View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by UPCommons. Portal del coneixement obert de la UPC

Rectification
and
Intermediate View synthesis

Universitat Politecnica
de Catalunya
EUETIT
Author: ***Dominik Tadeusz Piórkowski***
Tutor: ***Josep Ramon Morros i Rubió***
June 2008

Abstract

In this project c++ code supporting intermediate view synthesis was developed . The idea was to create classes and functions which can be later easily used to create intermediate views. Main part of the code is responsible for rectification. Images from two cameras may be rectified and then further operations with the images can be done. In this case the next operation on the rectified images is intermediate view synthesis. Special function computes from two rectified images the virtual view. The virtual image can be computed for any place set between two cameras taking the real image.

1

Table of contents

Abstract

1. Introduction
2. State of the art
3. Background

   3.1 Camera model

   3.2 Epipolar geometry

   3.3 Rectification

   3.4 Intermediate view synthesis
4. Background

   4.1 Rectification

   4.2 Intermediate view synthesis
5. Algorithm implementation

   5.1 Working platform

   5.2 Useful implementations

   5.3 Rectification implementation

   5.4 Intermediate view implementation
6. Test
7. Conclusions
8. Future Work
9. Bibliography

---

1 Introduction

The point of doing the intermediate view synthesis was to use it in three dimensional television (3D-TV). 3D–TV becomes last time more popular, and there are many researches and projects about it. Also many new devices for 3D vision becomes available.

For three-dimensional television (3D-TV) to become feasible and acceptable on a wide scale, the added realism must outweigh any required increases in processing and system complexity, and the stereoscopic information must be comfortable to view. Both of these goals can be achieved if intermediate views of the scene are available.

While binocular systems provide depth information through the sensation of stereopsis, a system consisting of only two views of a scene lacks the important depth cue of motion parallax, which provides the distinction between binocular and three-dimensional systems. Motion parallax can be synthesized from multiple intermediate views of the scene by presenting the correct stereo-pair according to the observer's position. Discomfort is often experienced when viewing stereoscopic images on two-dimensional displays. As with any subjective assessment, this discomfort is viewer-dependent. Viewers prefer varying degrees of depth perception from binocular imagery based on individual stereoscopic viewing ability and the range of depth present in the scene. A greater sense of depth is provided by a relatively large inter-camera separation, but the larger the separation the more difficulty marginal viewers have in fusing the images. If intermediate views of a scene are available, a viewer can dynamically select the inter-camera separation for comfort and preferred sense of depth.

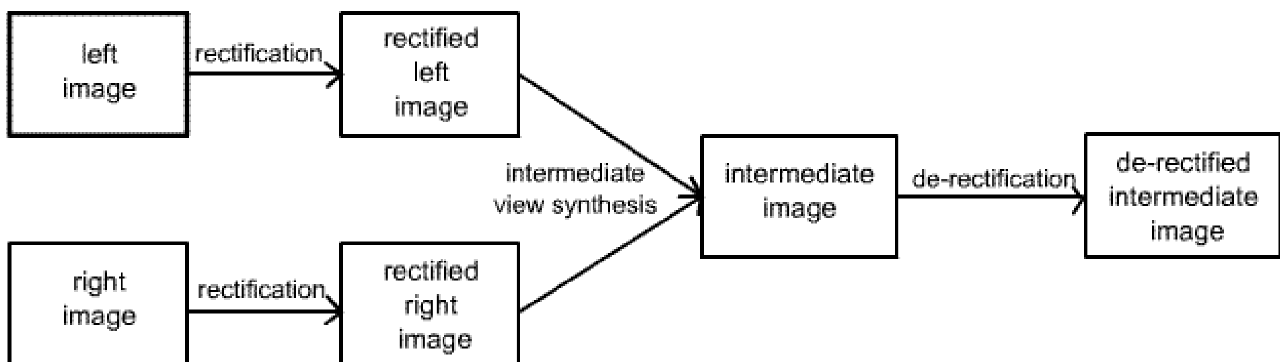
An intermediate view is defined as the image that would be obtained from a camera located between and on a straight line connecting the given stereo-pair's cameras. Computing the intermediate view from views from the real cameras, is much easier if the images are rectified. So first, before computing intermediate view, the images from the cameras have to be rectified.

Rectification is a process used to facilitate the analysis of a stereo pair of images by making it simple to enforce the two view geometric constraint. For a pair of related views the *epipolar geometry* provides a complete description of relative camera geometry. Once the epipolar geometry has been determined it is possible to constrain the match for a point in one image to lie on a line (the *epipolar line*) in the other image and vice versa. The process of rectification makes it very simple to impose this constraint by making all matching epipolar lines coincident and parallel with an image axis. Many stereo algorithms assume this simplified form because subsequent processing becomes much easier if differences between matched points will be in one direction only.

In the past, stereo images were primarily rectified using optical techniques, but more recently these have been replaced by software techniques. These model the geometry of optical projection by applying a single linear transformation to each image, effectively rotating both cameras until the image planes are the same. Such techniques are often referred to as planar rectification. The advantages of this linear approach are that it is mathematically simple, fast and preserves image features such as straight lines.

Having rectified images and corresponding them disparity maps computing of the intermediate view may be done. After this, to restore the right image geometry the de-rectification should be done. This operation is the inversion of rectification.

So after all the way of synthesis the intermediate view can be presented by such diagram:



1.1 Intermediate view synthesis

The figure below shows views at successive steps of the intermediate view synthesis.

a)

b)

c)

d)

1.2 a) pictures from left and right cameras b) the same pictures but rectified c) synthesized intermediate image d) de-rectified intermediate image

State of the art

There are few techniques for intermediate view synthesis. Some of them reconstruct whole 3D structure of the scene with all geometry information. For example, Seitz and Dyer proposed a method with affine model for computing 3D scene structure from point and line features in monocular image sequences. Another technique presented by Zhang is to recover Euclidean structure for facial images synthesis using some domain knowledge like distances and angles. Contrasting with the first method, the second one is discrete and generally more difficult to implement because the whole process does not require any 3D-scene information. These techniques may not be always the best choice, especially when the structure of the scene is very complicated.

The other solutions are for example centric mosaics and light field. They does not require any information about the 3D structure but analyse a certain number of views of a scene. These approaches yield very photo realistic results but typically require a very large number of reference images.

Method used in this project is something between those two described above. It is technique for image synthesis in perspective space. This means that with two reference images snapped by a calibrated camera a third view in-between two original ones can be generated. Pixels are transferred from real, rectificated input images to virtual image using a pre-computed disparity map. During the process, there are some information about the scene geometry needed (like the disparity) but not the whole 3D geometry information of the scene. These approaches assume the advantages of the first two categories: photo realistic results, low space requirements and time complexity independent from the scene complexity. Unfortunately, the disparity map is required to be dense and very accurate, otherwise no reasonable result can be obtained.

Rectification is a classical problem of stereo vision, however, few methods are available in the computer vision literature. Ayache and Lstman introduced a rectification

algorithm, in which a matrix satisfying a number of constraints is hand crafted. The distinction between necessary and arbitrary constraints is unclear. Some authors report rectification under restrictive assumptions, for instance assume a very restrictive geometry (parallel vertical axes of the camera references frames). Other works introduce algorithms which perform rectification given a weakly calibrated stereo rig, i.e. a rig for which only points correspondences between images are given. Some of works also concentrates on the issue of minimizing the rectified image distortion.

In this work it is assumed that stereo rig is calibrated, the cameras internal parameters, mutual position and orientation are known. Algorithm presented in this work rectify a calibrated stereo rig of unconstrained geometry and mounting general cameras.

3 Background

3.1 Camera model

Briefly description of camera model. Camera parameters and the way how 3D point is projected into 2D point will be described.

