Camproject

Release 0.42

Martin Israel

Feb 21, 2022

CONTENTS:

Camproject is a Python library for camera projections and reprojections. If you take a photo of a scene and you know the coordinates and orientation of the camera you can calculate the pixel position from a 3D scene object (projection) or - if you know the distance between camera and 3D scene object - you can invert this process and calculate the 3D coordinates from the pixel position (reprojection).

Check out the *Introduction* section for further information. *Quick Start* gives you the most important commands to work with this library.

Note: This project is under active development.

CHAPTER

ONE

CONTENTS

1.1 Introduction

1.1.1 What is the python camproject module?

camproject is a python module that provides functionality for projection from a 3D-scene to the 2D image plane of a camera. It also provides functionality for the reprojection from the 2d image plane to the scene in the 3D world coordinates. It is commonly required in engineering and science applications for georeferencing images.

1.1.2 How it works?

The camera geometry

Lets say, you take a photo of a scene with your camera. Your camera has a lens and a focal plane array (and a lot of other stuff we don't care about). `camera_example` shows how a real world point P(X,Y,Z) is being projected through the center of the lens on to the image pixel coordinates (u,v) of the camera's focal plane array. The optical axis pierces the center of the lens and hits the focal plane in the principle point (cx,cy).

The image on the focal plane array is upside down. This leads to the fact that the axes of the image coordinates (u,v) point always in the reverse direction of the world (respectively the camera) coordinates.

To get not that confused with the orientation, the computer vision people always invert the image coordinate system and move the image plane at the same distance (f) in front of the lens. Technically i think it is impossible to realize such a camera, but from the mathematical point of view this leads to the same solution. `pinhole_cameramodel` shows the simplified model.

Now the axes u and v point in the same direction as Xc and Yc. Zc points into the scene. The center of the lens is always the origin of the camera coordinate system. And we have a right sided coordinate system (left sided are used e.g. in geodetic applications).

Please have a look at the Open CV Camera Calibration Documentation The following documentation extends the OpenCV Docs or writes the same content in different words.

