# SPIV Configuration in Pinhole Camera Model

by Liu Ning

June 8, 2025

## 1 SPIV translation configuration

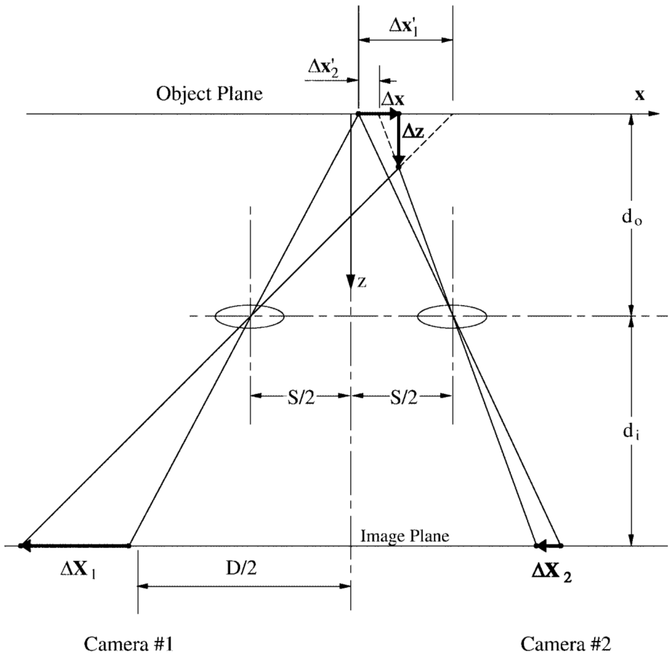


**Figure 1.** Schematic of stereocamera in the translation configuration.

### 1.1 Code validation

```
SymPy 1.13.3 under Python 3.13.1
Please see the documentation in Help -> Plugins -> SymPy
```

```
>>> from sympy import *
>>> do, di, S = symbols('d_o d_i S', real=True, positive=True)
>>> do + di + S
```

$$S + d_i + d_o$$

```
>>> Rl = Matrix([[1, 0, 0], [0, 1, 0], [0, 0, 1]])
>>> Rl # Rotating matrix from world frame to left camera frame
```

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

```
>>> Rr = Rl
```

```
>>> Rr # Rotating matrix from world frame to right camera frame
```

$$\begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & 1\end{bmatrix}$$

```
>>> Tl = Matrix([S/2, 0, -do])
>>> Tl # Translation vector from left camera frame origin to world frame origin
```

$$\begin{bmatrix}\frac{S}{2}\\0\\-d_o\end{bmatrix}$$

```
>>> Tr = Matrix([-S/2, 0, -do])
>>> Tr # Translation vector from right camera frame origin to world frame origin
```

$$\begin{bmatrix}-\frac{S}{2}\\0\\-d_o\end{bmatrix}$$

```
>>> x, y, z, dx, dy, dz = symbols('x y z Delta_x Delta_y Delta_z', real=True)
>>> x + y + z + dx + dy + dz
```

$$\Delta_x + \Delta_y + \Delta_z + x + y + z$$

```
>>> Xw1 = Matrix([x, y, z])
>>> Xw2 = Matrix([x+dx, y+dy, z+dz])
>>> Xw1 # Point 1 in world frame
```

$$\begin{bmatrix}x\\y\\z\end{bmatrix}$$

```
>>> Xw2 # Point 2 in world frame
```

$$\begin{bmatrix}\Delta_x + x\\\Delta_y + y\\\Delta_z + z\end{bmatrix}$$

```
>>> flipZ = Matrix([[1, 0, 0], [0, 1, 0], [0, 0, -1]])
>>> flipZ # Apply reflection matrix to flip Z direction between world frame and
    camera frame
```

$$\begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & -1\end{bmatrix}$$

```
>>> extrinsicl = Rl.row_join(Tl)
>>> extrinsicl # Extrinsic matrix ( [R | T] ) for left camera frame
```

$$\begin{bmatrix}1 & 0 & 0 & \frac{S}{2}\\0 & 1 & 0 & 0\\0 & 0 & 1 & -d_o\end{bmatrix}$$

```
>>> Pl = flipZ * extrinsicl
>>> Pl # Projection matrix: project point in world frame into left camera frame
```

