\_size*** – Size of the image, used only to initialize intrinsic camera matrix.
- ***R*** – The rotation matrix between the 1st and the 2nd cameras' coordinate systems.
- ***T*** – The translation vector between the cameras' coordinate systems.
- ***E*** – The optional output essential matrix.
- ***F*** – The optional output fundamental matrix.
- ***term\_crit*** – Termination criteria for the iterative optimization algorithm.

### 4.3 Find Correspondence

There is a fast and effective block-matching stereo algorithm implemented by OpenCV. The function is called : `cvFindStereoCorrespondenceBM()`. This function is similar to the one developed by Kurt Konolige [Konolige97]. It used a “sum of absolute difference” (SAD) window to find the matching points between the left stereo rectified image and right stereo rectified. The algorithm can finds strongly matching high-texture points between the two rectified images.

- `void cvFindStereoCorrespondenceBM(const CvArr *left, const CvArr *right, CvArr *disparity, CvStereoBMState* state)`
- Parameter “left”: The left one image, it must be single channel 8-bit image.
- Parameter “right”: The right image, it must be same size as left one image.
- Parameter “disparity”: The output single-channel 16-bit signed disparity of the same size as input images.
- Parameter “state”: Stereo correspondence structure, which is create by function `CvStereoBMState`
- The function computes disparity map for the input rectified stereo pair by using block matching algorithm.

## 5 Result and Discussion

### 5.1 Result

#### 5.1.1 Result of Calibration.

Image1 is the image which I used it to calibrate my camera.

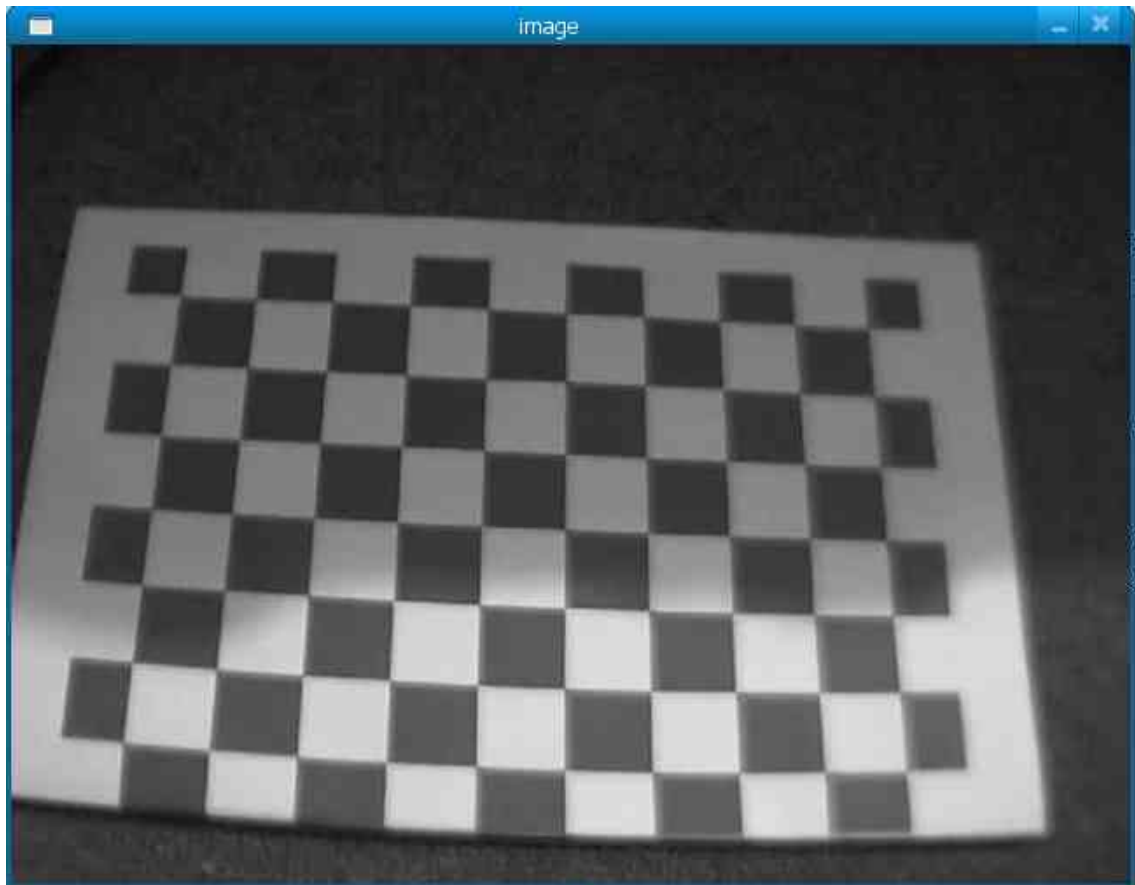


Image5

I use a 10x7 chessboard printed on a paper to calibrate my camera in this project. There are some results of different angle shown below.



Image6 (Center)

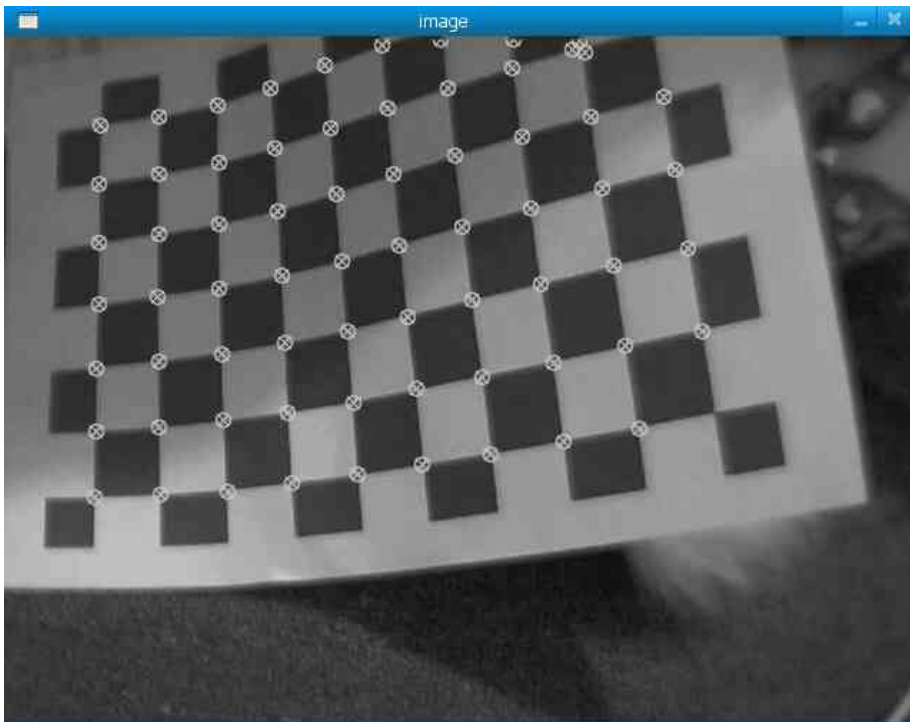


Image7 (Bad capture)

Compare Image2 to Image3. Image2 detected 70 corners but Image3 detected 68 corners. It is not quite accurate if you don't capture the right image.