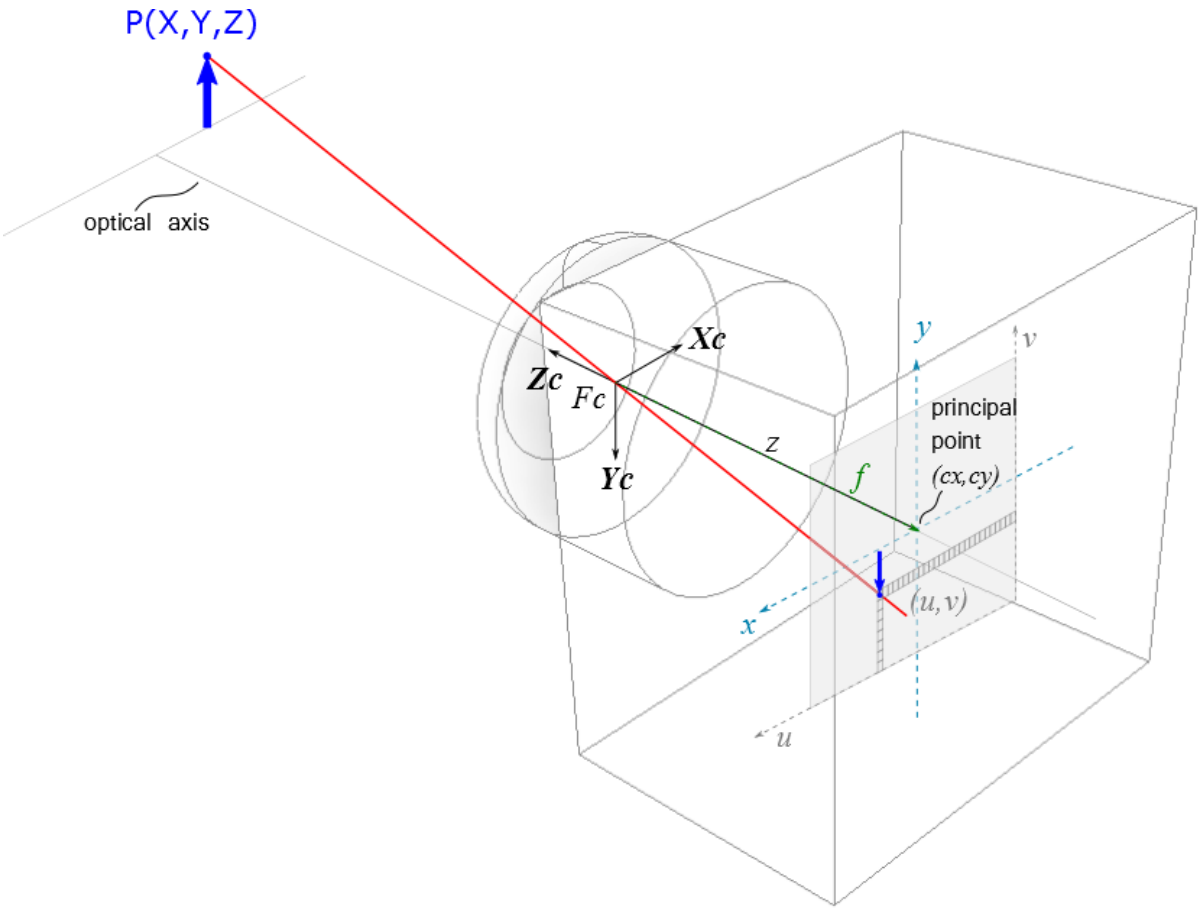

Fig. 1: the projection through a camera

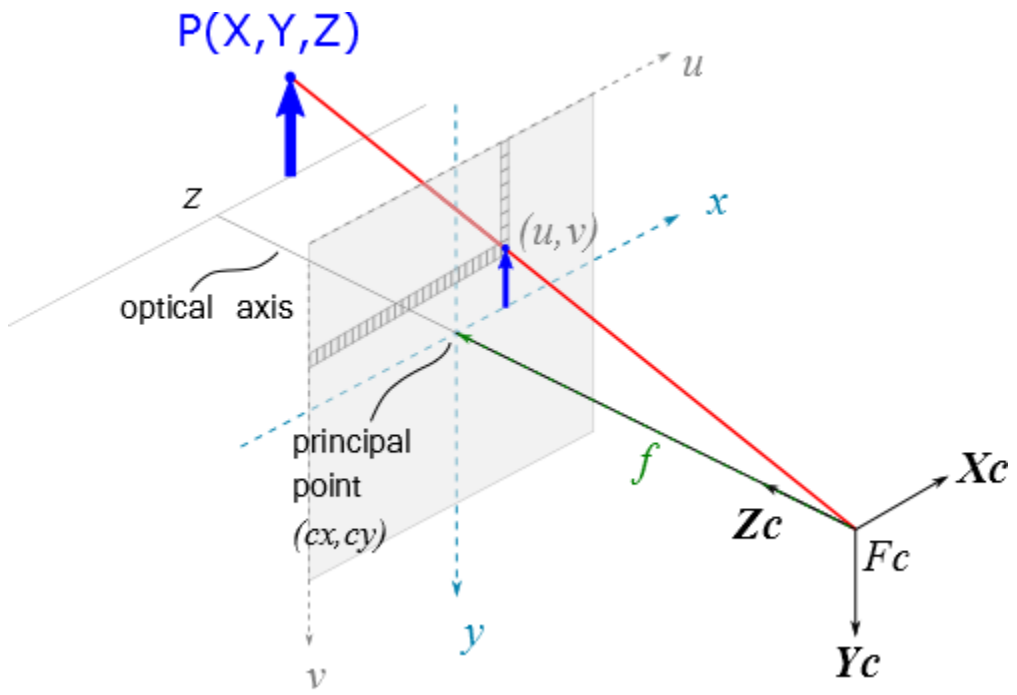

Fig. 2: simplified camera projection model (this image is based on an illustration from the openCV-Documentation)

The pinhole camera model

The most simple camera model is the pinhole camera. It consists of a light-tight hollow body with a very small pinhole and a lightsensitive film or an image detector. Due to the fact that it has no lens there exists no geometric distortion or blurring of unfocused objects. .. The pinhole camera can be used as a first order approximation of the mapping from a 3D scene to the image of a real camera.

From the mathematical point of view, the pinhole camera is simply a central projection from 3D to a 2D plane. The projection distance is the focal length of the camera.