The camera is modelled by its **optical centre *c*** and its **retinal plane** (or image plane) ***R***. In each camera, a 3D point **w** = $(x, y, z)^T$ in world coordinates (where the world coordinate frame is fixed arbitrarily) is projected into an image point **m** = $(u,v)^T$ in camera coordinates, where ***m*** is the intersection of ***R*** with the line containing **w** and **c**. In projective (or homogeneous) coordinates, the transformation from **w** to **m** is modelled by the linear transformation $\bar{P}$

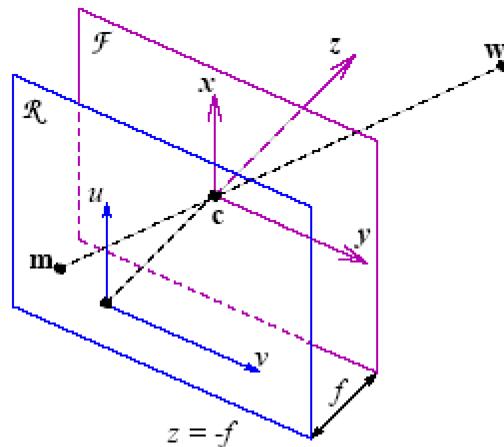
(3.1) $$\bar{m} = \bar{P}\,\bar{w}$$

where

(3.2) $$\tilde{\mathbf{m}} = \begin{pmatrix} U \\ V \\ S \end{pmatrix} \quad \tilde{\mathbf{w}} = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

(3.3) $$\mathbf{m} = \begin{pmatrix} U/S \\ V/S \end{pmatrix} \quad (\text{if } S \neq 0).$$

The points **w** for which S = 0 define the **focal plane** and are projected to infinity.



3.1 Pinhole camera model

Each pinhole camera is therefore modelled by its perspective projection matrix (PPM) $\tilde{\boldsymbol{P}}$, which can be decomposed into the product

(3.4) $$\tilde{\mathbf{P}} = \mathbf{A}(\mathbf{I} \mid \mathbf{0})\mathbf{G}$$

The matrix **A** gathers the intrinsic parameters of the camera, and has the following form:

(3.5) $$\mathbf{A} = \begin{pmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{pmatrix}.$$

where α_u α_v are the focal lengths in vertical and horizontal pixels, respectively, and (u_0, v_0) are the coordinates of the principal point. The matrix **G** is composed by a 3 x 3 rotation matrix and a vector **t**, encoding the camera position and orientation (extrinsic parameters) in the world reference frame, respectively:

(3.6) $$\mathbf{G} = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix}.$$

Let us write the PPM as

(3.7)

$$\tilde{\mathbf{P}} = \begin{pmatrix} \mathbf{q}_1^\top & q_{14} \\ \mathbf{q}_2^\top & q_{24} \\ \mathbf{q}_3^\top & q_{34} \end{pmatrix} = (\mathbf{P}|\tilde{\mathbf{p}}).$$

The plane $\mathbf{q}_3^T \mathbf{w} + q_{34} = 0$ (S = 0) is the **focal plane**, and the two planes $\mathbf{q}_1^T \mathbf{w} + q1_4 = 0$ and $\mathbf{q}_2^T \mathbf{w} + q2_4 = 0$ intersect the retinal plane in the vertical (U = 0) and horizontal (V = 0) axis of the retinal coordinates, respectively.

The optical centre **c** is the intersection of the three planes introduced in the previous paragraph, therefore

(3.8)

$$\tilde{\mathbf{P}} \begin{pmatrix} \mathbf{c} \\ 1 \end{pmatrix} = \mathbf{0}$$

and

(3.9)

$$\mathbf{c} = -\mathbf{P}^{-1}\tilde{\mathbf{p}}.$$

The optical ray associated to an image point **m** is the line **cm**, i.e. The set of points {**w :** $\bar{m} = \bar{P} \quad \bar{w}$ }. The equation of this ray can be written in parametric form as

(3.10)

$$\mathbf{w} = \mathbf{c} + \lambda \mathbf{P}^{-1}\tilde{\mathbf{m}}.$$

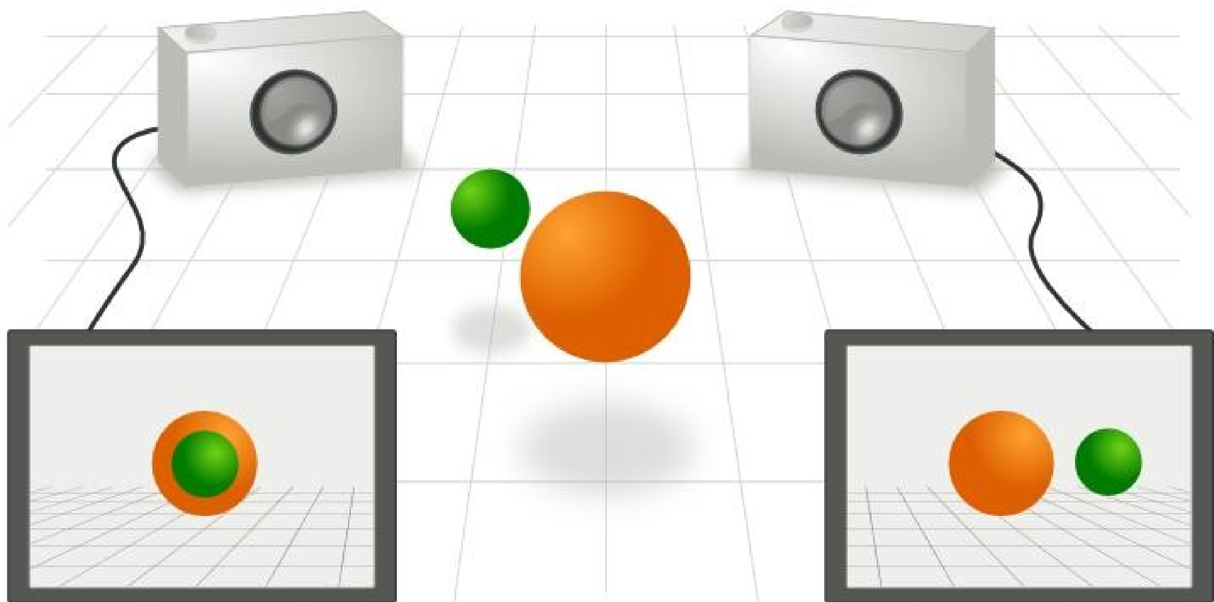
with λ an arbitrary real number.[1]

3.2 Epipolar geometry

This chapter gives a overview of epipolar geometry. Knowledge of the epipolar geometry is needed to understand the rectification.

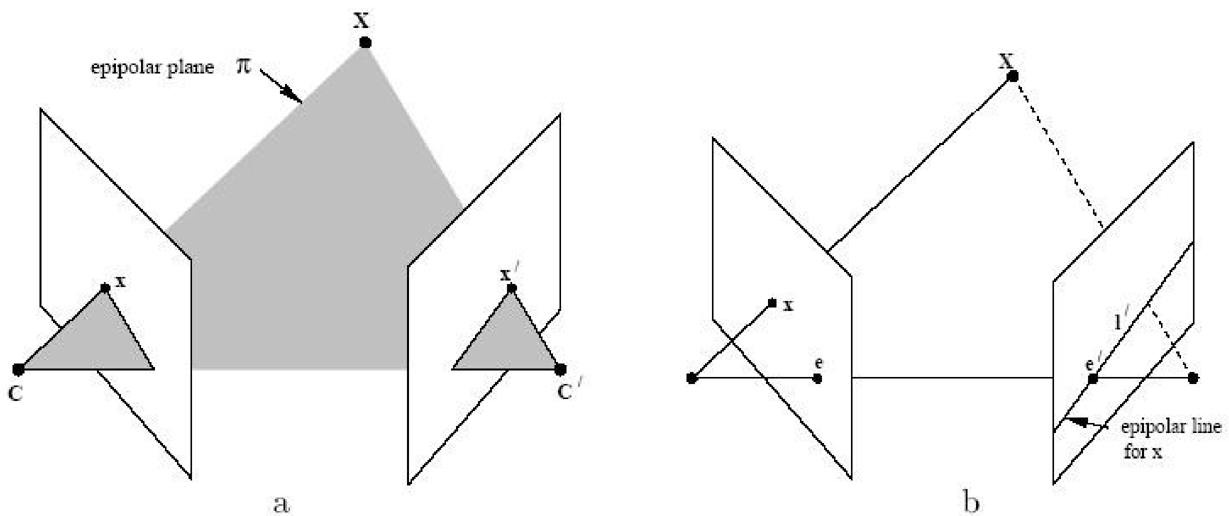
Epipolar geometry refers to the geometry of stereo vision. Each camera captures a 2D image of the 3D world. When two cameras view a 3D scene from two distinct positions, there are a number of geometric relations between the 3D points and their projections onto the 2D images that lead to constraints between the image points. This conversion from 3D to 2D is referred to as a perspective projection and is described by the pinhole camera model.