$$\begin{bmatrix}1 & 0 & 0 & \frac{S}{2}\\0 & 1 & 0 & 0\\0 & 0 & -1 & d_o\end{bmatrix}$$

```
>>> extrinsicr = Rr.row_join(Tr)
>>> Pr = flipZ * extrinsicr
>>> Pr
```

$$\begin{bmatrix} 1 & 0 & 0 & -\frac{S}{2} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & d_o \end{bmatrix}$$

```
>>> Xl1 = Pl * Xw1.row_insert(3, Matrix([1]))
>>> Xl1 # Point 1 in left camera frame
```

$$\begin{bmatrix} \frac{S}{2}+x \\ y \\ d_o-z \end{bmatrix}$$

```
>>> Xl2 = Pl * Xw2.row_insert(3, Matrix([1]))
>>> Xl2 # Point 2 in left camera frame
```

$$\begin{bmatrix} \Delta_x+\frac{S}{2}+x \\ \Delta_y+y \\ -\Delta_z+d_o-z \end{bmatrix}$$

```
>>> Xr1 = Pr * Xw1.row_insert(3, Matrix([1]))
>>> Xr1
```

$$\begin{bmatrix} -\frac{S}{2}+x \\ y \\ d_o-z \end{bmatrix}$$

```
>>> Xr2 = Pr * Xw2.row_insert(3, Matrix([1]))
>>> Xr2
```

$$\begin{bmatrix} \Delta_x-\frac{S}{2}+x \\ \Delta_y+y \\ -\Delta_z+d_o-z \end{bmatrix}$$

```
>>> # Project points to object plane (in camera frame) using pinhole model
>>> Xol1 = do/(Xl1[2]) * Xl1 # Point 1 at object plane (in left camera frame)
>>> Xol1
```

$$\begin{bmatrix} \frac{d_o\left(\frac{S}{2}+x\right)}{d_o-z} \\ \frac{d_o y}{d_o-z} \\ d_o \end{bmatrix}$$

```
>>> Xol2 = do/(Xl2[2]) * Xl2 # Point 2 at object plane (in left camera frame)
>>> Xol2
```

$$\begin{bmatrix} \frac{d_o\left(\Delta_x+\frac{S}{2}+x\right)}{-\Delta_z+d_o-z} \\ \frac{d_o\left(\Delta_y+y\right)}{-\Delta_z+d_o-z} \\ d_o \end{bmatrix}$$

```
>>> Xor1 = do/(Xr1[2]) * Xr1
>>> Xor1
```

$$\begin{bmatrix} \frac{d_o\left(-\frac{S}{2}+x\right)}{d_o - z} \\ \frac{d_o\,y}{d_o - z} \\ d_o \end{bmatrix}$$

```
>>> Xor2 = do/(Xr2[2]) * Xr2
>>> Xor2
```

$$\begin{bmatrix} \frac{d_o\left(\Delta_x-\frac{S}{2}+x\right)}{-\Delta_z + d_o - z} \\ \frac{d_o\,(\Delta_y + y)}{-\Delta_z + d_o - z} \\ d_o \end{bmatrix}$$

```
>>> dXl = Xol2 - Xol1 # Distance in object plane (in left camera frame)
>>> dXl
```

$$\begin{bmatrix} -\frac{d_o\left(\frac{S}{2}+x\right)}{d_o - z} + \frac{d_o\left(\Delta_x+\frac{S}{2}+x\right)}{-\Delta_z + d_o - z} \\ -\frac{d_o\,y}{d_o - z} + \frac{d_o\,(\Delta_y + y)}{-\Delta_z + d_o - z} \\ 0 \end{bmatrix}$$

```
>>> dXr = Xor2 - Xor1 # Distance in object plane (in right camera frame)
>>> dXr
```

$$\begin{bmatrix} -\frac{d_o\left(-\frac{S}{2}+x\right)}{d_o - z} + \frac{d_o\left(\Delta_x-\frac{S}{2}+x\right)}{-\Delta_z + d_o - z} \\ -\frac{d_o\,y}{d_o - z} + \frac{d_o\,(\Delta_y + y)}{-\Delta_z + d_o - z} \\ 0 \end{bmatrix}$$

```
>>> # Check paper results
>>> dx_true = ((x - S/2)*dXl[0] - (x + S/2)*dXr[0]) / (-S - (dXl[0] - dXr[0]))
>>> simplify(dx_true)
```