With the aid of homogenious coordinates, projective transformations like the central projection are much easier to describe. The projection of a 3D point $\mathbf{X} \in \mathbb{R}^3$ onto the image plane of a pinhole camera can be described by the equation

$$\bar{\mathbf{x}} = \mathbf{P}\bar{\mathbf{X}}.$$

The 3D point is expressed by the homogenious vector $\bar{\mathbf{X}} = [X, Y, Z, W]^T \in \mathbb{P}^3$, while X, Y and Z are the same as from our real world $\mathbf{X} \in \mathbb{R}^3$ and W you can easily set to 1. The resulting image vector $\bar{\mathbf{x}}$ has the projective coordinates $[x, y, w]^T$. To get the pixel coordinates $[u, v, 1]^T$ you have to devide $\bar{\mathbf{x}}$ by its third component w. $\mathbf{P}$ is a 3×4 projection matrix with

$$\mathbf{P} = \mathbf{K}\left[\mathbf{R^T} | -\mathbf{R^T}\mathbf{T}\right].$$

The rotation matrix $\mathbf{R}$ and the translation vector $\mathbf{T} \in \mathbb{R}^3$ are the euclidean transformation between the camera and the world coordinate system. We call these parameters the extrinsic camera parameters (or outer orientation). The camera calibration matrix (or inner orientation)

$$\mathbf{K} = \begin{bmatrix} f_x & s_{xy} & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

holds the intrinsic parameters of the camera. f_x and f_y are the focal distances, with $f_y = a_r \cdot f_x$. Usually the aspect ratio a_r is 1. When you now think: how could there be two focal distances for one lens? The answer is: when your detector elements are not quadratic ($a_r \neq 1$), then you can use the dectector element size in x or y direction as unit to measure the focal distance. When your focal plane array is sheared you need to set s_{xy} different to 1. The central point of the camera is at $[c_x, c_y]^T$ (in pixels).

The whole Equation is then:

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} f_x & s_{xy} & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

The image pixel coordinates (u,v) are

$$u = \frac{x}{w}, \qquad v = \frac{y}{w}.$$

Brown's Camera Model

Until now we have ignored the distortion of the lens, but real camera lenses do have distortion. The brown camera model considers radial and tangential lens distortions.

$$\begin{aligned} r &= \sqrt{x^2 + y^2} \\ \hat{x} &= x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + p_2(r^2 + 2x^2) + 2p_1 xy \\ \hat{y} &= y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + p_1(r^2 + 2y^2) + 2p_2 xy \\ u &= c_x + \hat{x} f_x + \hat{y} s_{xy} \\ v &= c_y + \hat{y} f_y \end{aligned}$$

1.2 Quick Start

1.2.1 Projection

this library has two main methods:

```
project()
reproject()
```

to use this library you need just a few lines of code. This projects the Point P(1,0,10) on to the camera image plane. We call the projected point p(u,v)

```
>>>import numpy as np
>>>import camproject

>>>P = np.array([1,0,10,1])
>>>cam = camproject.Camera()
>>>cam.intrinsics(640,512,1000,320,260)
>>>cam.attitudeMat(np.eye(4))
>>>p = cam.project(P)
>>>print(p)
    [420 260]
```

With cam.intrinsics you define the most important inner parameters of the camera. so 640 and 512 means that the image plane has a width of 640 and a height of 512 pixels. The third parameter describes the focal distance f_{px} in pixels.

$$f_{px} = \frac{f_{mm}}{s}, \qquad s = \frac{w_{mm}}{w_{px}}$$

where w_{mm} is the camera sensor width in mm, w_{px} is the image width in pixels and s is the image detector element size. The last two parameters describe the center pixels [cx cy] where the optical axis hits the image plane.

Until now we don't have rotated or moved the camera somewhere, so with cam.attitudeMat(np.eye(4)) it is positioned at the coordinates origin and the lens points to the positive z-axis. Later you will see how to change the camera orientation.

Our point P is directly centered in front of the camera with a distance of 10 units (for example meters). The point p will be projected exactly to the optical center pixels. In our example this is [320 260].