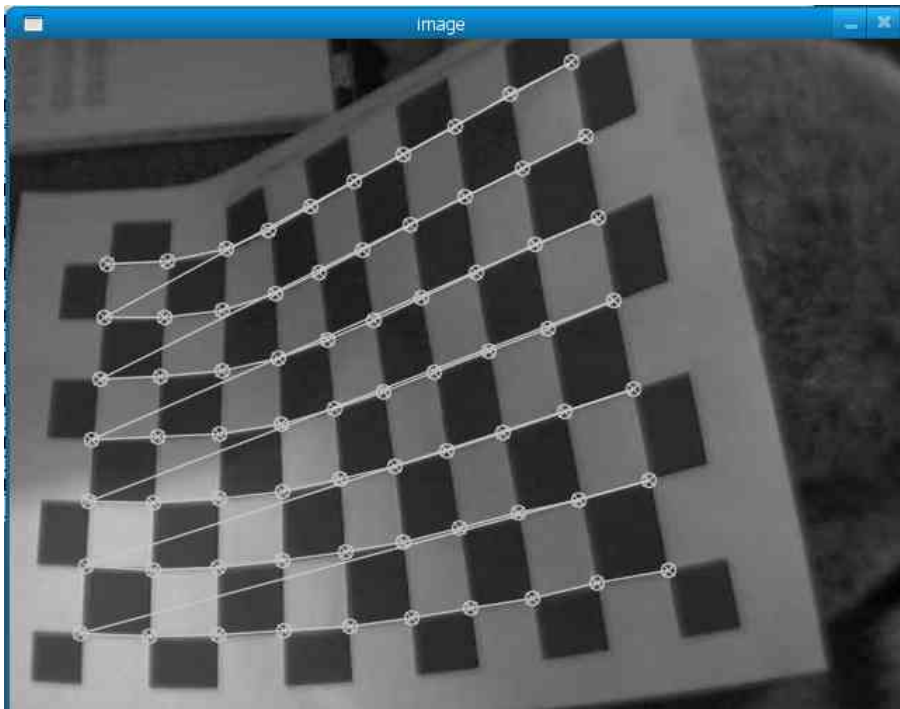


Image8(Right)

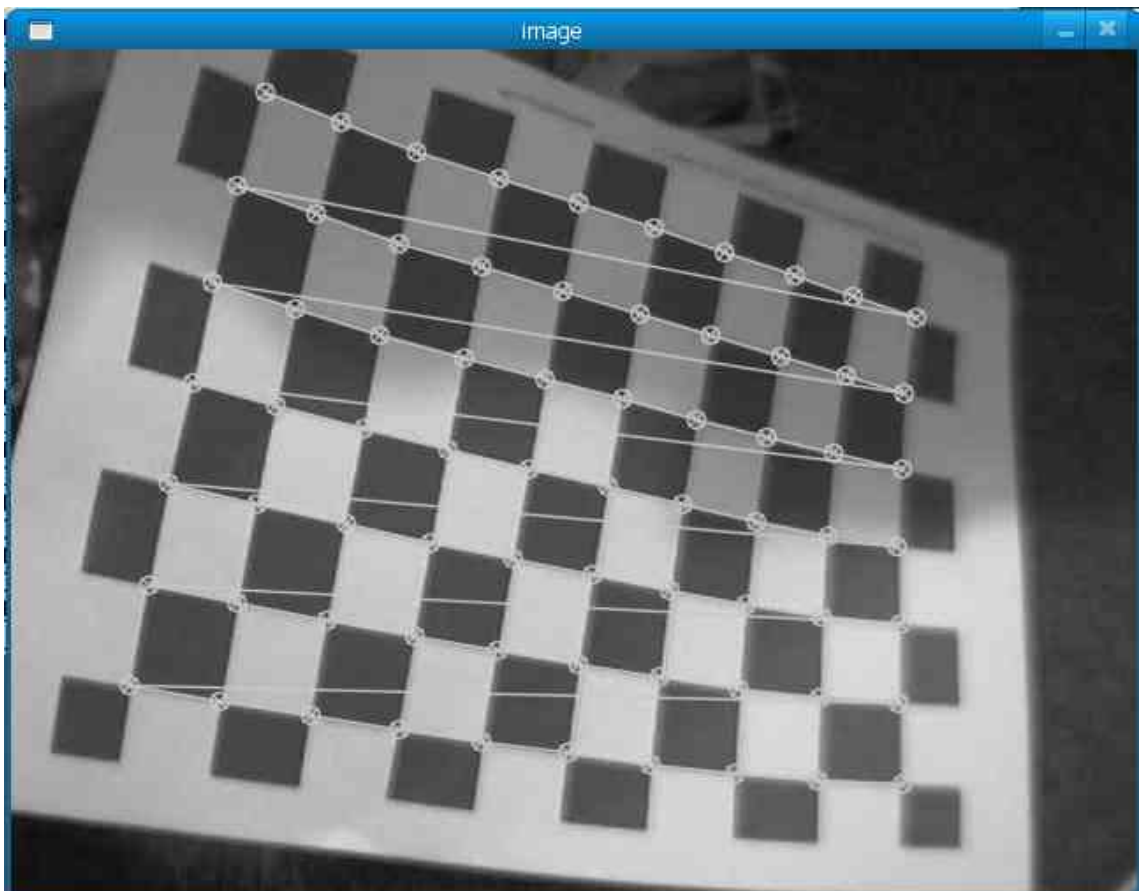


Image9(Left)



