TECHNISCHE UNIVERSITÄT DRESDEN
EINFÜHRUNGSPRAKTIKUM COMPUTER VISION

Assignment 2 : Projection and Homography

Hassan Abu Alhaija

November 7,2014

1 INTRODUCTION

In this exercise session we will get a hands-on introduction to the basics of geometric transformation, projection and Homography. Those are the basics of image formation and are important to understand how the 3D real world is projected into the 2D images. This projection is a one-way transformation by nature due to loss of information and recovering 3D information back from 2D images is One of the main challenges in computer vision.

we will introduce here a few of the basic concepts we are going to use.

1.1 HOMOGENEOUS COORDINATES

In the 2D Euclidean space we usually represent a point P with two coordinates $P = (x, y)$. This is called Euclidean coordinates $\mathbb{R}^2$. We extend this representation by simply adding 1 as a third coordinate and get the presentation of the point $\tilde{P} = (x, y, 1)$. This is called **Homogeneous Coordinates** and denoted $\mathbb{P}^2$. The same applies when moving from 3D Euclidean coordinates $P = (x, y, z) \in \mathbb{R}^3$ to 3D homogeneous coordinates $\tilde{P} = (x, y, z, 1) \in \mathbb{P}^3$

Euclidean to Homogeneous :

$$(x, y) \in \mathbb{R}^2 \quad \Rightarrow \quad (x, y, 1) \in \mathbb{P}^2$$

The reason we use Homogeneous coordinates is that it makes using matrix multiplication much easier to represent transformations. After applying all transformations on the point, we might end up with a Homogeneous coordinate vector that doesn't have $\tilde{z} = 1$ as its last component. Here we need to take care when moving back to the Euclidean coordinates to divide the other two coordinates $\tilde{x}, \tilde{y}$ by the last component $\tilde{z}$. Note that when the homogeneous coordinates are not normalized (divided by the last component) we annotate them with the tilde notation $\tilde{x}, \tilde{y}, \tilde{z}$

Homogeneous to Euclidean :

$$(\tilde{x}, \tilde{y}, \tilde{z}) \in \mathbb{P}^2 \quad \Rightarrow \quad (x, y) \in \mathbb{R}^2$$

$$where \;\; x = \frac{\tilde{x}}{\tilde{z}} \quad , \quad y = \frac{\tilde{y}}{\tilde{z}}$$

1.2 Transformations

When working in Homogeneous coordinates, we can represent a geometric transformation of a point in d-dimensional $P \in \mathbb{R}^d$ space by a matrix $H \in \mathbb{R}^{(d+1)\times(d+1)}$. So for example when transforming 2D points $\mathbf{x} = (x, y, 1)$ we need a transformation matrix of size 3×3. We will take a look at some transformations (in 2D space) characterized by the properties of geometry they preserve.

1.2.1 Isometry Transformation

A transformation that preserve the distance between points is called an Isometry. This means that the distances between a group of points before and after the transformation stay the same. This includes Rotation and Translation of points. It can be written in matrix form as following :

$$\mathbf{x}' = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^\mathbf{T} & 1 \end{pmatrix} \mathbf{x}$$

where R is a 2×2 matrix, t is a translation 2D vector and 0^T is a vector of zeros.

1.2.2 Similarity Transformation

A Similarity transformation changes the distance between points but preserves the angles between points and lines. Also, two parallel lines before the transformation stay parallel after so it preserves the parallel property between lines. It includes rotation and translation but with a an isometric scale. That is the scaling with constant factor s in all directions.

$$\mathbf{x}' = \begin{pmatrix} s\mathbf{R} & \mathbf{t} \\ \mathbf{0}^\mathbf{T} & 1 \end{pmatrix} \mathbf{x}$$

1.2.3 Affine transformation

In this transformation the angles between lines are not preserved any more. Instead the only property preserved is the parallel property between lines. This includes scaling with different factors along different axis and an additional rotation angle for the axis along which the scaling happens.

$$\mathbf{x}' = \begin{pmatrix} \mathbf{A} & \mathbf{t} \\ \mathbf{0}^{\mathbf{T}} & 1 \end{pmatrix} \mathbf{x}$$

Where A is a 2×2 non-singular matrix that includes the two rotations and two scaling factors. it can be written as :

$$A = R(\alpha)(R(-\beta)DR(\beta))$$

Where $R(\alpha)$ and $R(\beta)$ are two rotation matrices for angles α and β. and D is the scaling matrix with two scaling factors λ_1 on x-axis and λ_2 on y-axis.