The figure below shows two cameras taking picture of the same scene from different points of view. The epipolar geometry then describes the relation between the two resulting views.[2]



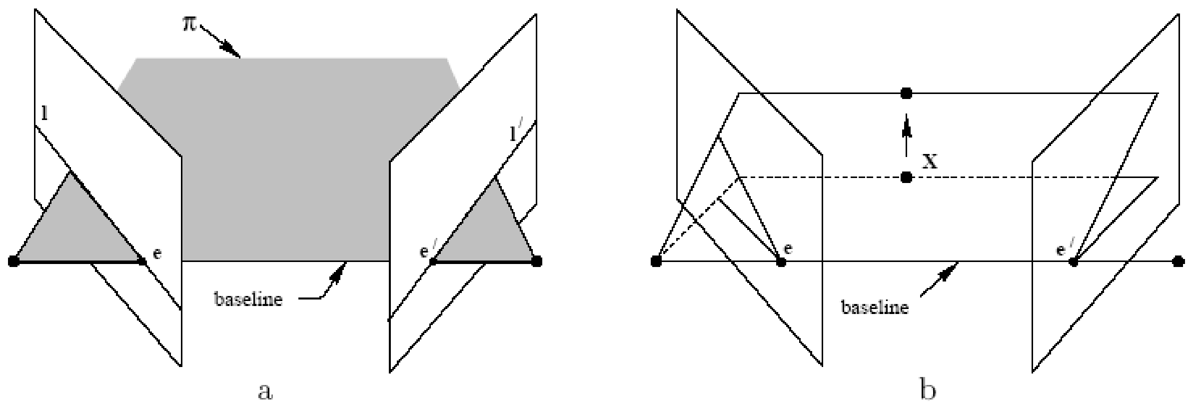
3.2 Two cameras taking picture of the same scene from different points of view.

The epipolar geometry between two views is essentially the geometry of the intersection of the image planes with the pencil of planes having the baseline as axis (the baseline is the line joining the camera centres). This geometry is usually motivated by considering the search for corresponding points in stereo matching. Suppose a point **X** in 3-space is imaged in two views, at **x** in the first, and **x′** in the second. As shown in figure 3.3a the image points **x** and **x′**, space point **X**, and camera centres are coplanar. Denote this plane as ***π***. Clearly, the rays back-projected from **x** and **x'** intersect at **X**, and the rays are coplanar, lying in ***π***. It is this latter property that is of most significance in searching for a correspondence.



3.3 Point correspondence geometry.

Supposing now that x is known, the question is: how the corresponding point **x'** is constrained. The plane is determined by the baseline and the ray defined by **x**. From above we know that the ray corresponding to the (unknown) point **x′** lies in ***π*** , hence the point **x'** lies on the line of intersection **l'** of ***π*** with the second image plane (pic. 3.3b). This line **l'** is the image in the second view of the ray back-projected from **x**. In terms of a stereo correspondence algorithm the benefit is that the search for the point corresponding to **x** need not cover the entire image plane but can be restricted to the line **l′**.

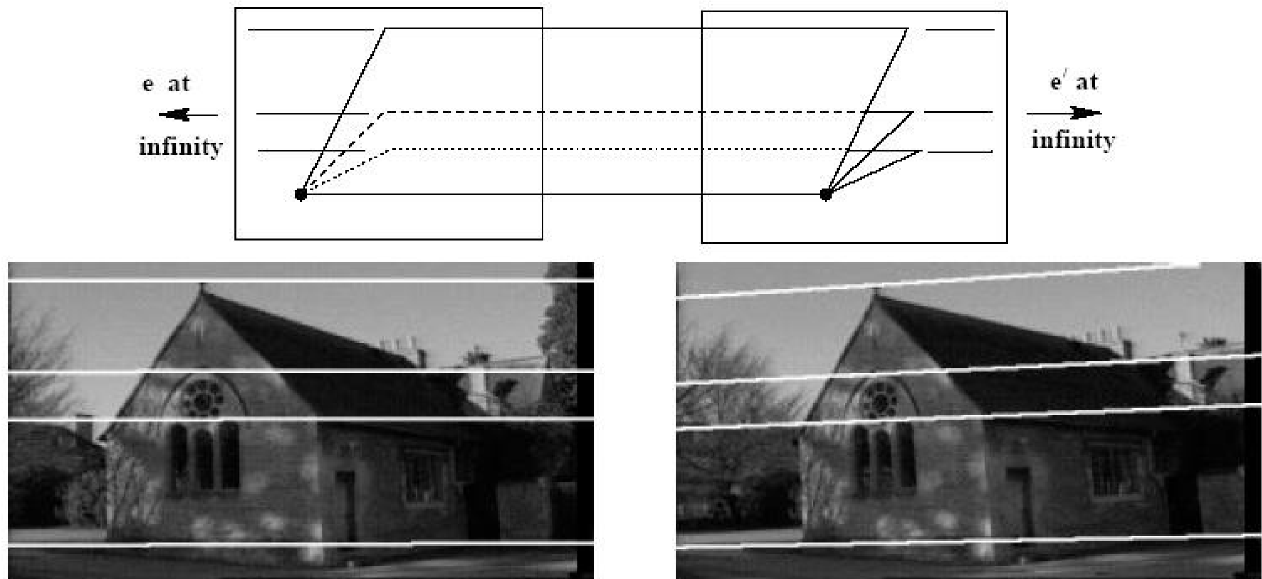


3.4 Epipolar geometry. a) The camera baseline intersects each image plane at the epipoles **e** *and* **e'**. *Any plane containing the baseline is an epipolar plane, and intersects the image planes in corresponding epipolar lines* **l** *and* **l'**. *b) As the position of the 3D point* **X** *varies, the epipolar planes „rotate" about the baseline. This family of planes is known as an epipolar pencil. All epipolar lines intersect at the epipole.*

The geometric entities involved in epipolar geometry are illustrated in figure 3.4. The terminology is:

- The **epipole** is the *point* of intersection of the line joining the camera centres (the baseline) with the image plane. Equivalently, the epipole is the image in one view of the camera centre of the other view. It is also the vanishing point of the baseline (translation) direction.
- An **epipolar plane** is a plane containing the baseline. There is a one-parameter family (a pencil) of epipolar planes.
- An **epipolar line** is the intersection of an epipolar plane with the image plane. All epipolar lines intersect at the epipole. An epipolar plane intersects the left and right image planes in epipolar lines, and defines the correspondence between the lines.[5]

Example of epipolar geometry is given in figure 3.5.

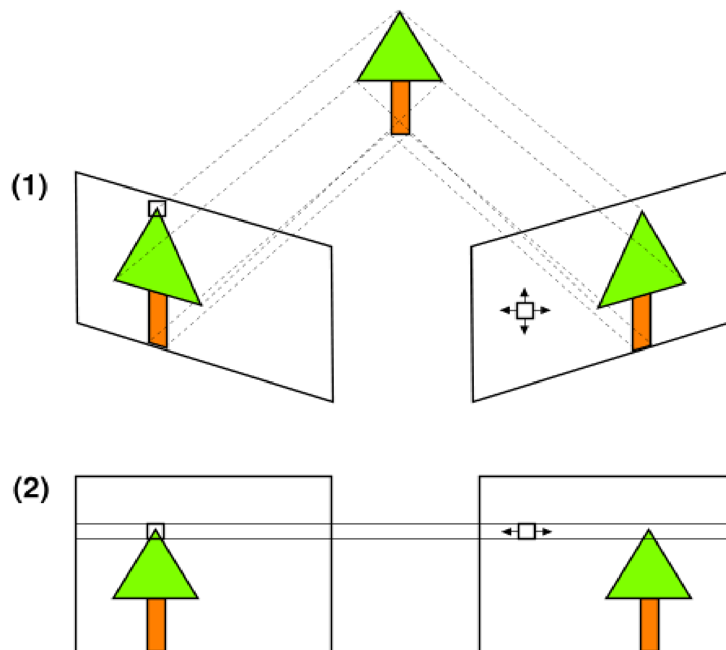


3.5 Example of epipolar geometry.

3.3 Rectification

Between two cameras there is a problem of finding a corresponding point viewed by one camera in the image of the other camera (this is called the correspondence problem.) In most camera configurations, finding correspondences requires a search in two dimensions. However, if the two cameras are aligned to have a common image plane, the search is simplified to one dimension - a line that is parallel to the line between the cameras (the baseline). Image rectification is an equivalent (and more often used) alternative to this precise camera alignment. It transforms the images to make the epipolar lines (epipolar geometry) align horizontally.