Image10

Image11

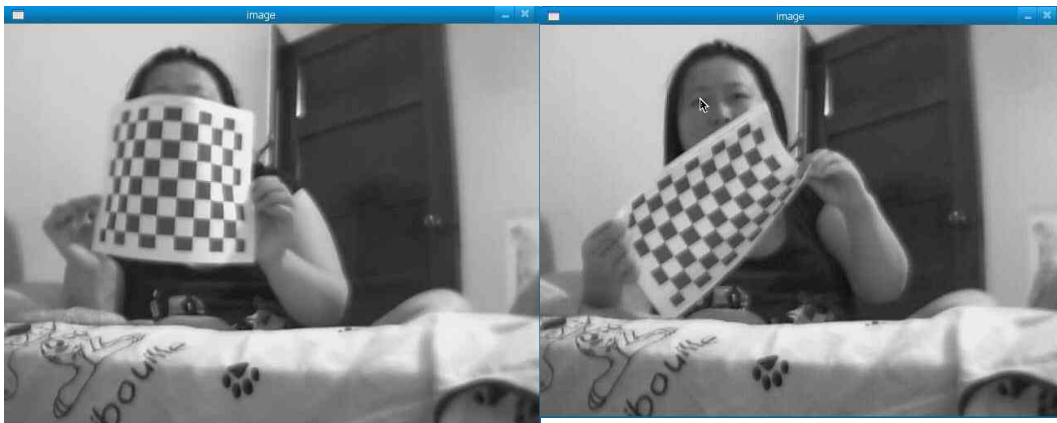


Image12

Image13

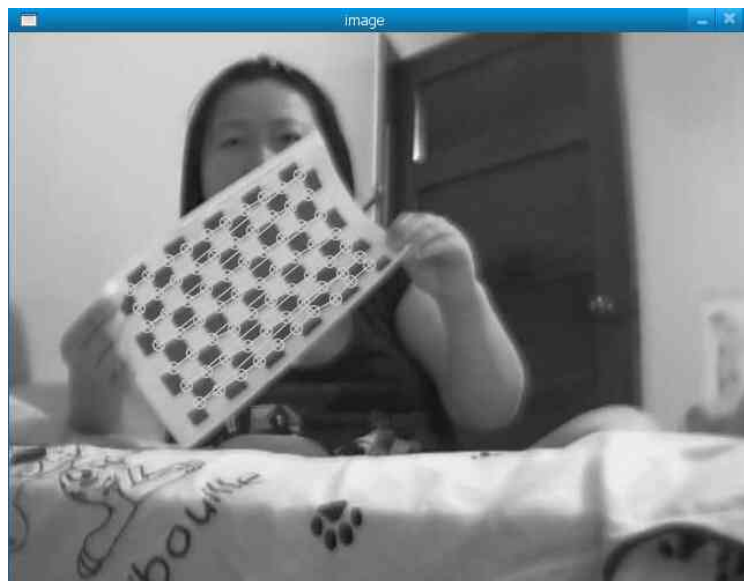


Image14



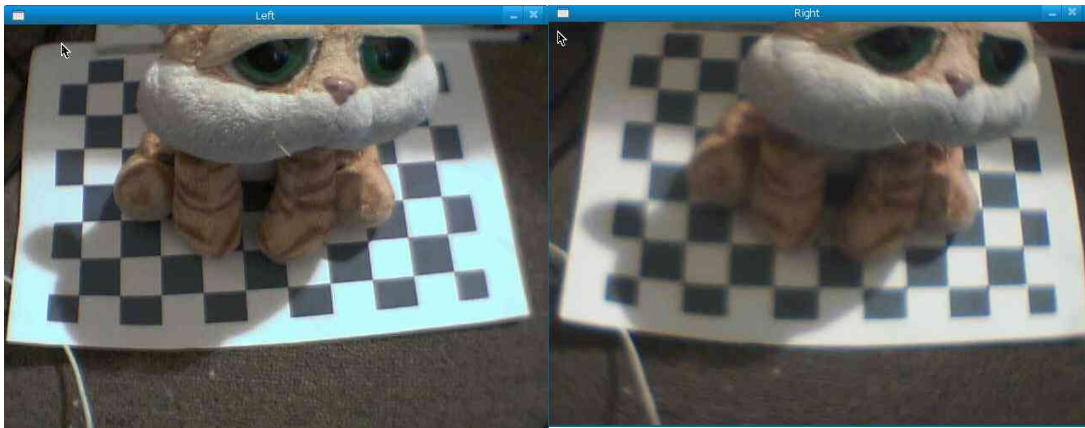


Image17

Image18

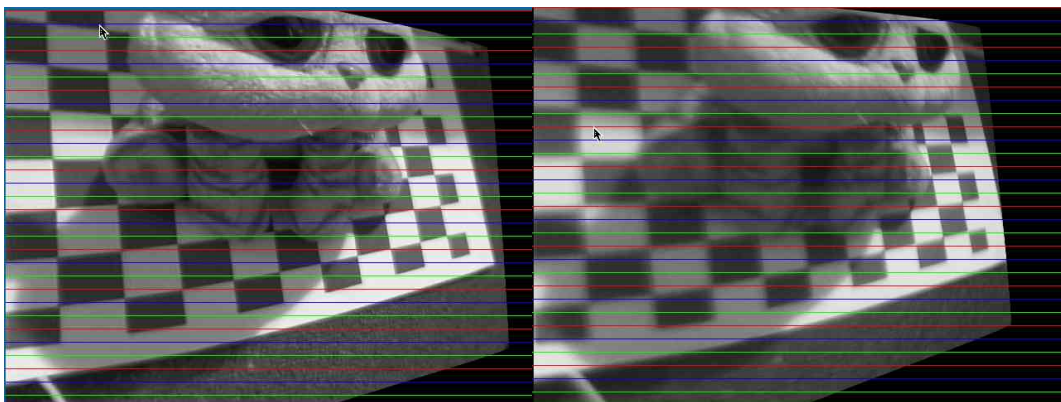


Image19

Image20

Image17 and Image18 are images from cameras and Image19 and Image20 are the results of real time stereo vision. As the qualities of those cameras are not good, that right image is not as good as left one. And the result images are out of shape.

In author's opinion, there are some points to get a better result:

- Get two better cameras
- Get a better environment ( better lighting, adjust the distance of the calibration image from object to the camera)
- Adjust focus point of each camera.

However, I have tried to get a better lighting, but still get the same result. There is no focus adjustment on the cameras.

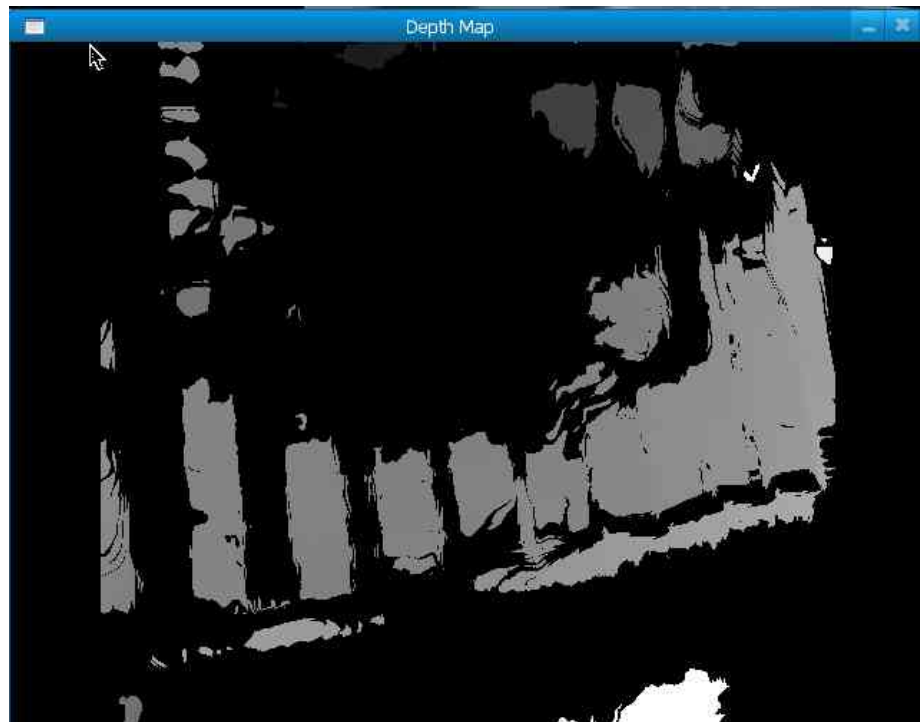


Image21

Image21 is the depth map from the real time stereo vision.

### 5.1.3 Good Cameras vs. Bad Cameras

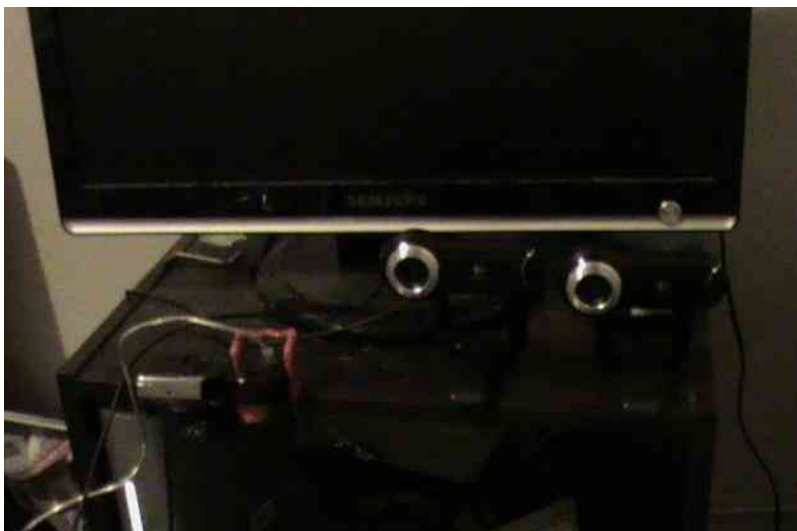


Image22

According to results of real-time stereovision (image19 and image20). The author found that it is necessary to get a better camera to compare with the “old” setting. Image22 is the “new” setting.