$$\frac{\Delta_x\,d_o\,(d_o - z)}{\Delta_z\,z + d_o^2 - 2\,d_o\,z + z^2}$$

```
>>> simplify(dx_true.subs(z, 0))
```

$$\Delta_x$$

```
>>> dz_true = (-do*(dXl[0] - dXr[0])) / (-S - (dXl[0]-dXr[0]))
>>> simplify(dz_true)
```

$$\frac{\Delta_z\,d_o^2}{\Delta_z\,z + d_o^2 - 2\,d_o\,z + z^2}$$

```
>>> simplify(dz_true.subs(z, 0))
```

$$\Delta_z$$

```
>>>
```

其中，$\Delta \boldsymbol{X}_{\textbf{left camera}}$和$\Delta \boldsymbol{X}_{\textbf{right camera}}$在上述代码中对应**dXl**和**dXr**，分别表示颗粒运动距离在object plane（对应激光面）中的投影[1]。两个矢量的第一项可与$\Delta X_{\text{left}}$和$\Delta X_{\text{right}}$构成两个线性方程组，

$$
\begin{aligned}
-\frac{d_o\left(\frac{S}{2}+x\right)}{d_o-z}+\frac{d_o\left(\Delta x_{\text{world}}+\frac{S}{2}+x\right)}{-\Delta z_{\text{world}}+d_o-z} &= \Delta X_{\text{left}},\\
-\frac{d_o\left(-\frac{S}{2}+x\right)}{d_o-z}+\frac{d_o\left(\Delta x_{\text{world}}-\frac{S}{2}+x\right)}{-\Delta z_{\text{world}}+d_o-z} &= \Delta X_{\text{right}}.
\end{aligned}
$$

求解得到世界坐标系下的$\Delta x_{\text{world}}$和$\Delta z_{\text{world}}$为

$$
\begin{aligned}
\Delta x_{\text{world}} &= \frac{\Delta X_{\text{left}}(x-S/2)-\Delta X_{\text{right}}(x+S/2)}{-S-(\Delta X_{\text{left}}-\Delta X_{\text{right}})},\\
\Delta z_{\text{world}} &= \frac{-d_o(\Delta X_{\text{left}}-\Delta X_{\text{right}})}{-S-(\Delta X_{\text{left}}-\Delta X_{\text{right}})}.
\end{aligned}
$$

而两个矢量的第二项可以分别独立求解世界坐标系下的$\Delta y_{\text{world}}$，

$$
\begin{aligned}
-\frac{d_o y}{d_o-z}+\frac{d_o(\Delta y_{\text{world}}+y)}{-\Delta z_{\text{world}}+d_o-z} &= \Delta Y_{\text{left}},\\
-\frac{d_o y}{d_o-z}+\frac{d_o(\Delta y_{\text{world}}+y)}{-\Delta z_{\text{world}}+d_o-z} &= \Delta Y_{\text{right}}.
\end{aligned}
$$

将二者的求解结果取平均以提高结果的精确性，

$$
\Delta y_{\text{world}} = \frac{-y\Delta z_{\text{world}}}{d_o}+\frac{1}{2}\cdot(\Delta Y_{\text{left}}+\Delta Y_{\text{right}})\left(1-\frac{\Delta z}{d_o}\right).
$$

至此，颗粒的实际距离矢量$[\ \Delta x_{\text{world}}\ \ \Delta y_{\text{world}}\ \ \Delta z_{\text{world}}\ ]^T$均已求出。

# 2 SPIV angular-displacement configuration

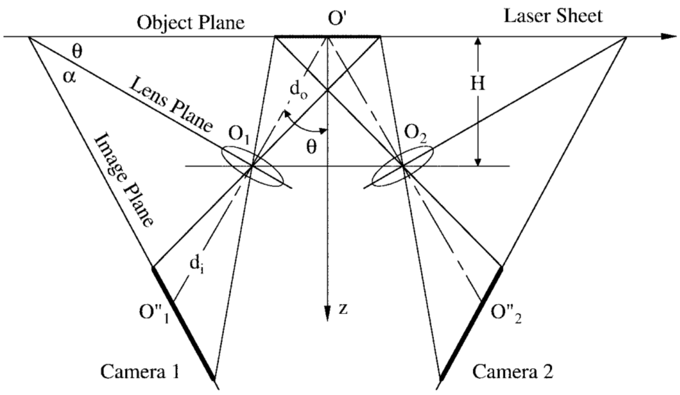


**Figure 2.** Schematic of stereocamera in the angular-displacement configuration.

1. 注意此处没有讨论成像平面上的投影。在平行布置的情况下，object、lens和image三者平面平行，object plane上的运动距离投影矢量$\Delta \boldsymbol{x_o}$满足$\Delta \boldsymbol{x_o}/d_o = -\Delta \boldsymbol{x_i}/d_i$（相似三角形）。