If the images to be rectified are taken from camera pairs without geometric distortion, this calculation can easily be made with a linear transformation. X & Y rotation puts the images on the same plane, scaling makes the image frames be the same size and Z rotation and skew adjustments make the image pixel rows directly line up. The rigid alignment of the cameras needs to be known (by calibration) and the calibration coefficients are used by the transform.[2]



3.6 The search space: 1) before rectification 2) after rectification

3.4 Intermediate view synthesis

Despite significant advances in 3D computer graphics, the realism of rendered images is limited by hand coded graphical models. Existing techniques for creating 3D models are time intensive and put high demands on the artistry of the model. In light of these limitations, there has been growing interest in the use of 2D image warping techniques for image synthesis and animation. The advantage of working in 2D is that photographs of real scenes can be used as a basis to create very realistic effects.

In this technique the intermediate view is computed from images taken by cameras from two different positions. The images have to be rectified, and their disparity maps need to be known.

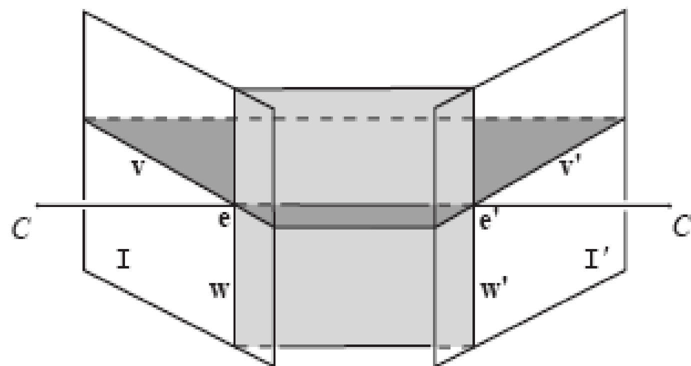
3.7 I_1 - the left imageI_2 - the right image$I_{1.5}$ – generated intermediate view image

4 Detailed description of algorithms

4.1 Rectification

In this chapter will be described algorithm used for image rectification. Mostly this are matrix operations, and it is closely connected with the epipolar geometry.

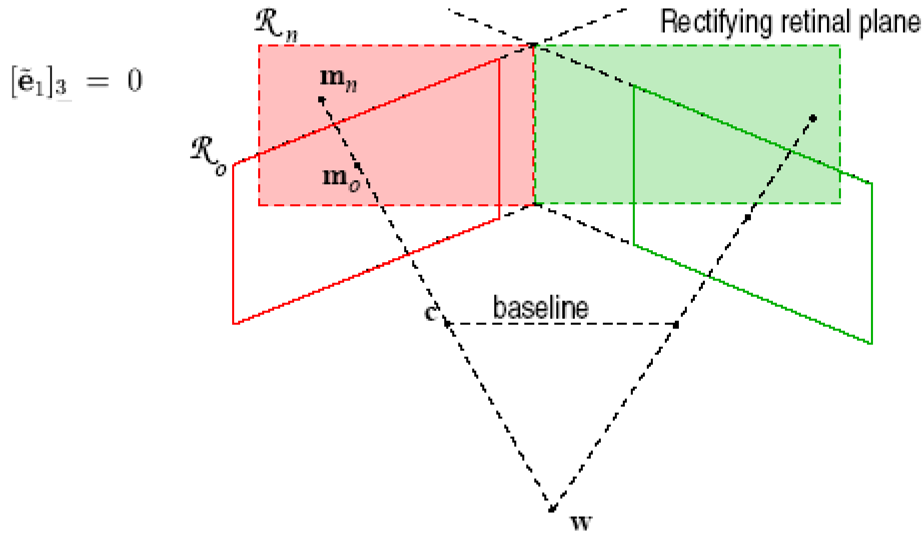
Image rectification can be view as the process of transforming the epipolar geometry of a pair of images into a canonical form. This is accomplished by applying a homography to each image that maps the epipole to a predetermined point.



4.1 The lines v and v', and w and w' must be corresponding epipolar lines that lie on common epipolar planes.

We assume that the stereo rig is calibrated, i.e., the perspective projection matrices $\bar{P}_{o1}$ and $\bar{P}_{o2}$ are known. This assumption is not strictly necessary, but leads to a simpler technique. The idea behind rectification is to define two new perspective projection matrices $\bar{P}_{n1}$ and $\bar{P}_{n2}$ which preserve the optical centres but with image planes parallel to the baseline. In addition we want the epipolar line of the point $m_1 = (u_1,v_1)^T$ in the right image to be the horizontal line $v_2 = v_1$ in the left image.

Consider figure 4.2, where the old retinal plane $\boldsymbol{R}_o$ and the new one $\boldsymbol{R}_n$ are depicted: image rectification is the operation of transforming from $\boldsymbol{R}_o$ to $\boldsymbol{R}_n$.



4.2 Rectification

Constraining the rectifying perspective projection matrices

The new perspective projection matrices $\bar{P}_{n1}$ and $\bar{P}_{n2}$ will be computed as the solution of a system of equations, which express the constraints arising from rectification requirements plus others constraints necessary to ensure a unique solution.

In the following, I will denote with $\bar{P}_{o1}$ $\bar{P}_{o2}$ the old perspective projection matrices. Let

(4.1)

$$\tilde{\mathbf{P}}_{n1} = \left(\begin{array}{c|c} \mathbf{a}_1^\top & a_{14} \\ \mathbf{a}_2^\top & a_{24} \\ \mathbf{a}_3^\top & a_{34} \end{array} \right) \qquad \tilde{\mathbf{P}}_{n2} = \left(\begin{array}{c|c} \mathbf{b}_1^\top & b_{14} \\ \mathbf{b}_2^\top & b_{24} \\ \mathbf{b}_3^\top & b_{34} \end{array} \right)$$

be the sought rectifying perspective projection matrices.

Common focal plane

If cameras share the same focal plane the common retinal plane is constrained to be parallel to the baseline and epipolar lines are parallel. The two rectifying perspective projection matrices have the same focal plane if

(4.2) $a_3 = b_3 \qquad a_{34} = b_{34}$

Position of the optical centres

The optical centres of the rectifying perspective projections must be the same as those of the original projections:

(4.3)
$$\tilde{\mathbf{P}}_{n1}\begin{pmatrix}\mathbf{c}_1\\1\end{pmatrix} = \mathbf{0} \qquad \tilde{\mathbf{P}}_{n2}\begin{pmatrix}\mathbf{c}_2\\1\end{pmatrix} = \mathbf{0}.$$

where optical centres $\mathbf{c}_1$ $\mathbf{c}_2$ are given:

(4.4)
$$\mathbf{c}_1 = -\mathbf{P}_{o1}^{-1}\tilde{\mathbf{p}}_{o1} \qquad \mathbf{c}_2 = -\mathbf{P}_{o2}^{-1}\tilde{\mathbf{p}}_{o2}$$

Equation (4.3) gives six linear constraints:

(4.5)
$$\begin{cases} \mathbf{a}_1^\top \mathbf{c}_1 + a_{14} = 0 \\ \mathbf{a}_2^\top \mathbf{c}_1 + a_{24} = 0 \\ \mathbf{a}_3^\top \mathbf{c}_1 + a_{34} = 0 \\ \mathbf{b}_1^\top \mathbf{c}_2 + b_{14} = 0 \\ \mathbf{b}_2^\top \mathbf{c}_2 + b_{24} = 0 \\ \mathbf{b}_3^\top \mathbf{c}_2 + b_{34} = 0. \end{cases}$$

Alignment of conjugate epipolar lines

The vertical coordinate of the projection of a 3D point onto the rectifying retinal plane must be the same in both image, i.e.:

(4.6)
$$\frac{\mathbf{a}_2^\top \mathbf{w} + a_{24}}{\mathbf{a}_3^\top \mathbf{w} + a_{34}} = \frac{\mathbf{b}_2^\top \mathbf{w} + b_{24}}{\mathbf{b}_3^\top \mathbf{w} + b_{34}}.$$

Using equation (2) the constraints can be obtained:[20]

(4.7) $a_2 = b_2 \qquad a_{24} = b_{24}$

The equations written to this point are sufficient to guarantee rectification: to prove this, let verify that epipolar lines are parallel and horizontal.
When (the epipole is at infinity) the epipolar lines are parallel to the vector

$([\tilde{\mathbf{e}}_1]_1 [\tilde{\mathbf{e}}_1]_2)^\top$.

As we know, each epipole is the projection of the conjugate optical centre, i.e.