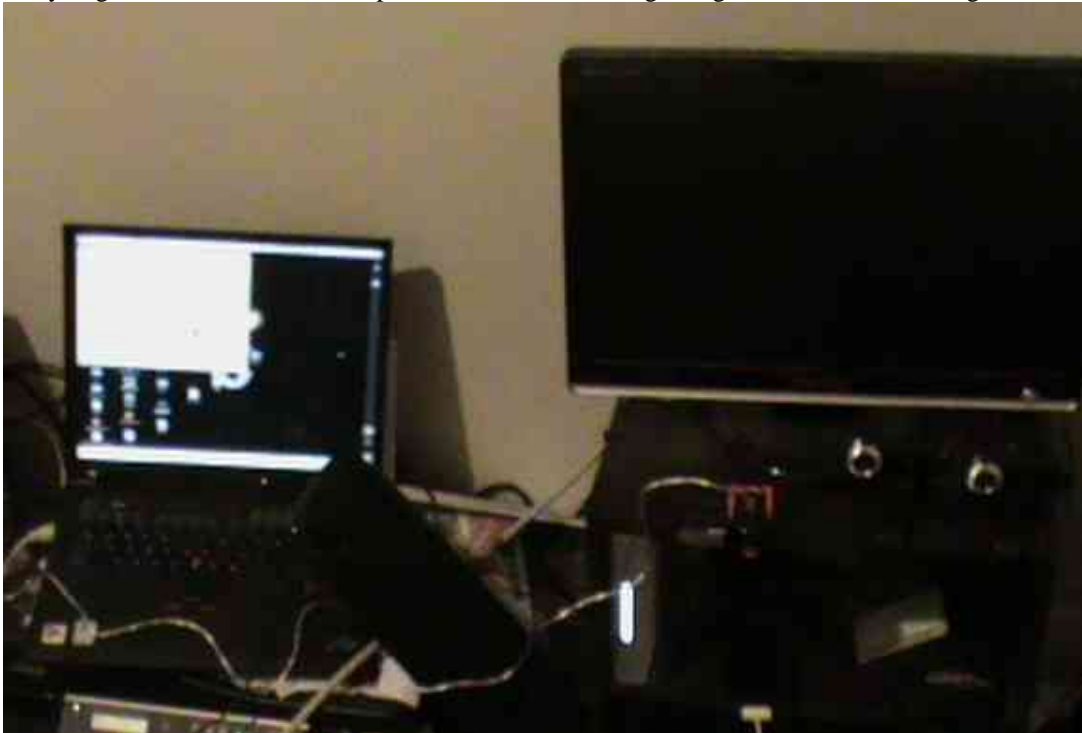


Image23

Image23 shows that both old setting and new setting. The environment is almost the same. Only different is the author change the cameras.



Image24

The new set of cameras include two same brand cameras, and also they are same model. The author put them on a flat desk , adjust them at the same height.

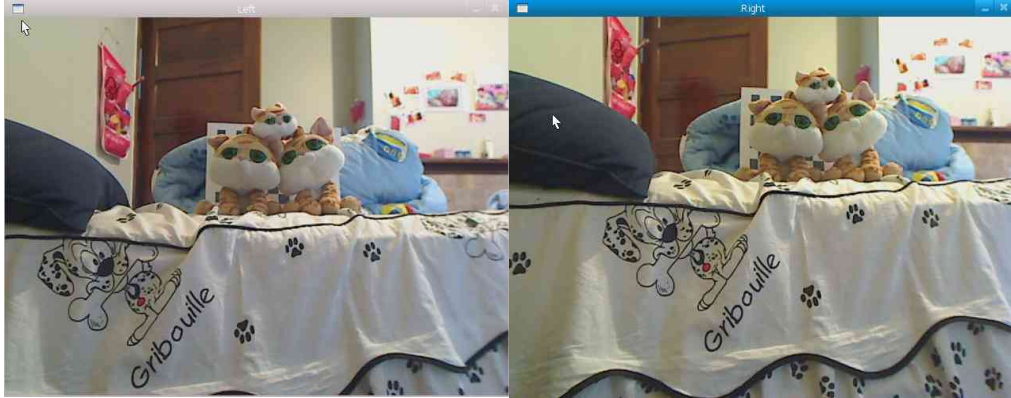


Image25

Image26

Image25 and Image26 are the source images from the cameras.



Image27

Image28

Image19 and Image20 are the results of real time stereo vision.



## Image29

Image29 is the depth map from the real time stereo vision.

The stereo correspondence should improve with the calibration. Image30 and Image31 are the source images from cameras.

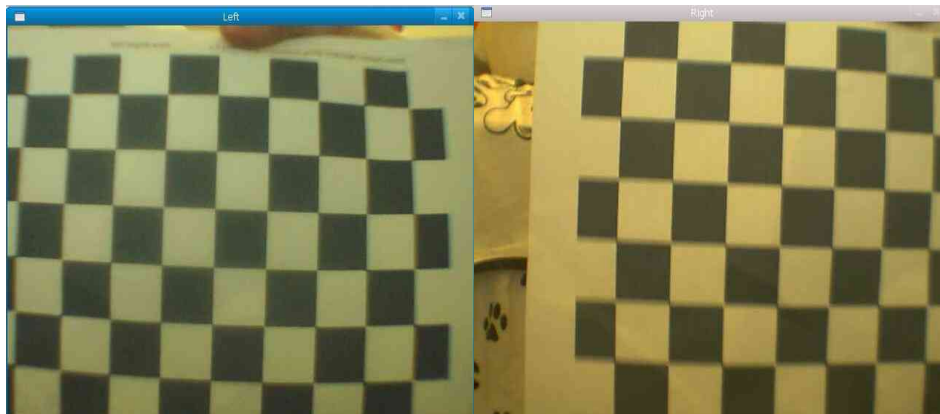


Image30

Image31



Image32

The author replace the OpenCV function `cvStereoRectifyUncalibrated()` with `cvStereoRectify()`, and use BOUGUET'S METHOD instead of Hartley rectification algorithm. Image32 is the result.

From the confrontation, we can know that there are many things can affect a stereo vision system. Lighting, distance of cameras, distance from object to cameras and camera quality.

## 5.2 Speed

Nowadays, stereo vision is not only use in mixing two images together but also used in many areas. For example, we use stereo vision on a robot, to simulate what human can see by their eyes. With this vision, robot can do some tasks for human. Such as pick up one object in the labyrinth, clean the inside of the pipe in some dangerous places. Speed is very important aspect for stereo vision process in real-time. It is necessary to parallel it.

### 5.2.1 History

There are two ways to speed up the real-time stereo vision, the first way is use some hardware to speed up. The other way is to parallel the algorithms for stereo vision.

#### 2. 5.2.1.1 Hardware

There are some existing hardware architectures in the world. One of the hardware architectures is created by J.Falcou, J.serot, T.Chateau and F.Jurie in 2005 [8].

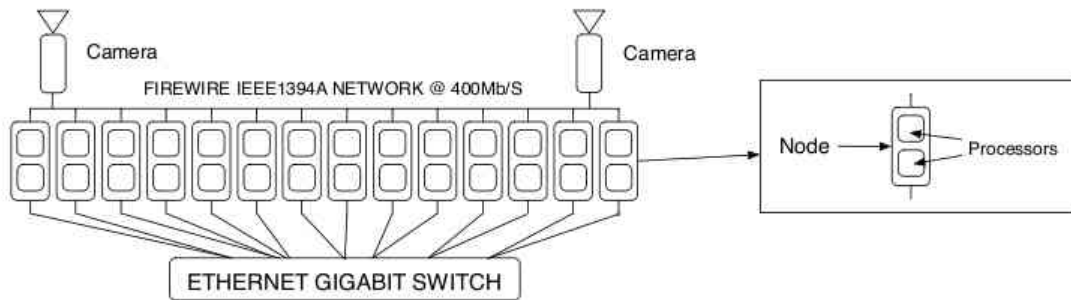


Figure 1: Cluster Architecture.

The Figure 1 ( from [8] ) shows the cluster architecture. It includes 14 computing nodes. Each node has a dual-processor Apple G5 running at 2GHz and 1Gb memory. All nodes connected to a gigabit ethernet switch. And the video streams is provided by two cameras via the FIREWIRE IEEE1394 NETWORK. It has a maximum-speed at 400Mb per second.

The cluster is doing the job what people's brain do. The camera can capture images and provide the video streams enough for the processing. After each node received the images, they can work on the image, get the result and then send back to the master node. The result will present to the user on the output drive such as screen or printer.

