Determining Camera XY-Axis in MuJoCo

Alireza Azimi

2026-05-24

Introduction

In mujoco schemas you can define camera objects like the following:

```
<camera name="my_camera" pos="0 0 2" xyaxes="1 0 0 0 1 0"/>
```

This camera has default xyaxes values and depending on it's frame of reference it's "lookat" vector is determined by the z-axis orthogonal to the xyaxes: $\mathbf{z} = [0, 0, 1]^T$.

Now if we want to do the opposite, that is, to determine xyaxes from the $\mathbf{z}$ vector we need to do a bit of linear algebra. We need to construct an orthonormal set of vectors $\mathbf{x}, \mathbf{y}, \mathbf{z}$. In this problem we know $\mathbf{z}$ but we need t o determine a plausible xyaxes. We can do so by first considering a non-parrallel normal vector $\mathbf{u}$ to $\mathbf{z}$:

$$\mathbf{x} = \frac{\mathbf{u} \times \mathbf{z}}{\|\mathbf{u} \times \mathbf{z}\|}$$

And finally to get $\mathbf{y}$:

$$\mathbf{y} = \mathbf{z} \times \mathbf{x}$$

This produces a valid orthonormal set $\mathbf{x}, \mathbf{y}, \mathbf{z}$ for our camera object.

Implementation

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from ipywidgets import interact, FloatSlider
```

```
def orthonormal_basis_from_z(z):
    """
    Constructs an orthonormal basis (x, y, z) given a z-axis vector.

    Args:
        z (np.ndarray): A 3-element array representing the z-axis direction.

    Returns:
        tuple: A tuple of three np.ndarray vectors (x, y, z), each of shape (3,), forming an
    """
    z = z / np.linalg.norm(z) # normalize z (or ensure it's normalized)

    ## Example
    ##  If z = [0,0,1] then u = [1,0,0]
    ##  Else z = [0,1,0] then u = [0,0,1]
    u = np.array([0, 0, 1]) if np.abs(z[2]) < 0.999 else np.array([0, 1, 0])

    x = np.cross(u, z) # x = u cross z
    x /= np.linalg.norm(x) # normalize x
    y = np.cross(z, x) # y = z cross x
    return x, y, z
```

```
# Test orthonormal_basis_from_z with a known z vector
z_test = np.array([0, 0, 1])
x, y, z = orthonormal_basis_from_z(z_test)
print(x,y,z)
# Check orthonormality
assert np.allclose(np.dot(x, y), 0), "x and y are not orthogonal"
assert np.allclose(np.dot(y, z), 0), "y and z are not orthogonal"
assert np.allclose(np.dot(z, x), 0), "z and x are not orthogonal"
assert np.allclose(np.linalg.norm(x), 1), "x is not unit length"
assert np.allclose(np.linalg.norm(y), 1), "y is not unit length"
assert np.allclose(np.linalg.norm(z), 1), "z is not unit length"
print("All assertions passed.")
```

```
[1. 0. 0.] [0. 1. 0.] [0. 0. 1.]
All assertions passed.
```

```python
def plot_basis(x, y, z, origin=np.zeros(3)):
    fig = plt.figure(figsize=(6,6))
    ax = fig.add_subplot(111, projection='3d')

    ax.quiver(*origin, *x, color='r', label='x', length=1, normalize=True)
    ax.quiver(*origin, *y, color='g', label='y', length=1, normalize=True)
    ax.quiver(*origin, *z, color='b', label='z', length=1, normalize=True)

    ax.set_xlim([-1, 1])
    ax.set_ylim([-1, 1])
    ax.set_zlim([-1, 1])
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')
    ax.set_title("Interactive Orthonormal Basis from z-axis")
    ax.legend([
        f"x: [{x[0]:.2f}, {x[1]:.2f}, {x[2]:.2f}]",
        f"y: [{y[0]:.2f}, {y[1]:.2f}, {y[2]:.2f}]",
        f"z: [{z[0]:.2f}, {z[1]:.2f}, {z[2]:.2f}]"
    ])
    plt.tight_layout()
    plt.show()
```

```python
def update(zx, zy, zz):
    z_vec = np.array([zx, zy, zz])
    if np.linalg.norm(z_vec) < 1e-6:
        print("Z norm is too small")
        return
    x, y, z = orthonormal_basis_from_z(z_vec)
    plot_basis(x, y, z)
```

```python
interact(
    update,
    zx=FloatSlider(min=-1, max=1, step=0.1, value=0.0, description='z_x'),
    zy=FloatSlider(min=-1, max=1, step=0.1, value=0.0, description='z_y'),
    zz=FloatSlider(min=-1, max=1, step=0.1, value=1.0, description='z_z'),
)
```

```
interactive(children=(FloatSlider(value=0.0, description='z_x', max=1.0, min=-
```

```
1.0), FloatSlider(value=0.0, des…
```

```
<function __main__.update(zx, zy, zz)>
```