1.2.2 Reprojection

To reproject the point back to the 3D world we use this code

```
>>>Q = cam.reprojectToPlane(p)
>>>print(Q)
    [-0. -0. -0.  1.]
```

The default plane is the xy-plane (z=0). that makes not much sense when the camera itself also is at these coordinates. Thats why we get [0 0 0 1]. So if we want to reproject the point to its real origin, we need a little more information, for example the z-coordinate of the point. So we could define a plane with z=10. For our plane parameter we need to write this: [0,0,1,-10]. The first three elements define the normal vector of the plane and the last element the negative distance. Then our reprojection code is

```
>>>plane = np.array([0,0,1,-10])
>>>Q = cam.reprojectToPlane(p,plane)
>>>print(Q)
    [ 1. -0. 10.  1.]
```

1.2.3 Multiple points

You can also project or reproject multiple points.

```
>>>PM = np.array([[2,0,10],[1,0,10],[0,0,10],[-1,0,10]])
>>>pm = cam.project(PM)
>>>print(pm)
    [[520. 256.]
    [420. 256.]
    [320. 256.]
    [220. 256.]]
>>>Q = cam.reprojectToPlane(pm,plane)
>>>print(np.around(Q,2))
    [[ 2. -0. 10.  1.]
    [ 1. -0. 10.  1.]
    [-0. -0. 10.  1.]
    [-1. -0. 10.  1.]]
```

If this easy reprojection does not fit your needs, you can use reproject(p) which returns a direction vector and write your own reprojection wrapper.

Note: reprojectToPlane returns a 3D Vector [X Y Z 1] in homogeneous coordinates. They are normalized, so the last element is always 1. You can use just the x,y,z-coordinates with Q[0:3]

1.3 Extrinsic Orientation

The extrinsics submodule is for rotating and translating the camera. It handles all the neccesary coordinate transformations.

We use our cameras on UAVs. So we additionally have a gimbal. The coordinate systems of the UAV and the gimbal are different to the cameras coordinate system. In the camera coordinate system z points through the lens. It is a right hand sided coordinate system. X describes the width and y the height of the image and [0,0] is not where the optical axis goes through the image plane. It is on the upper left corner of the image. The UAV coordinate system is normally right hand, with x pointing in front direction, y is pointing to the right side and z is pointing down.

Our real world geodetic coordinate system is a left handed one. Z points up and the compas orientation turns right. 0 degree is north, 90 degree is east, 180 degree is south and so on. So if you want to use real altitude above sea level values and normal compass orientation, you have to use the left handed coordinate system, where x points north and y points east and z to the sky.

But now we should just look at the code to rotate or translate the camera

```
ext = camproject.Extrinsics()
ext.setPose(X=0,Y=2,Z=10)
ext.setGimbal(roll=0,pitch=-90,yaw=0)
print(np.around(ext.transform(),2))
    [[ 0.  1.  0. -2.]
    [-1.  0. -0.  0.]
    [ 0.  0. -1. 10.]
    [ 0.  0.  0.  1.]]
```

In this example we position the UAV at postion [0,2,10] and let the camera point down (nadir). With ext.transform() we generate a 4 by 4 rotation and translation matrix. the rounding function np.around is just to get easy readable values, else you have very long float numbers with a lot of zeros.

This matrix can be used to set our Camera attitude matrix

```
cam.attitudeMat(ext.transform())
```

With this we reposition and reorientate the camera in our scene. So whenever you make a new picture with your drone and have new coordinates use ext.setPosition() and/or ext.setGimbal() and set cam.attitudeMat(ext.transform()).

Okay, back to the code. setPose is clear, X is the geodetic north direction, Y is geodetic east and Z is any sky pointing altitude. I prefer the barometric altitude, which starts with 0 at starting position. But never use latitude and longitude for Y and X, 'coz these are not orthogonal to each other. Convert them to UTM or another orthogonal system. Roll, pitch and yaw are in dregree (0 to 360). have alook at https://en.wikipedia.org/wiki/Aircraft_principal_axes for the uav orientations.

If you wonder why there is not a diagonal matrix, like in the quickstart example. It is due to the coordinate system transformation. Using the extrinsics module allows you to set in coordinates and orientations from a left hand coordinate system.

1.4 API Reference

1.4.1 CamModel

1.4.2 Camera

1.4.3 Extrinsics

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search