But there is a synchronization problem: each node must be work on same frame in the video stream. On the other word, it must wait until each node finished the task on the current frame. In OpenMPI library we can use MPI\_Barrier to make sure all the nodes received the images and work on the right images.

Here's the result of the cluster Architecture:

Step	Seq	np=2	np=4	np=8	np=16	np=24	np = 28
RECTIF	246ms	139.1ms	70.5ms	36.1ms	19.5ms	13.1ms	12.6ms
DETECT	262ms	80.1ms	40.5ms	20.6ms	11.2ms	7.1ms	6.4ms
MATCH	304.2ms	180ms	91.5ms	47.4ms	22.4ms	13.8ms	9.7ms
BUILD	180ms	100ms	53ms	27.5ms	18.2ms	12.0ms	9.5ms
TOTAL	992.2ms	479.2ms	244.6ms	122.6ms	68.2ms	42.9ms	38.2ms
FPS	1.02	2.08	4.08	8.15	14.66	23.31	26.17

Figure 2: Result of Cluster Architecture.[8]

From the result of the cluster Architecture (Figure2 form [8]), if the task process by using single processor. It will take 992.2ms. With 2 nodes it return the result 479.2ms. And it get the 38.2ms when using 28 nodes. That means it almost got 30 times speed up. A normal video got 30fps , and with the 28 nodes on processing the task, it almost got the quality of the real-time video.

### 5.2.1.2 Software

Paralleling in programming takes a very important part in stereo vision. Here are some of the example codes which are working on the Cluster Architecture.

```

struct Data : public MPIData<Data>
{
    Frame input;
    Frame output;
5 };

struct Work : public Task<Work>
{
    bool operator()( Data& d )
10 {
        d.output = where(d.input > 127, 255, 0);
        return true;
    }
};

15 int main(int argc, const char** argv )
{
    Data d;
    Camera camera( 30, res640x480, 8 );
20 Cluster cluster( argc, argv );

    task_list( RowSplit<Frame>, Work, RowMerge<Frame> ) act;
    cluster.task() = (SCM<act>(cluster.root(), cluster.world()));

25 camera >> d.input;
    cluster.run( d );

    return 0;
}

```

Figure 3: Software Architecture, A simple binary thresholding application using in cluster architecture[8]

According to the Figure.3 [8], the software architecture contains a data structure for the input and output. The example code received the frame from the cameras via FIREWIRE Network, create a task on the cluster world then process the task on the cluster. After the node finish processing the data, the nodes will send back the data to the root which means the master, and the master will output the data.

## 6 Conclusion

Stereo vision is an important branch of computer vision. And it has been the focus of the computer vision. It directly simulates the way of human vision processing features.

Nowadays, stereo vision becomes an important part of computer vision. Stereo vision used in many areas of computer vision. People use stereo vision in robots. The robot can use their stereo vision to detect the distance between itself and the target object by the depth map from the result of stereo vision. People use stereo vision in the security system, to join two or more images together, so that can increase the range of vision. And also people can use it to do the 3D reconstruction.

In this paper , the stereo vision system is built by OpenCV library. The OpenCV library is very simple , accurate , high efficiency , can be across multiple platforms and so on. The system can be effectively applied to various computer vision applications.

This system is suitable for the measuring range is not too large and block less scene. For the scene have many blocks , we need to increase the number of cameras , more images which are from different angles. To build the stereo vision scene.

How to build stereo vision in real-time becomes a big problem. Whatever in the case of robot or the case of security system, speed becomes more and more important. And in the paper author mention a parallel algorithms or parallel hardware system to speed up.

## **7 Acknowledgements**

I would like to say thank you to my supervisor Mr. Andre Barczak, thanks to help me to review and support me. I would also want to thank all my lecturers for their teaching and provided support to me during the course of my studies.

## 8 References

- [1] Retrieved 15 MAY 2009. "Stereopsis," in Wikipedia, The free Encyclopedia,  
From [http://en.wikipedia.org/wiki/Stereo\\_vision](http://en.wikipedia.org/wiki/Stereo_vision)
  
- [2] Retrieved 21 JUNE 2009. "Camera resectioning," in Wikipedia, The free Encyclopedia,  
From [http://en.wikipedia.org/wiki/Camera\\_resectioning](http://en.wikipedia.org/wiki/Camera_resectioning)
  
- [3] Retrieved 5 SEPTEMBER 2009. "Correspondence problem," in Wikipedia, The free Encyclopedia,  
From [http://en.wikipedia.org/wiki/Correspondence\\_problem](http://en.wikipedia.org/wiki/Correspondence_problem)
  
- [4] Retrieved 1 JULY 2009. "Block-matching algorithm," in Wikipedia, The free Encyclopedia,  
[http://en.wikipedia.org/wiki/Block-matching\\_algorithm](http://en.wikipedia.org/wiki/Block-matching_algorithm)
  
- [5] Klaus D. Toennies., "5. Stereo Vision (Introduction)," in *3D Computer Vision*, University  
Magdeburg
  
- [6] Gary Bradski and Adrian Kaehler., "Learning OpenCV," in O'Reilly Media, Inc., 1005 Gravenstein  
Highway North, Sebastopol, CA 95472.
  
- [7] Retrieved SEPTEMBER 2009., "Opencv 2.0 Reference," in OpenCV Wiki,  
[http://opencv.willowgarage.com/documentation/camera\\_calibration\\_and\\_3d\\_reconstruction.html](http://opencv.willowgarage.com/documentation/camera_calibration_and_3d_reconstruction.html)

- [8] J. Falcou, J. Serot, T. Chateau, F. Jurie, "A Parallel Implementation of a 3D Reconstruction Algorithm for Real-Time Vision." in *Parallel Computing 2005*, 2005

## **9 Bibliography**

Gary Bradski and Adrian Kaehler., "Learning OpenCV," in O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472,Chapter 11.

## 10 Appendices

### 10.1 Camera Calibration

Stereovision.cpp

```
#include "stereovision.h"
#include "stereocamera.cpp"
#include <stdio.h>
```

```
StereoVision::StereoVision(int imageWidth,int imageHeight)
{
    this->image_Size = cvSize(imageWidth,imageHeight);
    calibrationInited = false;
    calibrationDone = false;
```

```

    CvMat *matrix_x1=0;
    CvMat *matrix_y1=0;
}
```

```
StereoVision::~StereoVision()
{
```

```
}
```

```
void StereoVision::calibrationInit(int cornersX,int cornersY){
    this->cornersX = cornersX;//...
    this->cornersY = cornersY;//...
    this->cornersN = cornersX*cornersY;//number of corners
    tempPoints.resize(cornersN);

    calibrationInited = true;
}
```

```
int StereoVision::calibrate(IplImage* myimage){
```

```
    if(!calibrationInited) return 1;
```

```
    IplImage* image;
    image=myimage;
```

```

CvSize image_Size = cvGetSize(myimage);

        if(image_Size.width != this->image_Size.width || image_Size.height != this-
>image_Size.height)
            return 1;

int cornersDetected = 0;

//find chessboard corners
int result = cvFindChessboardCorners(
    myimage, cvSize(cornersX, cornersY),
    &tempPoints[0], &cornersDetected,
    CV_CALIB_CB_ADAPTIVE_THRESH
);

printf("cornersDetected= %d\n",cornersDetected);

cvDrawChessboardCorners(myimage,cvSize(cornersX,cornersY),&tempPoints[0],cornersD
etected,result);

cvNamedWindow("image",1);
cvShowImage("image",myimage);
cvWaitKey(0);

if(result && cornersDetected == cornersN){

    //find corner sub pix
    cvFindCornerSubPix(
        myimage, &tempPoints[0], cornersDetected,
        cvSize(11, 11), cvSize(-1,-1),
        cvTermCriteria(CV_TERMCRIT_ITER+CV_TERMCRIT_EPS,30, 0.01)
    );
}

    points.resize(cornersN);
    copy( tempPoints.begin(), tempPoints.end(), points.begin() );

calibrationInited = false;

// ARRAY AND VECTOR STORAGE:
// -----

