1 问题描述

在利用2d图像进行相机位姿估计时需要进行实验验证，因此根据相机模型编写仿真代码，将3维空间点映射到最终的像素平面。

2 相机模型

总体过程可以分位3部分：

1. 将世界坐标系下的三维点用相机坐标系坐标表示
2. 将相机坐标系下的点映射到成像平面，这个过程中深度信息丢失
3. 将成像平面的点映射到像素坐标系，也就是最终相机拍摄到的照片上的像素坐标

2.1 世界坐标系-相机坐标系

2.1.1 欧拉角表示

这是一个单纯的旋转平移过程

1. 现在世界坐标系下平移$[T_x, T_y, T_z]$

$$T\left(T_x, T_y, T_z\right)=\begin{bmatrix}1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1\end{bmatrix} \tag{1}$$

2. 先绕世界坐标系的x轴旋转α

$$\mathbf{R}_x(\alpha)=\begin{bmatrix}1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1\end{bmatrix} \tag{2}$$

3. 再绕世界坐标系的y轴旋转β

$$\mathbf{R}_y(\beta)=\begin{bmatrix}\cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1\end{bmatrix} \tag{3}$$

4. 最后绕世界坐标系的z轴旋转γ

$$\mathbf{R}_z(\gamma)=\begin{bmatrix}\cos\gamma & -\sin\gamma & 0 & 0\\ \sin\gamma & \cos\gamma & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix} \tag{4}$$

因为是对坐标系的变换，因此按照顺序左乘各个矩阵可以描述为一个总体的变换矩阵

$$T=T(T_x,T_y,T_z)R_z(\gamma)R_y(\beta)R_x(\alpha) \tag{5}$$

假设$P_w:(X_w,Y_w,Z_w)$是世界坐标系下的坐标，$P_c:(X_c,Y_c,Z_c)$是相机坐标系下的坐标

则

$$\begin{aligned}P_w&=TP_c\\ P_c&=T^{-1}P_w\end{aligned} \tag{6}$$

2.1.2 四元数表示

因为(5)中的运算涉及到大量三角函数的运算，使问题复杂，因此引入四元数的表达方式，用4个变量表示旋转矩阵

设四元数

$$q=(w,x,y,z) \tag{7}$$

其中

$$w^2+x^2+y^2+z^2=1 \tag{8}$$

四元数到旋转矩阵的变化可以描述为（齐次坐标系）

$$R=\begin{bmatrix}-2y^2-2z^2+1 & -2wz+2xy & 2wy+2xz & 0\\ 2wz+2xy & -2x^2-2z^2+1 & -2wx+2yz & 0\\ -2wy+2xz & 2wx+2yz & -2x^2-2y^2+1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix} \tag{9}$$

平移矩阵的逆可以表达为

$$T\left(T_x,T_y,T_z\right)=\begin{bmatrix}1 & 0 & 0 & -T_x\\ 0 & 1 & 0 & -T_y\\ 0 & 0 & 1 & -T_z\\ 0 & 0 & 0 & 1\end{bmatrix} \tag{10}$$

总的坐标变换矩阵可以写为

$$P_c=TP_w \tag{11}$$

其中

$$T=RT(T_x,T_y,T_z)=\begin{bmatrix} -2y^2-2z^2+1 & -2wz+2xy & 2wy+2xz & -T_x\left(-2y^2-2z^2+1\right)-T_y\left(-2wz+2xy\right)-T_z\left(2wy+2xz\right) \\ 2wz+2xy & -2x^2-2z^2+1 & -2wx+2yz & -T_x\left(2wz+2xy\right)-T_y\left(-2x^2-2z^2+1\right)-T_z\left(-2wx+2yz\right) \\ -2wy+2xz & 2wx+2yz & -2x^2-2y^2+1 & -T_x\left(-2wy+2xz\right)-T_y\left(2wx+2yz\right)-T_z\left(-2x^2-2y^2+1\right) \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{12}$$

2.2 相机坐标系-成像平面坐标系

相机成像原理是在光在传感器上产生信号，也就是说实际上空间点和传感器上的位置是一一对应的。控制这个对应关系的变量是焦距f

根据相似三角形的几何特征，很容易可以得出

$$\begin{cases} x=f\frac{X_c}{Z_c} \\ y=f\frac{Y_c}{Z_c} \\ z=f \end{cases} \tag{13}$$

其中(X_c,Y_c,Z_c)为相机坐标系中的坐标，(x,y,z)为成像平面坐标系中的坐标。为了和之前的齐次坐标相结合，这个变换也可以写成

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}=\begin{bmatrix} \frac{f}{Z_c} & 0 & 0 & 0 \\ 0 & \frac{f}{Z_c} & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \tag{14}$$