## 2.1 沙姆定律（Scheimpflug principle）

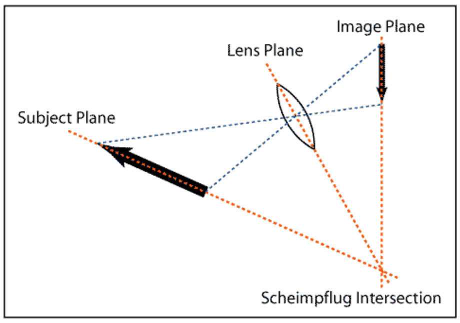


**Figure 3.** The angles of the Scheimpflug principle, using the example of a photographic lens.

只要满足成像、镜头、测量三个平面交于同一条直线（对于未移轴的镜头，认为三个平行平面相交于无限远处）的条件，则可以保证测量平面的物体能够清晰成像，因此能够起到调整景深区域位置的作用。

目的：

- 增大公共景深区域（common focus area）

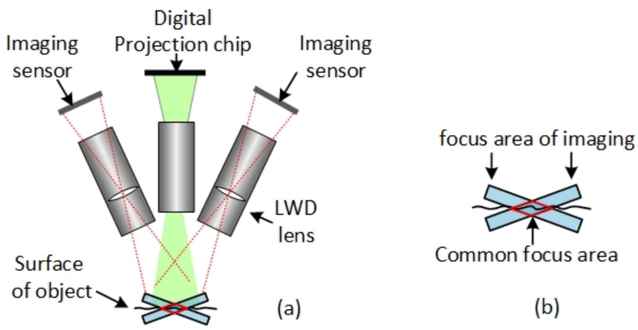


**Figure 4.** (a) The schematic of setup. (b) The common focus area (Marked with red quadrangle).

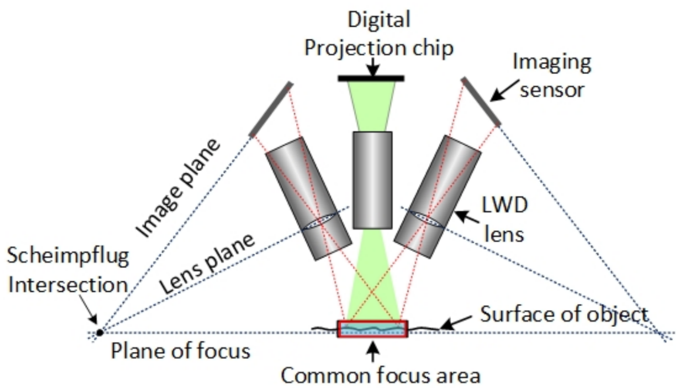


**Figure 5.** System design based on the Scheimpflug principle.

副作用：

- 不均匀放大倍率 $\Longleftarrow \alpha \neq 0$

- 同侧布置两台相机畸变拉伸方向相反（oppositely stretched）

### 2.2 $\alpha = 0$ condition: Image Plane // Lens Plane

为满足$\alpha=0$情况的成像条件，需要牺牲光圈[2]，增大景深，使其能够覆盖颗粒运动距离，

$$\delta z = 4(1+M_n^{-1})^2 f^{\#^2}\lambda.$$

其中，$\lambda$为激光波长，$M_n = d_i/d_o$为相机放大倍率（Camera magnification）。

```
   SymPy 1.13.3 under Python 3.13.1
   Please see the documentation in Help -> Plugins -> SymPy
>>> from sympy import *
>>> theta, do, di, S = symbols('theta d_o d_i S', real=True, positive=True)
>>> theta + di + S + do
```