(4.8)

$$\begin{aligned} \tilde{\mathbf{e}}_1 &= \tilde{\mathbf{P}}_{n1} \begin{pmatrix} \mathbf{c}_2 \\ 1 \end{pmatrix} = \begin{pmatrix} \mathbf{a}_1^\top \mathbf{c}_2 + a_{14} \\ \mathbf{a}_2^\top \mathbf{c}_2 + a_{24} \\ \mathbf{a}_3^\top \mathbf{c}_2 + a_{34} \end{pmatrix} \\ \tilde{\mathbf{e}}_2 &= \tilde{\mathbf{P}}_{n2} \begin{pmatrix} \mathbf{c}_1 \\ 1 \end{pmatrix} = \begin{pmatrix} \mathbf{b}_1^\top \mathbf{c}_1 + b_{14} \\ \mathbf{b}_2^\top \mathbf{c}_1 + b_{24} \\ \mathbf{b}_3^\top \mathbf{c}_1 + b_{34} \end{pmatrix} \end{aligned}$$

Hence epipolar lines are horizontal when

(4.9)

$$\begin{cases} \mathbf{a}_1^\top \mathbf{c}_2 + a_{14} \neq 0 \\ \mathbf{a}_2^\top \mathbf{c}_2 + a_{24} = 0 \\ \mathbf{a}_3^\top \mathbf{c}_2 + a_{34} = 0 \\ \mathbf{b}_1^\top \mathbf{c}_1 + b_{14} \neq 0 \\ \mathbf{b}_2^\top \mathbf{c}_1 + b_{24} = 0 \\ \mathbf{b}_3^\top \mathbf{c}_1 + b_{34} = 0 \end{cases}$$

The four equations are satisfied as long as equations (4.2) (4.3) and (4.7) hold, as one can easily verify.

Although rectification is guaranteed, the orientation of the retinal plane has still one degree of freedom. Moreover, the constraints written up to now are not enough to obtain a unique perspective projection matrix. We shall therefore choose explicitly the intrinsic parameters to obtain enough equations.

20

Orientation of the rectifying retinal plane

Rectifying focal planes are chosen to be parallel to the intersection of the two original focal planes, i.e.

(4.10)
$$\mathbf{a}_3^\top(\mathbf{f}_1 \wedge \mathbf{f}_2) = 0$$

where f_1 and f_2 are the third rows of $\bar{P}_{o1}$ $\bar{P}_{o2}$ respectively. The corresponding equation, $\mathbf{b}_3^\top(\mathbf{f}_1 \wedge \mathbf{f}_2) = 0$, is redundant thanks to equation (2).

Orthogonality of the rectifying reference frames

The intersection of the retinal plane with the planes $\mathbf{a}_1^T\mathbf{w} + a_{14} = 0$ and $\mathbf{a}_2^T\mathbf{w} + a_{24} = 0$ correspond to the v and u axes, respectively, of the retinal reference frame. In order for this reference frame to be orthogonal, the planes must be perpendicular to each other, hence, taking into account equation (7)

(4.11)
$$\mathbf{a}_1^\top\mathbf{a}_2 = 0 \qquad \mathbf{b}_1^\top\mathbf{a}_2 = 0.$$

Principal points

The principal point (u_0,v_0) is given by

(4.12)
$$u_0 = \mathbf{a}_1^\top\mathbf{a}_3 \qquad v_0 = \mathbf{a}_2^\top\mathbf{a}_3.$$

The two principal points are set to (0,0) and using equations (2) and (7) constraints are obtained

(4.13)
$$\begin{cases} \mathbf{a}_1^\top\mathbf{a}_3 = 0 \\ \mathbf{a}_2^\top\mathbf{a}_3 = 0 \\ \mathbf{b}_1^\top\mathbf{a}_3 = 0. \end{cases}$$

Focal lengths in pixels

The horizontal and vertical focal lengths in pixels, respectively, are given by

(4.14)
$$\alpha_u = ||\mathbf{a}_1 \wedge \mathbf{a}_3|| \qquad \alpha_v = ||\mathbf{a}_2 \wedge \mathbf{a}_3||.$$

$$\tilde{\mathbf{P}}_o = (\mathbf{P}_o|\tilde{\mathbf{p}}_o)$$

By setting arbitrary the values of α_u and α_v we obtain the constraints

(4.15)
$$\begin{cases} ||\mathbf{a}_1 \wedge \mathbf{a}_3||^2 = \alpha_u^2 \\ ||\mathbf{a}_2 \wedge \mathbf{a}_3||^2 = \alpha_v^2 \\ ||\mathbf{b}_1 \wedge \mathbf{a}_3||^2 = \alpha_u^2 \end{cases}$$

which, by virtue of the equivalence

(4.16)
$$||\mathbf{x} \wedge \mathbf{y}||^2 = ||\mathbf{x}||^2||\mathbf{y}||^2 - (\mathbf{x}^T\mathbf{y})^2$$

and equation (13), can be rewritten as

(4.17)
$$\begin{cases} ||\mathbf{a}_1||^2||\mathbf{a}_3||^2 = \alpha_u^2 \\ ||\mathbf{a}_2||^2||\mathbf{a}_3||^2 = \alpha_v^2 \\ ||\mathbf{b}_1||^2||\mathbf{a}_3||^2 = \alpha_u^2 \end{cases}$$

Set the scale factor

Perspective projection matrices are defined up to a scale factor, and a common choice to block the latter is to set

(4.18)
$$||\mathbf{a}_3|| = 1 \qquad ||\mathbf{b}_3|| = 1.$$

Solving for the rectifying perspective projection matrix

Let organise the constraints introduced in the previous section in the following four systems:

$$\begin{cases} \mathbf{a}_3^\top \mathbf{c}_1 + a_{34} = 0 \\ \mathbf{a}_3^\top \mathbf{c}_2 + a_{34} = 0 \\ \mathbf{a}_3^\top (\mathbf{f}_1 \wedge \mathbf{f}_2) = 0 \\ ||\mathbf{a}_3|| = 1 \end{cases}$$

(4.19)

$$\begin{cases} \mathbf{a}_2^\top \mathbf{c}_1 + a_{24} = 0 \\ \mathbf{a}_2^\top \mathbf{c}_2 + a_{24} = 0 \\ \mathbf{a}_2^\top \mathbf{a}_3 = 0 \\ ||\mathbf{a}_2|| = \alpha_v \end{cases}$$

(4.20)

$$\begin{cases} \mathbf{a}_1^\top \mathbf{c}_1 + a_{14} = 0 \\ \mathbf{a}_1^\top \mathbf{a}_2 = 0 \\ \mathbf{a}_1^\top \mathbf{a}_3 = 0 \\ ||\mathbf{a}_1|| = \alpha_u \end{cases}$$

(4.21)

$$\begin{cases} \mathbf{b}_1^\top \mathbf{c}_2 + b_{14} = 0 \\ \mathbf{b}_1^\top \mathbf{a}_2 = 0 \\ \mathbf{b}_1^\top \mathbf{a}_3 = 0 \\ ||\mathbf{b}_1|| = \alpha_u \end{cases}$$

(4.22)

Plus

(4.23)

$$\begin{cases} \mathbf{a}_2 = \mathbf{b}_2 \\ a_{24} = b_{24} \\ \mathbf{a}_3 = \mathbf{b}_3 \\ a_{34} = b_{34} \end{cases}$$

The first four systems have all the same structure, each one being a 3 x 4 linear homogeneous system subject to a quadratic constraint, that is

(4.24)

$$\begin{cases} \mathbf{A}\mathbf{x} = 0 \\ ||\mathbf{x}'|| = k \end{cases}$$

where **x'** is a vector composed by the first three components of **x**, and k is a real number. The four systems above are solved in sequence, top to bottom. The solution of each system is obtained by first computing (for example by SVD factorisation) a one parameter family of solutions to **Ax** = 0 of the form **x** = αx_0, where x_0 is a non trivial solution and λ is an arbitrary real number, and the letting $\alpha = k/||x_0'||$.

The rectifying transformation

Now that – for each camera – the perspective projection matrix is known, we want to compute the linear transformation (in projective coordinates) that maps the retinal plane of the old perspective projection matrix

onto the retinal plane of

$$\tilde{\mathbf{P}}_n = (\mathbf{P}_n|\tilde{\mathbf{p}}_n).$$

This transformation is the 3 x 3 matrix $T = P_n P_o^{-1}$.