```

```

//R C The rotation matrix between the 1st and the 2nd cameras coordinate systems.
//T C The translation vector between the cameras coordinate systems.
//E C The optional output essential matrix.
//F C The optional output fundamental matrix.
//-----
    double camera_matrix1[3][3], dist_coeffs1[4];
    double R[1][3], T[3];
    CvMat _camera_matrix1,_dist_coeffs1,_R,_T;

    _camera_matrix1 = cvMat(3, 3, CV_64F, camera_matrix1 );

    _dist_coeffs1 = cvMat(1, 4, CV_64F, dist_coeffs1 );

    _R = cvMat(1, 3, CV_64F, R );
    _T = cvMat(1, 3, CV_64F, T );

    // HARVEST CHESSBOARD 3D OBJECT POINT LIST:
    objectPoints.resize(cornersN);

    for(int i = 0; i < cornersY; i++ )
        for(int j = 0; j < cornersX; j++)
            objectPoints[ i*cornersX + j] = cvPoint3D32f(i, j, 0);

    npoints.resize(1,cornersN);

    int N = cornersN;

    CvMat _objectPoints = cvMat(1, N, CV_32FC3, &objectPoints[0] );
    CvMat _imagePoints1 = cvMat(1, N, CV_32FC2, &points[0] );
    CvMat _npoints = cvMat(1, npoints.size(), CV_32S, &npoints[0] );
    cvSetIdentity(&_camera_matrix1);
    cvZero(&_dist_coeffs1);

    //calibrate camera
    cvCalibrateCamera2(&_objectPoints,&_imagePoints1,&_npoints,image_Size,&_camera_matrix1
, &_dist_coeffs1, &_R, &_T,0);

    calibrationDone = true;

    cvWaitKey(0);

    return 0;

```

```

}

int main(int argc, char *argv[]){
StereoVision *sv=new StereoVision(640,480);
StereoCamera *camera=new StereoCamera();
if (camera->setup(cvSize(640,480))==0){
printf("Doing...");
sv->calibrationInit(10,7);
camera->capture();
sv->calibrate(camera->getFramesGray());
}
else printf("error");
}

```

Stereovision.h

```

#ifndef STEREOVISION_H
#define STEREOVISION_H

```

```

#include "cv.h"
#include "cxmisc.h"
#include "cvaux.h"
#include "highgui.h"
#include <vector>

```

```

class StereoVision
{
private:

```

```

int cornersX,cornersY,cornersN;

```

```

CvSize image_Size;

```

```

int imageWidth;

```

```

int imageHeight;

```

```

std::vector<CvPoint2D32f> tempPoints;

```

```

std::vector<CvPoint3D32f> objectPoints;

```

```

std::vector<CvPoint2D32f> points;

```

```

std::vector<int> npoints;

```

```
int sample_Count;
bool calibrationInited;
bool calibrationDone;

public:
    StereoVision(int imageWidth,int imageHeight);
    ~StereoVision();

    CvMat *matrix_x1;
    CvMat *matrix_y1;

    void calibrationInit(int cornersX,int cornersY);
    int calibrate(IplImage* myimage);
    int calibrationEnd();

};

#endif // STEREOVISION_H
```

Stereocamera.h

```
#ifndef STEREOCAMERA_H
#define STEREOCAMERA_H

#include "cv.h"
#include "cxmisc.h"
#include "cvaux.h"
#include "highgui.h"
#include <vector>

class StereoCamera
{
    CvCapture* captures;

    CvSize imageSize;

public:

    IplImage* frames;
    IplImage* framesGray;
```

```

StereoCamera();
~StereoCamera();
int setup(CvSize imageSize);
bool ready;
int capture();
IplImage* getFramesGray();

};

#endif // STEREOCAMERA_H

Stereocamera.cpp

#include "stereocamera.h"

StereoCamera::StereoCamera()
{
    captures = 0;
    frames = 0;
    framesGray = 0;

    ready = false;
}

StereoCamera::~StereoCamera()
{
    cvReleaseImage(&frames);
    cvReleaseImage(&framesGray);
    cvReleaseCapture(&captures);
}

int StereoCamera::setup(CvSize imageSize){
    this->imageSize = imageSize;

    captures = cvCaptureFromCAM(0);
    //CV_CAP_DSHOW+

    if(captures){
        cvSetCaptureProperty(captures
,CV_CAP_PROP_FRAME_WIDTH,imageSize.width);

        ready = true;
        return 0;

```

```
    }else{
        ready = false;
        return 1;
    }
}

int StereoCamera::capture(){
    frames = cvQueryFrame(captures);

    if (captures) return 0;
    else return 1;
}

IplImage* StereoCamera::getFramesGray(){
    if(!frames) return 0;
    if(frames->depth == 1){
        framesGray = frames;
        return frames;
    }else{
        if(0 == framesGray) framesGray = cvCreateImage(cvGetSize(frames),IPL_DEPTH_8U,1);
        cvCvtColor(frames,framesGray,CV_BGR2GRAY);
        return framesGray;
    }
}
```

## 10.2 Real-time Stereo Vision

Stereovision.cpp

```
#include "stereovision.h"  
#include "stereocamera.cpp"
```

```
StereoVision::StereoVision(int imageWidth,int imageHeight)  
{
```

```
    this->image_Size = cvSize(imageWidth,imageHeight);  
    matrix_x1 = 0;  
    matrix_y1 = 0;  
    matrix_x2 = 0;  
    matrix_y2 = 0;  
    calibrationInited = false;  
    calibrationDone = false;  
    images_Rectified[0] = 0;  
    images_Rectified[1] = 0;  
    image_Depth = 0;  
    image_DepthNormalized = 0;  
    image_Depth = 0;
```

```
}
```

```
StereoVision::~StereoVision()
```

```
{  
    cvReleaseMat(&images_Rectified[0]);  
    cvReleaseMat(&images_Rectified[1]);  
    cvReleaseMat(&image_DepthNormalized);  
    cvReleaseMat(&image_Depth);
```

```
}
```

```
void StereoVision::calibrationInit(int cornersX,int cornersY){
```

```
    this->cornersX = cornersX;//...  
    this->cornersY = cornersY;//...  
    this->cornersN = cornersX*cornersY;//number of corners  
    tempPoints[0].resize(cornersN);  
    tempPoints[1].resize(cornersN);  
    sample_Count = 0;  
    calibrationInited = true;
```

```
}
```

```
int StereoVision::calibrate(IplImage* imageLeft,IplImage* imageRight){
```

```

if(!calibrationInited) return 1;

IplImage* image[2];
image[0]=imageLeft;
image[1]=imageRight;

int succeses = 0;

for(int i=0;i<2;i++){
    CvSize image_Size = cvGetSize(image[i]);

    if(image_Size.width != this->image_Size.width || image_Size.height != this-
>image_Size.height)
        return 1;

    int cornersDetected = 0;

    //find chessboard corners
    int result = cvFindChessboardCorners(
        image[i], cvSize(cornersX, cornersY),
        &tempPoints[i][0], &cornersDetected,
        CV_CALIB_CB_ADAPTIVE_THRESH | CV_CALIB_CB_NORMALIZE_IMAGE
    );

    if(result && cornersDetected == cornersN){

        //find corner sub pix
        cvFindCornerSubPix(
            image[i], &tempPoints[i][0], cornersDetected,
            cvSize(11, 11), cvSize(-1,-1),
            cvTermCriteria(CV_TERMCRIT_ITER+CV_TERMCRIT_EPS,30, 0.01)
        );
        succeses++;
    }
}

if(succeses==2){
    for(int i=0;i<2;i++){
        points[i].resize((sample_Count+1)*cornersN);
        copy( tempPoints[i].begin(), tempPoints[i].end(), points[i].begin() +
sample_Count*cornersN);
    }
}