$$S + d_i + d_o + \theta$$

```
>>> Rl = Matrix([[cos(theta), 0, -sin(theta)], [0, 1, 0], [sin(theta), 0,
    cos(theta)]])
>>> Rl # Rotating matrix from world frame to left camera frame
```

$$\begin{bmatrix} \cos(\theta) & 0 & -\sin(\theta) \\ 0 & 1 & 0 \\ \sin(\theta) & 0 & \cos(\theta) \end{bmatrix}$$

```
>>> Rr = Matrix([[cos(theta), 0, sin(theta)], [0, 1, 0], [-sin(theta), 0,
    cos(theta)]])
>>> Rr # Rotating matrix from world frame to right camera frame
```

$$\begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}$$

```
>>> Tl = Matrix([do*sin(theta), 0, -do*cos(theta)])
>>> Tl # Translation vector from left camera frame origin to world frame origin
```

$$\begin{bmatrix} d_o\sin(\theta) \\ 0 \\ -d_o\cos(\theta) \end{bmatrix}$$

```
>>> Tr = Matrix([-do*sin(theta), 0, -do*cos(theta)])
>>> Tr # Translation vector from right camera frame origin to world frame origin
```

$$\begin{bmatrix} -d_o\sin(\theta) \\ 0 \\ -d_o\cos(\theta) \end{bmatrix}$$

```
>>> x, y, z, dx, dy, dz = symbols('x y z Delta_x Delta_y Delta_z', real=True)
>>> x + y + z + dx + dy + dz
```

$$\Delta_x + \Delta_y + \Delta_z + x + y + z$$

```
>>> Xw1 = Matrix([x, y, z])
>>> Xw2 = Matrix([x+dx, y+dy, z+dz])
>>> Xw1 # Point 1 in world frame
```

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

2. $f^{\#}$值越大，光圈越小，得到的景深$\delta z$越大。

```
>>> Xw2 # Point 2 in world frame
```

$$\begin{bmatrix} \Delta_x + x \\ \Delta_y + y \\ \Delta_z + z \end{bmatrix}$$

```
>>> flipZ = Matrix([[1, 0, 0], [0, 1, 0], [0, 0, -1]])
>>> flipZ # Apply reflection matrix to flip Z direction between world frame and
    camera frame
```

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

```
>>> extrinsicl = Rl.row_join(Tl) # Extrinsic matrix ( [R | T] ) for left camera
    frame
>>> extrinsicl
```

$$\begin{bmatrix} \cos(\theta) & 0 & -\sin(\theta) & d_o \sin(\theta) \\ 0 & 1 & 0 & 0 \\ \sin(\theta) & 0 & \cos(\theta) & -d_o \cos(\theta) \end{bmatrix}$$

```
>>> Pl = flipZ * extrinsicl
>>> Pl # Projection matrix: project point in world frame into left camera frame
```

$$\begin{bmatrix} \cos(\theta) & 0 & -\sin(\theta) & d_o \sin(\theta) \\ 0 & 1 & 0 & 0 \\ -\sin(\theta) & 0 & -\cos(\theta) & d_o \cos(\theta) \end{bmatrix}$$

```
>>> extrinsicr = Rr.row_join(Tr)
>>> Pr = flipZ * extrinsicr
>>> Pr
```

$$\begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) & -d_o \sin(\theta) \\ 0 & 1 & 0 & 0 \\ \sin(\theta) & 0 & -\cos(\theta) & d_o \cos(\theta) \end{bmatrix}$$

```
>>> Xl1 = Pl * Xw1.row_insert(3, Matrix([1]))
>>> Xl1 # Point 1 in left camera frame
```

$$\begin{bmatrix} d_o \sin(\theta) + x \cos(\theta) - z \sin(\theta) \\ y \\ d_o \cos(\theta) - x \sin(\theta) - z \cos(\theta) \end{bmatrix}$$

```
>>> Xl2 = Pl * Xw2.row_insert(3, Matrix([1]))
>>> Xl2 # Point 2 in left camera frame
```

$$\begin{bmatrix} d_o \sin(\theta) + (\Delta_x + x) \cos(\theta) - (\Delta_z + z) \sin(\theta) \\ \Delta_y + y \\ d_o \cos(\theta) - (\Delta_x + x) \sin(\theta) - (\Delta_z + z) \cos(\theta) \end{bmatrix}$$

```
>>> Xr1 = Pr * Xw1.row_insert(3, Matrix([1]))
>>> Xr1
```