For any 3D point **w**

(4.25)
$$\begin{cases} \tilde{\mathbf{m}}_o = \tilde{\mathbf{P}}_o \tilde{\mathbf{w}} \\ \tilde{\mathbf{m}}_n = \tilde{\mathbf{P}}_n \tilde{\mathbf{w}}. \end{cases}$$

From equation (3.10) is known that the equation of the optical ray associated to $\mathbf{m}_o$ is

(4.26)
$$\mathbf{w} = \mathbf{c}_o + \lambda \mathbf{P}_o^{-1} \tilde{\mathbf{m}}_o$$

hence

(4.27)
$$\begin{aligned} \tilde{\mathbf{m}}_n &= \tilde{\mathbf{P}}_{n1} \begin{pmatrix} \mathbf{c}_o + \lambda \mathbf{P_o}^{-1} \tilde{\mathbf{m}}_o \\ 1 \end{pmatrix} = \\ &= \tilde{\mathbf{P}}_{n1} \begin{pmatrix} \mathbf{c}_o \\ 1 \end{pmatrix} + \tilde{\mathbf{P}}_{n1} \begin{pmatrix} \lambda \mathbf{P_o}^{-1} \tilde{\mathbf{m}}_o \\ 0 \end{pmatrix} = \\ &= \tilde{\mathbf{P}}_{n1} \begin{pmatrix} \mathbf{c}_n \\ 1 \end{pmatrix} + \mathbf{P}_n \mathbf{P}_o^{-1} \tilde{\mathbf{m}}_o. \end{aligned}$$

Assuming that rectification does not alter optical centre ($\mathbf{c}_n = \mathbf{c}_o$):

(4.28)
$$\tilde{\mathbf{m}}_n = \mathbf{P}_n \mathbf{P}_o^{-1} \tilde{\mathbf{m}}_o.$$

The transformation **T** is then applied to the original image to produce a rectified image. The pixels (integer – coordinate positions) of the rectified image correspond, in general, to non – integer positions on the original image plane. Therefore, the grey levels of the rectified image are computed by bilinear interpolation.[1]

a)

b)

4.3 Rectification: a) pictures with epipolar lines from left and right camera before rectification, b) pictures with epipolar lines from left and right camera after rectification.

4.2 Intermediate view synthesis

26

The parallel, rectified stereo view V_{rectiL} and V_{rectiR} are known. Assume that the two disparity maps D_{LR} (based on the left view, the disparity value at **p** is noted as d_p^{LR}) and D_{RL} (based on the right view, the disparity value at **p** is noted as d_p^{RL}) are also known. The x and y coordinates of $\mathbf{p}_{rectiL}$ in the view V_{rectiL} are x_p^{rectiL} and y_p^{rectiL} respectively, and the same holds for $\mathbf{p}_{rectiR}$ in V_{rectiR} :

$$d_p^{LR} = x_p^{rectiR} - x_p^{rectiL} \quad d_p^{RL} = x_p^{rectiL} - x_p^{rectiR}$$
$$y_p^{rectiL} = y_p^{rectiR}$$

C_x (virtual camera), C_{rectiL} ,(left camera with rectified image on the output) and C_{rectiR} (right camera with rectified image on the output) are parallel to each other. The projections of **w** have the same y - coordinate but are only different in x – coordinate:

(4.29)
$$x_p^{rectiL} = -\frac{f_M}{s_{xM}} \cdot \frac{x_w^M - (-b/2)}{z_w^M} + x_{0M}$$

(4.30)
$$x_p^{rectiR} = -\frac{f_M}{s_{xM}} \cdot \frac{x_w^M - b/2}{z_w^M} + x_{0M}$$

(4.31)
$$x_p^X = -\frac{f_M}{s_{xM}} \cdot \frac{x_w^M - x_{cD}^M}{z_w^M} + x_{0M}$$

Subtracting (4.29) from (4.30), gives:

(4.32)
$$z_w^M = \frac{f_M b}{s_{xM}(x_p^{rectiR} - x_p^{rectiL})} = \frac{f_M b}{s_{xM} d_p^{LR}} = -\frac{f_M b}{s_{xM} d_p^{RL}}$$

Subtracting (4.30) from (4.31), gives:

26

(4.33)
$$x_p^X = \frac{f_M}{s_{xM} z_w^M}(x_{cD}^M - \frac{b}{2}) + x_p^{rectiR}$$

Taking (4.32) into (4.33) generates:

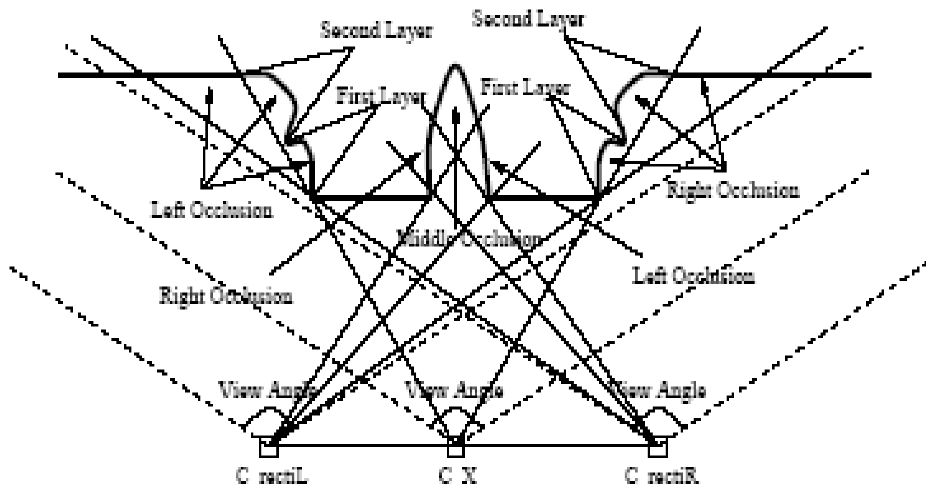
(4.34)
$$x_p^X = \underbrace{(x_p^{rectiL} + x_p^{rectiR})/2}_{\text{Middle View}} + \underbrace{(x_p^{rectiR} - x_p^{rectiL})}_{\text{Disparity}} \cdot \frac{x_{cD}^M}{b}$$

This means:

(4.35)
$$x_p^X = x_p^{rectiL} + d_p^{LR}/2 + d_p^{LR} \cdot \frac{x_{cD}^M}{b}$$

(4.36)
$$x_p^X = x_p^{rectiR} + d_p^{RL}/2 - d_p^{RL} \cdot \frac{x_{cD}^M}{b}$$

From equation (4.35) and (4.36), it seems that we need only one disparity map plus one view to construct the x - interpolated view perfectly. However, in practice, we always suffer from occlusions (see figure 4.4).



4.4 Views from the three parallel cameras. Bold curves indicate scene surface from the point view of the three cameras.

In general, four cases can be distinguished:

1) **Complete 3D info**: The info that can be viewed in C_{rectiL} , C_{rectiR} and C_X

2) **Left Occlusion**: The parts of the view that are visible in the left camera but not in the right camera.

3) **Right Occlusion**: The parts of the view that are visible in the right camera but not in the left camera.

4) **Middle Occlusion**: The parts of the view that are only visible in the x - interpolated view.

Assuming that the estimated disparity maps contain the correct (pseudo) disparity in occluded regions, we can then generate the x - interpolated view V_X by:

1) **For Complete 3D parts**: Either equation (4.35) or (4.36) is applicable.

2) **For Left Occlusion parts**: Equation (4.35) can be used.

3) **For Right Occlusion parts**: Equation (4.36) can be employed.

4) **For Middle Occlusion parts**: No info is available in either V_{rectiL} or V_{rectiR} . We have to approximate it e.g. by nearest-neighbour (zero order) or linear interpolation (first order).

Assuming that the light condition between the left and the right view does not change too much, for most parts of the x – interpolated view, we just fetch them from the left view. Only for right occlusion parts, we take them from the right view. To investigate the possibility of handling sparse disparity maps without upsampling, the following steps are adopted:

1. From equation (4.35), is constructed a disparity map D_{XL} which is based on the view V_X. The relations are:

$$d_p^{XL} = x_p^M - x_p^{rectiL} = d_p^{LR}/2$$

and

$$x_p^X = x_p^{rectiL} + d_p^{XL} + d_p^{XL} \cdot \frac{2x_{cD}^M}{b}$$

2. From equation (4.36), can be constructed a disparity map D_{XR} which is based on the view V_X. The relations are

$$d_p^{XR}a = x_p^{rectiR} - x_p^{M} = d_p^{RL}/2$$

and

$$x_p^{X} = x_p^{rectiR} - d_p^{XR} + d_p^{XR}/2 \quad \frac{2x_{cD}^{M}}{b}$$

3. $D_X L$ and $D_X R$ are compared to generate D_X:

for each pixel in D_X,

if there is assigned a value at the same position in $D_X L$,

then copy that value to the current position in D_X, and at the same time mark the current position as left available (in fact, this marking indicates both left occlusion and complete 3D info),

else if there is assigned a value at the same position in $D_X R$,

then copy that value to the current position in D_X, and at the same time mark the current position as right available

else mark the current position as middle occlusion

endif

endfor

4. In D_X, for each middle occlusion part, if two ends are both left available or right available, then filling in by linear interpolation, otherwise by nearest neighbour.