```

```

        sample_Count++;
        return 0;
    }else{
        return 1;
    }
}

int StereoVision::calibrationEnd(){
    calibrationInited = false;

    // ARRAY AND VECTOR STORAGE:
    // -----
    //R C The rotation matrix between the 1st and the 2nd cameras coordinate systems.
    //T C The translation vector between the cameras coordinate systems.
    //E C The optional output essential matrix.
    //F C The optional output fundamental matrix.
    //-----

    double camera_matrix1[3][3], camera_matrix2[3][3], dist_coeffs1[5], dist_coeffs2[5];
    double R[3][3], T[3], E[3][3], F[3][3];
    CvMat _camera_matrix1,_camera_matrix2,_dist_coeffs1,_dist_coeffs2,_R,_T,_E,_F;

    _camera_matrix1 = cvMat(3, 3, CV_64F, camera_matrix1 );
    _camera_matrix2 = cvMat(3, 3, CV_64F, camera_matrix2 );
    _dist_coeffs1 = cvMat(1, 5, CV_64F, dist_coeffs1 );
    _dist_coeffs2 = cvMat(1, 5, CV_64F, dist_coeffs2 );
    _R = cvMat(3, 3, CV_64F, R );
    _T = cvMat(3, 1, CV_64F, T );
    _E = cvMat(3, 3, CV_64F, E );
    _F = cvMat(3, 3, CV_64F, F );

    // HARVEST CHESSBOARD 3D OBJECT POINT LIST:
    objectPoints.resize(sample_Count*cornersN);

    for(int k=0;k<sample_Count;k++)
        for(int i = 0; i < cornersY; i++ )
            for(int j = 0; j < cornersX; j++ )
                objectPoints[k*cornersY*cornersX + i*cornersX + j] = cvPoint3D32f(i, j, 0);

    npoints.resize(sample_Count,cornersN);

    int N = sample_Count * cornersN;

```

```

CvMat _objectPoints = cvMat(1, N, CV_32FC3, &objectPoints[0] );
CvMat _imagePoints1 = cvMat(1, N, CV_32FC2, &points[0][0] );
CvMat _imagePoints2 = cvMat(1, N, CV_32FC2, &points[1][0] );
CvMat _npoints = cvMat(1, npoints.size(), CV_32S, &npoints[0] );
cvSetIdentity(&_camera_matrix1);
cvSetIdentity(&_camera_matrix2);
cvZero(&_dist_coeffs1);
cvZero(&_dist_coeffs2);

//calibrate stereo camera
cvStereoCalibrate( &_objectPoints, &_imagePoints1,
    &_imagePoints2, &_npoints,
    &_camera_matrix1, &_dist_coeffs1, &_camera_matrix2, &_dist_coeffs2,
    image_Size, &_R, &_T, &_E, &_F,
    cvTermCriteria(CV_TERMCRIT_ITER+CV_TERMCRIT_EPS, 100, 1e-5),
    CV_CALIB_FIX_ASPECT_RATIO + CV_CALIB_ZERO_TANGENT_DIST +
CV_CALIB_SAME_FOCAL_LENGTH
    );

//work in undistorted space
    cvUndistortPoints( &_imagePoints1, &_imagePoints1,&_camera_matrix1, &_dist_coeffs1, 0,
    &_camera_matrix1 );
    cvUndistortPoints( &_imagePoints2, &_imagePoints2,&_camera_matrix2, &_dist_coeffs2, 0,
    &_camera_matrix2 );

//rectification

double R1[3][3], R2[3][3];
CvMat _R1 = cvMat(3, 3, CV_64F, R1);
CvMat _R2 = cvMat(3, 3, CV_64F, R2);

//Hartley rectification algorithm
double H1[3][3], H2[3][3], iM[3][3];
CvMat _H1 = cvMat(3, 3, CV_64F, H1);
CvMat _H2 = cvMat(3, 3, CV_64F, H2);
CvMat _iM = cvMat(3, 3, CV_64F, iM);

cvStereoRectifyUncalibrated(
    &_imagePoints1,&_imagePoints2, &_F,
    image_Size,
    &_H1, &_H2, 3
    );

```

```

cvInvert(&_camera_matrix1, &_iM);
cvMatMul(&_H1, &_camera_matrix1, &_R1);
cvMatMul(&_iM, &_R1, &_R1);
cvInvert(&_camera_matrix2, &_iM);
cvMatMul(&_H2, &_camera_matrix2, &_R2);
cvMatMul(&_iM, &_R2, &_R2);

//Prepare cvRemap()
cvReleaseMat(&matrix_x1);
cvReleaseMat(&matrix_y1);
cvReleaseMat(&matrix_x2);
cvReleaseMat(&matrix_y2);
matrix_x1 = cvCreateMat( image_Size.height,image_Size.width, CV_32F );
matrix_y1 = cvCreateMat( image_Size.height,image_Size.width, CV_32F );
matrix_x2 = cvCreateMat( image_Size.height,image_Size.width, CV_32F );
matrix_y2 = cvCreateMat( image_Size.height,image_Size.width, CV_32F );

cvInitUndistortRectifyMap(&_camera_matrix1,&_dist_coeffs1,&_R1,&_camera_matrix1,matrix
_x1,matrix_y1);
cvInitUndistortRectifyMap(&_camera_matrix2,&_dist_coeffs1,&_R2,&_camera_matrix2,matrix
_x2,matrix_y2);

calibrationDone = true;

return 0;
}

int StereoVision::stereoProcess(CvArr* imageSrcLeft,CvArr* imageSrcRight){
    if(!calibrationDone) return 1;

    if(!images_Rectified[0]) images_Rectified[0] =
cvCreateMat( image_Size.height,image_Size.width, CV_8U );
    if(!images_Rectified[1]) images_Rectified[1] =
cvCreateMat( image_Size.height,image_Size.width, CV_8U );
    if(!image_Depth) image_Depth = cvCreateMat( image_Size.height,image_Size.width, CV_16S );
    if(!image_DepthNormalized) image_DepthNormalized =
cvCreateMat( image_Size.height,image_Size.width, CV_8U );

//rectify images
cvRemap( imageSrcLeft, images_Rectified[0] , matrix_x1, matrix_y1 );
cvRemap( imageSrcRight, images_Rectified[1] , matrix_x2, matrix_y2 );

```

```

CvStereoBMState *BMState = cvCreateStereoBMState();
BMState->preFilterSize=41;
BMState->preFilterCap=31;
BMState->SADWindowSize=41;
BMState->minDisparity=-64;
BMState->numberOfDisparities=128;
BMState->textureThreshold=10;
BMState->uniquenessRatio=15;

    cvFindStereoCorrespondenceBM( images_Rectified[0], images_Rectified[1], image_Depth,
BMState);
    cvNormalize( image_Depth, image_DepthNormalized, 0, 256, CV_MINMAX );

    cvReleaseStereoBMState(&BMState);

    return 0;
}

int StereoVision::calibrationSave(const char* filename){
    if(!calibrationDone) return 1;
    FILE* f = fopen(filename,"wb");
    if(!f) return 1;
    if(!fwrite(matrix_x1->data.fl,sizeof(float),matrix_x1->rows*matrix_x1->cols,f)) return 1;
    if(!fwrite(matrix_y1->data.fl,sizeof(float),matrix_y1->rows*matrix_y1->cols,f)) return 1;
    if(!fwrite(matrix_x2->data.fl,sizeof(float),matrix_x2->rows*matrix_x2->cols,f)) return 1;
    if(!fwrite(matrix_y2->data.fl,sizeof(float),matrix_y2->rows*matrix_y2->cols,f)) return 1;
    fclose(f);
    return 0;
}

// int getsample_Count();

int StereoVision::calibrationLoad(const char* filename){
    cvReleaseMat(&matrix_x1);
    cvReleaseMat(&matrix_y1);
    cvReleaseMat(&matrix_x2);
    cvReleaseMat(&matrix_y2);
    matrix_x1 = cvCreateMat( image_Size.height,image_Size.width, CV_32F );
    matrix_y1 = cvCreateMat( image_Size.height,image_Size.width, CV_32F );
    matrix_x2 = cvCreateMat( image_Size.height,image_Size.width, CV_32F );
    matrix_y2 = cvCreateMat( image_Size.height,image_Size.width, CV_32F );
    FILE* f = fopen(filename,"rb");
    if(!f) return 1;