$$D = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}$$

1.2.4 Projective transformation

Projective transformation, also called a *Homography*, is the most general Transformation in sense of the properties it affects. Here the only property preserved is the linearity, meaning that straight lines stay straight after the Projective transformation (this is actually the weakest condition for a matrix to be called a transformation). While the angles, distance and parallelness all can change. This can be seen as a mixture of all the previous transforms. While all the other transformations are applied the same to all points regardless of their position, the projective transformation adds the property of having different transformation according to the position of the point in space. This is done through the vector v^T which replaces the 0^T in other transformation matrices. Here we can see the advantage of using homogeneous coordinates for doing this transformation.

$$\mathbf{x}' = \begin{pmatrix} \mathbf{A} & \mathbf{t} \\ \mathbf{v}^{\mathbf{T}} & v \end{pmatrix} \mathbf{x}$$

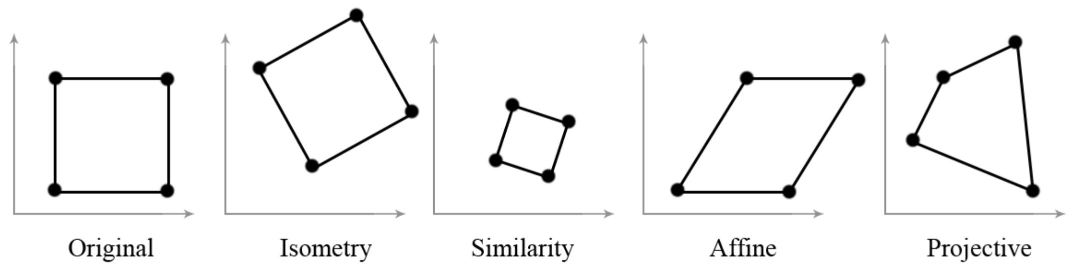

Figure 1.1: Transformation applied to a rectangle

1.3 Camera and Perspective Projection

Cameras in computer vision is devices that project the 3D world points into a 2D image plane. The most simple and common camera model is the *Pinhole camera.* In this model the image is formed on a plane set behind an infinitely small pinhole that allows only a single ray of light from each point in the scene to pass through it. The simplicity of this model comes from ability to characterize this camera by only one number which is the focal length f which is the distance from the whole to the image plane.

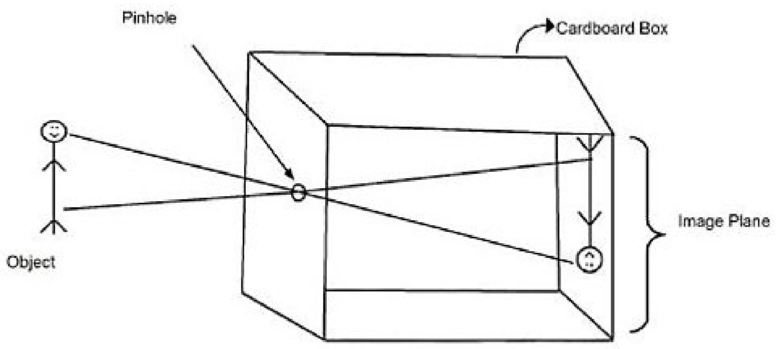

Figure 1.2: Pinhole camera model

The Perspective Projection is computing the projection of a 3D scene point $\tilde{P} = (X, Y, Z, 1)$ on the image plane as $\tilde{p} = (x, y, 1)$ using the 3×4 camera matrix C. In this case of pinhole camera the matrix C is very simple.

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \end{pmatrix} = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

Extending this simple pinhole camera into digital imaging needs some modifications. First the focal f in this case is in real world units (meters) and so the coordinates of projection x, y are also in meters. In digital images we need those coordinates in pixels (u, v) instead. For this conversion, we need the size of pixel in meters ρ and we can divide the focal length by it to move to pixel units. this gives us $\alpha = \frac{f}{\rho}$. The other modification we need is moving the center of coordinates. While in the pinhole camera case the center of coordinates is considered at the center of the image (behind the pinhole), in digital images the coordinates start from the upper left corner. For that we need a translation of amount (u_0, v_0) which is the distance from the corner to the center in pixels. After applying those modifications, we get our *intrinsic camera parameters matrix K*. This matrix has a total of 3 parameters and stays constant for the same camera regardless of its or object's position and rotation.

$$K = \begin{pmatrix} \alpha & 0 & u_0 \\ 0 & \alpha & v_0 \\ 0 & 0 & 1 \end{pmatrix}$$