5. Generate the view V_X, by backward mapping based on D_X, V_{rectiL}, and V_{rectiR}. [3]

A example of left, right and generated intermediate views is shown in figure 4.5

a)

b)

c)

4.5 a) view from left camera b)generated view c) view from right camera

5 Algorithm implementation

5.1 Working platform

I was working on linux operating system, and the program, all functions, classes were tested on this [31]system. However there is big probability that it works without (or after some small changes) also under windows operating system.

Point of this work was to implement the rectification algorithm in the c++ language. C++ is very efficient, and autonomous what's makes it very mobile, that's why I suppose that this rectification may be very easy used on other operating systems.

Since big part of the algorithm (actually all of it) base on matrix operations, and the standard c++ libraries doesn't support them, I used the ImagePlus library. ImagePlus is the new development platform in C++ language for the Video and Image Processing Group at UPC. The Imageplus contain big part of the functions, mathematics operations that were indispensable for the algorithm.

Short characteristic of ImagePlus:

Basic classes

- The basic class for vectors, matrices, 3D arrays is class MultiArray
  - This means there are no Vector or Matrix class, use MultiArray<T,1> for vectors, MultiArray<T,2> for matrices, etc.

- A unique class named ImaVol provides the basic structure for both image and volume classes. This way, common functions can be defined to the ImaVol class and they will work for all other subclasses (through polymorphism)
  - All images and volumes are derived from the ImaVol class

31

- A MultiArray class is used to represent each colour channel on an ImaVol
- The ImaVol class is generic in terms of the data type so different image and volume types can be created
- The ImaVol class is generic in terms of the number of colour channels of the image/volume so that subclasses for different colour spaces can be derived
- The ImaVol class is generic in terms of the number of dimensions of these channels so it that subclasses for images or volumes can be derived

Common classes

- Filter class:
  Filter class provides a standard framework to implement all kind of filters that work over any MultiArray or ImaVol objects. The concept can be extended to other data types. Each different filter is defined as a new class derived from Filter. These derived classes inherit two public member functions from Filter: filter() and name() (see filter.hpp). The filtering process returns an object of the same type as the input.
- Room class:
  Room class is the basic container for voxelized volumes resulting from a 3D reconstruction in multi-camera scenarios. It contains a volume object and the parameters needed to relate the discretized volume with the real scenario. These parameters are the voxel size in cm. and the offset. The offset is a point in the scenario coordinate system, and is the coordinate origin of the reconstructed zone. The scenario coordinate system is set by the camera parameters. The voxelized zone may not start at the scenario origin, i.e a case where we reconstruct a small zone in the room. Then the offset is used to relate the volume coordinates with the scenario coordinates.

ImagePlus modules

- The ImagePlus library is divided into several modules based on functionality
- Each module is implemented in an unique namespace with the same name as the module and in a unique directory (see folder structure)
- The list of modules implemented can be checked under the namespaces list tab.[4]

5.1 Useful implementations

This chapter takes view at some classes and functions, which implementation was needed and which can be used in future.

However the ImagePlus gave me some portion of needed functions I still have to implement some by myself. The most expanded operations on matrices I implemented into the Imageplus as classes or functions for future use. Here they are:

- Singular value decomposition class
- Interpolation function
- Class for computing intrinsic and extrinsic parameters from perspective projection matrices

In next part I will shortly describe this implementations.

5.2.1 SVD class

This class do the singular value decomposition.

In linear algebra, the singular value decomposition (SVD) is an important factorization of a rectangular real or complex matrix.

Suppose M is an m-by-n matrix whose entries come from the field K, which is either the field of real numbers or the field of complex numbers. Then there exists a factorization of the form

$$M = U\Sigma V^*,$$

where U is an m-by-m unitary matrix over K, the matrix Σ is m-by-n with non-negative numbers on the diagonal (as defined for a rectangular matrix) and zeros off the diagonal, and V* denotes the conjugate transpose of V, an n-by-n unitary matrix over K. Such a factorization is called a singular-value decomposition of M.

- The matrix V thus contains a set of orthonormal "input" or "analysing" basis vector directions for M
- The matrix U contains a set of orthonormal "output" basis vector directions for M
- The matrix Σ contains the singular values, which can be thought of as scalar "gain controls" by which each corresponding input is multiplied to give a corresponding output.

Constructor:

SVD(const MultiArray<float64,2> &Arg)

where:

Arg – is an m by n matrix with m >= n

Private attributes of the class:

- **MultiArray<float64,2> _u** – matrix containing orthonormal "output" basis vector directions for Arg
- **MultiArray<float64,2> _v** - matrix containing orthonormal "input" or "analysing" basis vector directions for Arg
- **MultiArray<float64,1> _s** – diagonal matrix with singular values.
- **int64 _m** – number of columns in Arg
- **int64 _n** – number of rows in Arg

Methods of the class:

- **void SVD::getU(MultiArray<float64,2> &A)** – return matrix U containing orthonormal "output" basis vector directions
- **void SVD::getV(MultiArray<float64,2> &A)** – Return the right singular vectors
- **void SVD::getSingularValues(MultiArray<float64,1> &x)** – Return the one-dimensional array of singular values
- **void SVD::getS(MultiArray<float64,2> &A)** - Return the diagonal matrix of singular values
- **float64 SVD::norm2()** - returns two norm i.e. the maximum value from singular values
- **float64 SVD::cond()** - returns two norm of condition number (max(S)/min(S))

- **int64 SVD::rank()** - returns effective numerical matrix rank

5.2.2 Interpolation function

This function is used for warping a image. It takes values from one matrix and relocates them to other indicated positions. If the indicated positions are not the exact values of matrix coordinates the functions use interpolation to count the correct values.

template<typename T>
MultiArray<T,2> interpolation (const MultiArray<T,2>& ma_Img,
const MultiArray<float64,2>& ma_X, const MultiArray<float64,2>& ma_Y,
const std::string& method)
where:

- **ma_Img** is the input matrix which values will be relocated.
- **ma_X** is matrix containing the new x coordinates (columns)
- **ma_Y** is matrix containing the new y coordinates (rows)
- **method** determine what kind of method will be used to interpolate the coordinates. Now only one method is available:
  - “nn” - nearest neighbour method.

Function returns matrix.

5.2.3 Art class

This class lets compute camera intrinsic and extrinsic parameters from the perspective projection matrix. The computed extrinsic parameters are: rotation matrix R, translation vector t, and intrinsic that is matrix A (see chapter 3.1).

Constructor:

Art (const MultiArray<float64,2>& ppm, int64 s = 1)

where:

ppm – perspective projection matrix

s – sign for focal length, by default 1

Private attributes of the class:

- **MultiArray<float64,2> _ma_A** – calibration matrix A,
- **MultiArray<float64,2> _ma_R** – rotation matrix R,
- **MultiArray<float64,1> _ma_t** – translation vector t,
- **float64 _fsign** – sign of focal length.

Methods of the class:

- **MultiArray<float64,2> reta()** - returns calibration matrix A,
- **MultiArray<float64,2> retr()** - returns rotation matrix R,
- **MultiArray<float64,1> rett()** - returns translation vector.

5.3 Rectification implementation

"Rectify" class is responsible for doing rectification. It do all the calculations, all needed operations indispensable to rectify the image.