2.3 成像平面坐标系-像素坐标系

最后一步是转化到像素坐标系，也就是最终图像上的位置

上面推导中使用的像点坐标(x,y)是成像平面坐标系下，以成像平面的中心为原点。而实际像素点的表示方法是以像素来描述，坐标原点通常是图像的左上角，X轴沿着水平方向向左，Y轴竖直向下。像素是一个矩形块，这里假设其在水平和竖直方向的长度分别为：μ_x和μ_y。所以像素坐标和成像平面坐标之间，相差了一个**缩放**和**原点的平移**。

假设像素坐标的水平方向的轴为u，竖直方向的轴为v，那么将一个成像平面的坐标(x,y)在水平方向上缩放μ_x倍，在竖直方向上缩放μ_y倍，同时平移(c_x,c_y)，就可以得到像素坐标系的坐标(u,v)，其公式如下：

$$\begin{aligned} u&=\frac{x}{\mu_x}+c_x \\ v&=\frac{y}{\mu_y}+c_y \end{aligned} \tag{15}$$

写成齐次坐标的形式为

$$\begin{bmatrix} \mu \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{\mu_x} & 0 & c_x \\ 0 & \frac{1}{\mu_y} & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{16}$$

2.4 合并

将上述变换合成，可以得到

$$Z_c \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{\mu_x} & 0 & c_x \\ 0 & \frac{1}{\mu_y} & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R & T \\ \vec{0} & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{17}$$

3 代码实现

```python
class VirtualCamera:
    def __init__(self, alpha, beta, gamma, Tx, Ty, Tz, focal, resolution) -> None:
        """ Initiate the camera parameter
        Args:
            alpha ([float]): [Rotation along X-axis]
            beta ([float]): [Rotation along Y-axis]
            gamma ([float]): [Rotation along Z-axis]
            Tx ([float]): [Position along X-axis]
            Ty ([float]): [Position along Y-axis]
            Tz ([float]): [Position along Z-axis]
        """
        self.alpha = alpha
        self.beta = beta
        self.gamma = gamma
        self.Tx = Tx
        self.Ty = Ty
        self.Tz = Tz
        self.focal = focal
        self.resolution = np.array(resolution)
        self.center = np.array(resolution)/2
        self.pixel_size_x = 3.45e-6
        self.pixel_size_y = 3.45e-6
        self.RT_camera_in_world = np.zeros((4, 4))
        self.RT_world_in_camera = np.zeros((4, 4))
        self.update()

    def update(self):
        Rx = np.array([[1, 0, 0], [0, np.cos(
            self.alpha), -np.sin(self.alpha)], [0, np.sin(self.alpha), np.cos(self.alpha)]])
        Ry = np.array([[np.cos(self.beta), 0, -np.sin(self.beta)],
                       [0, 1, 0], [np.sin(self.beta), 0, np.cos(self.beta)]])
        Rz = np.array([[np.cos(self.gamma), -np.sin(self.gamma), 0],
                       [np.sin(self.gamma), np.cos(self.gamma), 0], [0, 0, 1]])
```

```python
        R0 = np.matmul(Rz, np.matmul(Ry, Rx))
        R1 = np.column_stack((R0, np.array([self.Tx, self.Ty, self.Tz]).T))
        R2 = np.row_stack((R1, [0, 0, 0, 1]))
        self.RT_camera_in_world = R2
        camera_in_image = np.array([[self.focal, 0, 0, 0],
                                    [0, self.focal, 0, 0],
                                     [0, 0, 1, 0]])
        image_in_pixel = np.array([[1/self.pixel_size_x, 0, self.center[0]],
                                    [0, 1/self.pixel_size_y, self.center[1]],
                                    [0, 0, 1]])

        self.RT_world_in_camera = np.linalg.inv(R2)

    def project_world_to_camera(self, points):
        assert points.shape[1] == 3, 'The n points\' shape should be (n,3)'
        n = points.shape[0]
        points = points.T
        points = np.row_stack((points, np.ones((1, n))))

        points = self.RT_world_in_camera@points
        points = points.T
        return points[:, 0:3]

    def project_camera_to_image(self, points):
        assert points.shape[1] == 3, 'The n points\' shape should be (n,3)'
        n = points.shape[0]
        points = points.T
        z = points[2, :]
        # print(z)
        trans = np.array([[self.focal, 0, 0],
                          [0, self.focal, 0],
                          [0, 0, 1]])
        points = trans@points
        points = points/z
        points = points.T
        return points

    def project_image_to_pixel(self, points):
        assert points.shape[1] == 3, 'The n points\' shape should be (n,3)'
        n = points.shape[0]
        points = points.T
        trans = np.array([[1/self.pixel_size_x, 0, self.center[0]],
                          [0, 1/self.pixel_size_y, self.center[1]],
                          [0, 0, 1]])
        points = trans@points
        points = points.T
        return points

    def project_world_to_pixel(self, points):
        camera_points = self.project_world_to_camera(points)
        image_points = self.project_camera_to_image(camera_points)
        pixel_points = self.project_image_to_pixel(image_points)
```

```
        return pixel_points
```

4 实验结果

在gazebo中进行仿真，将参数带入上述代码，进行对比，实验结果支持了模型的正确性