$$\begin{bmatrix} -d_o \sin(\theta) + x \cos(\theta) + z \sin(\theta) \\ y \\ d_o \cos(\theta) + x \sin(\theta) - z \cos(\theta) \end{bmatrix}$$

```
>>> Xr2 = Pr * Xw2.row_insert(3, Matrix([1]))
>>> Xr2
```

$$\begin{bmatrix} -d_o \sin(\theta) + (\Delta_x + x) \cos(\theta) + (\Delta_z + z) \sin(\theta) \\ \Delta_y + y \\ d_o \cos(\theta) + (\Delta_x + x) \sin(\theta) - (\Delta_z + z) \cos(\theta) \end{bmatrix}$$

```
>>> # Project points to object plane (in camera frame) using pinhole model
>>> Xol1 = (do/(1-tan(theta)*Xl1[0]/Xl1[2])) / (Xl1[2]) * Xl1 # Point 1 at object
    plane (in left camera frame)
>>> simplify(Xol1)
```

$$\begin{bmatrix} \frac{d_o(-d_o\sin(2\theta) - x\cos(2\theta) - x + z\sin(2\theta))}{2(-d_o\cos(2\theta) + x\sin(2\theta) + z\cos(2\theta))} \\ \frac{d_o y\cos(\theta)}{d_o\cos(2\theta) - x\sin(2\theta) - z\cos(2\theta)} \\ \frac{d_o(-d_o\cos(2\theta) - d_o + x\sin(2\theta) + z\cos(2\theta) + z)}{-2d_o\cos(2\theta) + 2x\sin(2\theta) + 2z\cos(2\theta)} \end{bmatrix}$$

```
>>> simplify(Xol1.subs(theta, 0)) # Compare to translation case (check)
```

$$\begin{bmatrix} \frac{d_o x}{d_o - z} \\ \frac{d_o y}{d_o - z} \\ d_o \end{bmatrix}$$

```
>>> Xol2 = (do/(1-tan(theta)*Xl2[0]/Xl2[2])) / (Xl2[2]) * Xl2 # Point 2 at object
    plane (in left camera frame)
>>> simplify(Xol1)
```

$$\begin{bmatrix} \frac{d_o(-d_o\sin(2\theta) - x\cos(2\theta) - x + z\sin(2\theta))}{2(-d_o\cos(2\theta) + x\sin(2\theta) + z\cos(2\theta))} \\ \frac{d_o y\cos(\theta)}{d_o\cos(2\theta) - x\sin(2\theta) - z\cos(2\theta)} \\ \frac{d_o(-d_o\cos(2\theta) - d_o + x\sin(2\theta) + z\cos(2\theta) + z)}{-2d_o\cos(2\theta) + 2x\sin(2\theta) + 2z\cos(2\theta)} \end{bmatrix}$$

```
>>> simplify(Xol2.subs(theta, 0)) # Compare to translation case (check)
```

$$\begin{bmatrix} -\frac{d_o(\Delta_x + x)}{\Delta_z - d_o + z} \\ -\frac{d_o(\Delta_y + y)}{\Delta_z - d_o + z} \\ d_o \end{bmatrix}$$

```
>>> # For right camera, only change theta to -theta.
>>> Xor1 = (do/(1+tan(theta)*Xr1[0]/Xr1[2])) / (Xr1[2]) * Xr1 # Point 1 at object
    plane (in right camera frame)
>>> simplify(Xor1)
```

$$\begin{bmatrix} \frac{d_o(-d_o\sin(2\theta) + x\cos(2\theta) + x + z\sin(2\theta))}{2(d_o\cos(2\theta) + x\sin(2\theta) - z\cos(2\theta))} \\ \frac{d_o y\cos(\theta)}{d_o\cos(2\theta) + x\sin(2\theta) - z\cos(2\theta)} \\ \frac{d_o(d_o\cos(2\theta) + d_o + x\sin(2\theta) - z\cos(2\theta) - z)}{2(d_o\cos(2\theta) + x\sin(2\theta) - z\cos(2\theta))} \end{bmatrix}$$

```
>>> simplify(Xor1.subs(theta, 0)) # Compare to translation case (check)
```