Constructor

Rectify(const std::string& paramFileL,const std::string& paramFileR)

where:

- paramFileL – path to file with data about left camera calibration parameters
- paramFileR – path to file with data about right camera calibration parameters

The file format should be as follows:

```
[Image size:]
  768  //width
  576  //height

[Rotation matrix:]
  0.0135   0.9616    -0.2741
 0.9851   0.0342   0.1686
  0.1715    -0.2723   -0.9468

[Translation vector:]
  -187.7996    -131.4920    477.1830

[Calibration matrix:]
  536.9826    0.0000    326.4721
  0.0000    536.5694    249.3326
  0.0000    0.0000    1.0000

[Projection matrix:]
```

6.1148e+002 -1.9271e+002 -1.9952e+002 4.4551e+004
-2.5597e+001 -4.9420e+001 -6.2306e+002 1.4522e+005
6.1393e-001 5.5082e-001 -5.6540e-001 1.1527e+002

[Distortion parameters:]
-3.7158430763281464e-001 //kappa1
1.5427446086242583e-001 //kappa2
-9.2183481277601561e-004 //tau1
-1.5875244995451997e-004 //tau2

For rectification only rotation matrix, translation vector and calibration matrix are required, the other won't be used.

Private attributes of the class:

- **MultiArray<float64,2> _T1** – transformation matrix for left image,
- **MultiArray<float64,2> _T2** – transformation matrix for right image,
- **MultiArray<float64,2> _Pn1** – new perspective projection matrix for left camera, containing the baseline,
- **MultiArray<float64,2> _Pn2** – new perspective projection matrix for right camera, containing the baseline.

Methods of the class:

- **MultiArray<float64,2> ret_T1()** - returns the transformation matrix for the left camera,
- **MultiArray<float64,2> ret_T2()** - returns the transformation matrix for the right camera,
- **template<typename T, int N>**
  Image<T,N> do_rec (const Image<T,N>& oriImgL) – returns rectified image oriImgL

5.4 Intermediate view synthesis implementation

For intermediate view synthesis is responsible function inView. This function compute the virtual view, but it does not do the rectification. So before using this function you need to use the “rectify” class to have the images rectified and the output image need to be de-rectified.

ImageRGB< uint8 > inView (const std::string& imgL,const std::string& imgR,const std::string& disparityL,const std::string& disparityR, float64 position,const std::string& outImage)

where:

- imgL – path to real rectified left image ,
- imgR – path to real rectified right image,
- disparityL – path to disparity map for left image,
- disparityR – path to disparity map for right image,
- position – describes x coordinate of the virtual camera, values can be between 0 and 1, where 0 is close to left image and 1- close to right image. So for example 0.5 is in the middle.
- outImage – path for new intermediate view image.

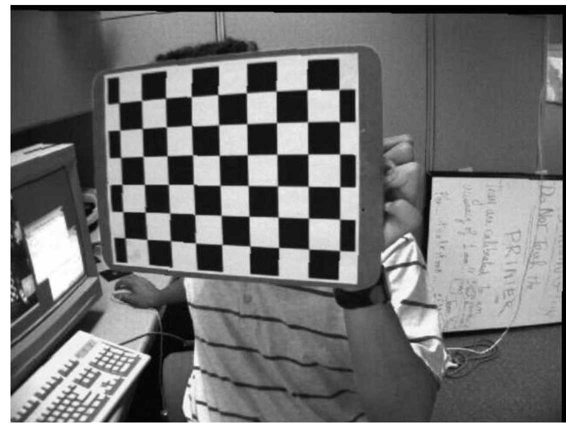
6 Tests

For test with rectification I used images delivered with matlab calibration toolbox. I also used the matlab code to compute the camera calibration parameters for this images. At the images below you can see the results. Unfortunately I did not put the epipolar lines on the images, but you can easily see the result of the rectification on the right top corner of the monitor, or on the left bottom corner of the chess box.
The original images and the rectified one are shown below.

a)

b)

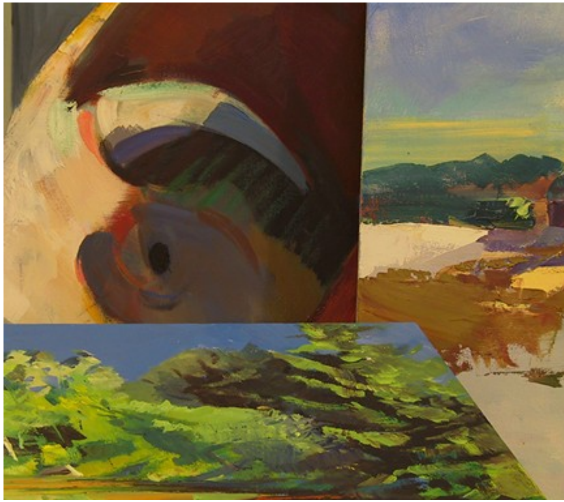
c)



d)

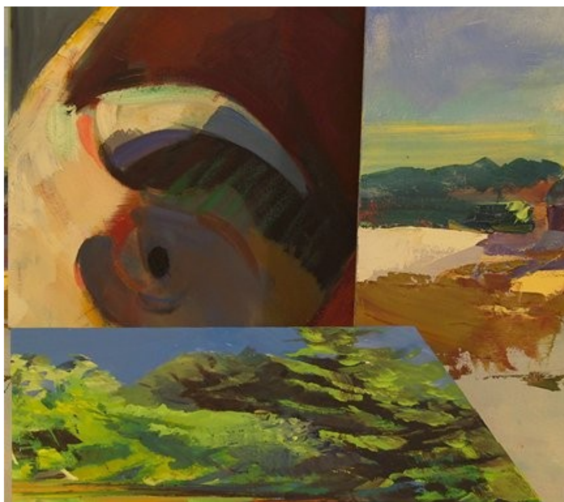
6.1 a) left image b) right image c) left rectified image d) right rectified image

For intermediate view synthesis I used images from http://vision.middlebury.edu/. Below are the results for quite simple view. There are no big differences between right and left camera, so the computed intermediate view is perfect.



a)

b)



c)

d)

6.2 a) real image from left camera b) virtual image from position 0.5 (in the middle between left and right real images) c) virtual image from position 0.75 (closer to right image) d) real image from right camera

a) b)

c) d)

6.3 a) real image from left camera b) virtual image from position 0.25 (closer to the left camera) c) virtual image from position 0.5 (in the middle between left and right real images) d) real image from right camera

Using views more complicated with many small details, and with quite big differences between position of left and right camera brings some artifacts. This is because there are many areas visible in one camera but not in the other.

Below are results of tests with such images. Images 6.4 b) and c) shows that the artifacts are smaller if the virtual view is closer to one of the cameras. In the middle the defects will be the biggest.

a)

b)

c)

d)

6.4 a) real image from left camera b) virtual image from position 0.25 (closer to the left camera) c) virtual image from position 0.5 (in the middle between left and right real images) d) real image from right camera

7 Conclusions

The idea of this work was to develop decent basics for intermediate view synthesis. I think this was accomplished. However its not complete intermediate view synthesis instrument, because there is no tool for obtaining disparity maps, but the foundation are implemented. There is tool for rectification and tool to compute the intermediate view from rectified images and disparity maps.

Tool for the rectification take me most of the time. This algorithm is quite complicated, and c++ is rather not mathematics friendly environment. The rectify class has 235 lines of code, and the SVD, interpolation and Art class has together 745 lines of code. The intermediate view part was simple to implement. Code of it has 153 lines.

This work also shows how many information we can obtain from two stereo images. It shows the power of stereo vision and its possibilities.

8 Future Work

There are quite some options to improve this tool. I will concentrate on obtaining disparity maps. As I mentioned before this part is needed to have complete intermediate view synthesis.

Other opportunity is to implement rectification methods for uncalibrated cameras. This would let this tool to be used with all possible input images.

If I will complete the disparity map tool, I rather concentrate on using this informations in mobile robots system that continue work with the intermediate view synthesis. Disparity map can be modified into depth map. Informations from depth map can be used to avoid obstacles. Robot then may use stereo camera in the same way as humans use eyes, to identify the environment around.

8 Bibliography

[1] Fusiello, Emanuele Trucco, Alessandro Verri - „Compact algorithm for rectification of stereo pairs" Machine Vision and Applications 2000

[2] http://en.wikipedia.org

[3] B. J. Lei E. A. Hendriks - „Multi-step View Synthesis with occlusion handling" Stuttgart, Germany 2001

[4] https://147.83.50.70/imageplus/

[5] Przemysław Kowalski Krzysztof Skabek - „Przetwarzanie informacji wizyjnej w komputerowym systemie z mobilną głowicą stereowizyjną" Studia Informatica 2001 n.3

[6] Jeffrey S. McVeigh, M.W. Siegel, A. G. Jordan - „Intermediate view synthesis considering occluded and ambiguously referenced image regions" Signal processing image communication 1996 n. 9

[7] author unknown - „View synthesis"

[8] Richard I. Hartley - „ Theory and Practice of Projective Rectification"