```

```

    if(!fread(matrix_x1->data.fl,sizeof(float),matrix_x1->rows*matrix_x1->cols,f)) return 1;
    if(!fread(matrix_y1->data.fl,sizeof(float),matrix_y1->rows*matrix_y1->cols,f)) return 1;
    if(!fread(matrix_x2->data.fl,sizeof(float),matrix_x2->rows*matrix_x2->cols,f)) return 1;
    if(!fread(matrix_y2->data.fl,sizeof(float),matrix_y2->rows*matrix_y2->cols,f)) return 1;
    fclose(f);
    calibrationDone = true;
    return 0;
}

```

```

int main(int argc,char *argv[]){
StereoVision *sv=new StereoVision(640,480);
StereoCamera *camera = new StereoCamera();
int mode=0;//1 for calibrate , 0 for real-time
int realtime=0; // 1 for realtime
int calcnt=10;
printf("Setting up camera");
if (!camera->setup(cvSize(640,480))){
printf ("Camera Ready!\n");
    cvNamedWindow("Left",1);
cvNamedWindow("Right",1);
cvNamedWindow("Rectified Image",1);
cvNamedWindow("Depth Map",1);

    if (mode){
        sv->calibrationInit(10,7);//x=10 y=7
        while(sv->getsample_Count()<calcnt){
            camera->capture();
            sv->calibrate(camera->getFramesGray(0),camera->getFramesGray(1));
            printf("Calibration : %d\n",sv->getsample_Count());
        }

        sv->calibrationEnd();
        printf("Finish Calibration\n");
        sv->calibrationSave("rstcal");
        printf("Result Saved!\n");
    }

    else{
        sv->calibrationLoad("rstcal");
        printf("Result Load!\n");
        CvMat *image_RectifiedPair;
        CvMat part;
        while(!camera->capture()){
            cvShowImage("Left",camera->frames[0]);

```

```

cvShowImage("Right",camera->frames[1]);

CvSize img_size=sv->getimage_Size();

image_RectifiedPair = cvCreateMat( img_size.height, img_size.width*2,CV_8UC3 );

sv->stereoProcess(camera->getFramesGray(0), camera->getFramesGray(1));

printf("Size:%dx%d",sv->getimage_Size().width,sv->getimage_Size().height);
cvGetCols( image_RectifiedPair, &part, 0, img_size.width );
cvCvtColor( sv->images_Rectified[0], &part, CV_GRAY2BGR );
cvGetCols( image_RectifiedPair, &part, img_size.width,img_size.width*2 );
cvCvtColor( sv->images_Rectified[1], &part, CV_GRAY2BGR );

for(int j = 0; j < img_size.height; j += 16 )
    cvLine( image_RectifiedPair, cvPoint(0,j),cvPoint(img_size.width*2,j),CV_RGB((j%3)?
0:255,((j+1)%3)?0:255,((j+2)%3)?0:255));
    cvShowImage( "Rectified Image", image_RectifiedPair );
    cvShowImage( "Depth Map", sv->image_DepthNormalized );

printf("Renew\n");
if(cvWaitKey(1)==27)
    break;
if(!realtime)cvWaitKey(0);

}

}
}
else printf ("Camera Fai\nl");
}

```

Stereovision.h

```

#ifndef STEREOVISION_H
#define STEREOVISION_H

#include "cv.h"
#include "cxmisc.h"
#include "cvaux.h"
#include "highgui.h"
#include <vector>

```

```

class StereoVision
{
private:

```

```

//chesboard board corners X,Y, N = X*Y , number of corners = (number of cells - 1)
int cornersX,cornerRadius,cornerN;

CvSize image_Size;
int imageWidth;
int imageHeight;

std::vector<CvPoint2D32f> tempPoints[2];
std::vector<CvPoint3D32f> objectPoints;
std::vector<CvPoint2D32f> points[2];
std::vector<int> npoints;

int sample_Count;
bool calibrationInited;
bool calibrationDone;

public:
StereoVision(int imageWidth,int imageHeight);
~StereoVision();

//matrices resulting from calibration (used for cvRemap to rectify images)
CvMat *matrix_x1;
CvMat *matrix_y1;
CvMat *matrix_x2;
CvMat *matrix_y2;

CvMat *images_Rectified[2];
CvMat *image_Depth;
CvMat *image_DepthNormalized;

void calibrationInit(int cornersX,int cornersY);
int calibrate(IplImage* imageLeft,IplImage* imageRight);
int calibrationEnd();

int calibrationSave(const char* filename);
int calibrationLoad(const char* filename);

int stereoProcess(CvArr* imageSrcLeft,CvArr* imageSrcRight);

CvSize getimage_Size(){return image_Size;}

```

```

    bool getCalibrationInited(){return calibrationInited;};
    bool getCalibrationDone(){return calibrationDone;};
    int getsample_Count(){return sample_Count;};
};

#endif // STEREOVISION_H

Stereocamera.cpp

#include "stereocamera.h"

StereoCamera::StereoCamera()
{
    for(int lr=0;lr<2;lr++){
        captures[lr] = 0;
        frames[lr] = 0;
        framesGray[lr] = 0;
    }
    ready = false;
}

StereoCamera::~StereoCamera()
{
    for(int lr=0;lr<2;lr++){
        cvReleaseImage(&frames[lr]);
        cvReleaseImage(&framesGray[lr]);
        cvReleaseCapture(&captures[lr]);
    }
}

int StereoCamera::setup(CvSize imageSize){
    this->imageSize = imageSize;
    //CV_CAP_DSHOW for directX
    captures[0] = cvCaptureFromCAM( 0);
    captures[1] = cvCaptureFromCAM( 1);

    if( captures[0] && captures[1]){

        for(int i=0;i<2;i++){

,CV_CAP_PROP_FRAME_WIDTH,imageSize.width);
cvSetCaptureProperty(captures[i]

,CV_CAP_PROP_FRAME_HEIGHT,imageSize.height);
cvSetCaptureProperty(captures[i]

        }
}

```

```

        ready = true;
        return 0;
    }else{
        ready = false;
        return 1;
    }
}

int StereoCamera::capture(){
    frames[0] = cvQueryFrame(captures[0]);
    frames[1] = cvQueryFrame(captures[1]);
    if (captures[0] && captures[1]) return 0;
    else return 1;
}

IplImage* StereoCamera::getFramesGray(int lr){
    if(!frames[lr]) return 0;
    if(frames[lr]->depth == 1){
        framesGray[lr] = frames[lr];
        return frames[lr];
    }else{
        if(0 == framesGray[lr]) framesGray[lr] =
cvCreateImage(cvGetSize(frames[lr]),IPL_DEPTH_8U,1);
        cvCvtColor(frames[lr],framesGray[lr],CV_BGR2GRAY);
        return framesGray[lr];
    }
}

```

Stereocamera.h

```

#ifndef STEREOCAMERA_H
#define STEREOCAMERA_H

```

```

#include "cv.h"
#include "cxmisc.h"
#include "cvaux.h"
#include "highgui.h"
#include <vector>

```

```

class StereoCamera
{
    CvCapture* captures[2];

```

```
CvSize imageSize;

public:

    IplImage* frames[2];
    IplImage* framesGray[2];

    StereoCamera();
    ~StereoCamera();
    int setup(CvSize imageSize);
    bool ready;
    int capture();
    IplImage* getFramesGray(int lr);

};

#endif // STEREOCAMERA_H
```