The last ingredient we need is the *extrinsic camera parameters matrix* which represent the current status of the camera in the 3D scene. This is a 3×4 matrix that contains the 3×3 rotation matrix R and the 3×1 translation (position) vector t of the camera. This means it contain 6 independent parameters (degrees of freedom), 3 for rotation and 3 for translation.

$$K = \begin{pmatrix} \mathbf{R} & | & t \end{pmatrix}$$

The final perspective projective has 9 parameters and looks like this :

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{pmatrix} = \begin{pmatrix} \alpha & 0 & u_0 \\ 0 & \alpha & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} R_{11} & R_{12} & R_{13} & t_1 \\ R_{21} & R_{22} & R_{23} & t_2 \\ R_{31} & R_{32} & R_{33} & t_2 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

2 Exercise 01 : Homography

Sometimes in sport event broadcasting the names of teams, players or advertisement images are projected on the field with the right perspective to fit the images from the camera. Create an OpenCV program that takes two images as input. One is a perspective photo of a sport field, a swimming pool or any other arena. The other is a flat image of some logo or advertisement. You can chose ny clicking any 4 points on the photo of the arena and the logo image should be transformed and overplayed over the original photo.

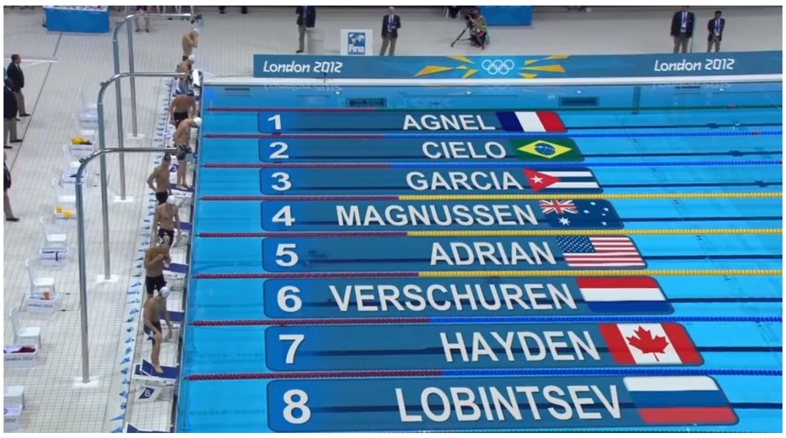

Figure 2.1: Homography in sport broadcasting

3 Exercise 02 :

Create a 3D scene in OpenCV where a rectangular image is positioned so that its centre is in the origin of the 3D coordinate system. A camera is looking always in the direction of the image (the coordinate origin) and gives an image of size 512 x 512 pixels and fix the focal length to a reasonable value (for example 100 pixels) to keep the image inside the view of the camera. The position of the camera is described by spherical coordinates (figure 3.1) and controller using three parameters that are input by the user using sliders on the main interface. Those are :

- r : the distance from the coordinates origin point.
- ϕ : the angle on the horizontal plane.
- θ : the angel with the vertical axis.

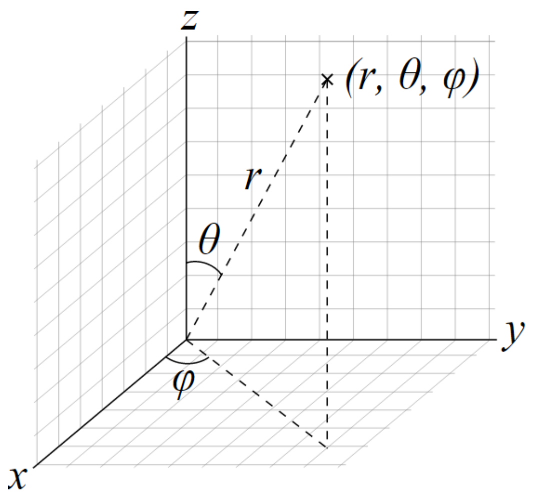

Figure 3.1: Spherical coordinate system

Remember that the camera still needs to always be looking at the coordinate origin point. Apply the proper computation and display the image as seen by the camera.

After that, apply an Affine transform on the image in the centre. The input is 3 parameters also controlled by the user using sliders (with reasonable range):

- α : main rotation angle.
- β : secondary rotation angle (related to the scaling).
- λ_1 : scale factor (the other factor $\lambda_2 = 1$)

4 SUBMISSION :

Please send me the code for each exercise in a separate file with any additional images or header files needed on this email : *hassan.abu_alhaija@tu-dresden.de*
Deadline : 28 November 2014.