$$\begin{bmatrix} \frac{d_o x}{d_o - z} \\ \frac{d_o y}{d_o - z} \\ d_o \end{bmatrix}$$

```
>>> Xor2 = (do/(1+tan(theta)*Xr2[0]/Xr2[2])) / (Xr2[2]) * Xr2 # Point 2 at object
    plane (in right camera frame)
>>> simplify(Xor2)
```

$$\begin{bmatrix} \dfrac{d_o\left(-d_o\sin(\theta)+(\Delta_x+x)\cos(\theta)+(\Delta_z+z)\sin(\theta)\right)\cos(\theta)}{\Delta_x\sin(2\theta)-\Delta_z\cos(2\theta)+d_o\cos(2\theta)+x\sin(2\theta)-z\cos(2\theta)} \\ \dfrac{d_o(\Delta_y+y)\cos(\theta)}{\Delta_x\sin(2\theta)-\Delta_z\cos(2\theta)+d_o\cos(2\theta)+x\sin(2\theta)-z\cos(2\theta)} \\ \dfrac{d_o\left(d_o\cos(\theta)+(\Delta_x+x)\sin(\theta)-(\Delta_z+z)\cos(\theta)\right)\cos(\theta)}{\Delta_x\sin(2\theta)-\Delta_z\cos(2\theta)+d_o\cos(2\theta)+x\sin(2\theta)-z\cos(2\theta)} \end{bmatrix}$$

```
>>> simplify(Xor2.subs(theta, 0)) # Compare to translation case (check)
```

$$\begin{bmatrix} -\dfrac{d_o(\Delta_x+x)}{\Delta_z-d_o+z} \\ -\dfrac{d_o(\Delta_y+y)}{\Delta_z-d_o+z} \\ d_o \end{bmatrix}$$

```
>>> dXl = Xol2 - Xol1 # Distance in object plane (in left camera frame)
>>> simplify(dXl.subs(theta, 0))
```

$$\begin{bmatrix} \dfrac{d_o\left(-x(\Delta_z-d_o+z)-(\Delta_x+x)(d_o-z)\right)}{(d_o-z)(\Delta_z-d_o+z)} \\ \dfrac{d_o\left(-y(\Delta_z-d_o+z)-(\Delta_y+y)(d_o-z)\right)}{(d_o-z)(\Delta_z-d_o+z)} \\ 0 \end{bmatrix}$$

```
>>> dXr = Xor2 - Xor1 # Distance in object plane (in right camera frame)
>>> simplify(dXr.subs(theta, 0)) # should sameas dXl under theta=0 condition
```

$$\begin{bmatrix} \dfrac{d_o\left(-x(\Delta_z-d_o+z)-(\Delta_x+x)(d_o-z)\right)}{(d_o-z)(\Delta_z-d_o+z)} \\ \dfrac{d_o\left(-y(\Delta_z-d_o+z)-(\Delta_y+y)(d_o-z)\right)}{(d_o-z)(\Delta_z-d_o+z)} \\ 0 \end{bmatrix}$$

```
>>>
```

## 2.3 双相机异侧布置

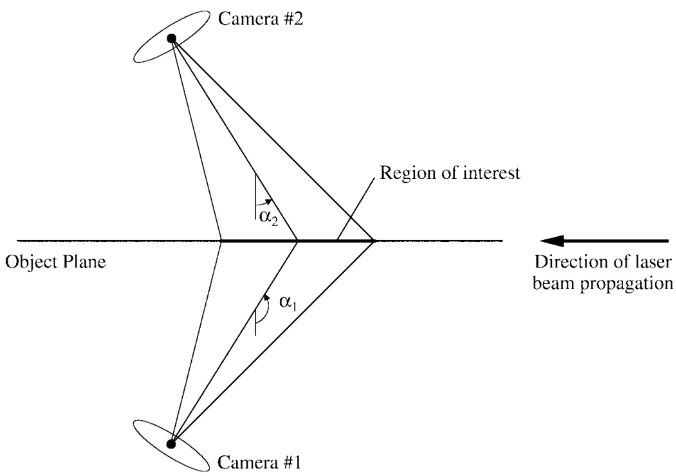


**Figure 6.** Stereoscopic arrangement with cameras on either side of the light sheet (adapted from Willert 1997).

优势:

- 增大forward scatter，提高信噪比?
- 两台相机的畸变拉伸方向一致

## 2.4 水棱镜