cameramodels Documentation

Iori Yanokura

May 11, 2023

CONTENTS

CHAPTER

ONE

API REFERENCE

1.1 PinholeCamera

1.1.1 PinholeCameraModel class

cameramodels.PinholeCameraModel	A Pinhole Camera Model

cameramodels.PinholeCameraModel

class cameramodels.**PinholeCameraModel**(*image_height*, *image_width*, *K*, *P*, *R=array([[1., 0., 0.], [0., 1., 0.], [0., 0., 1.]])*, *D=array([0., 0., 0., 0., 0.])*, *roi=None*, *tf_frame=None*, *stamp=None*, *distortion_model='plumb_bob'*, *name=''*, *full_K=None*, *full_P=None*, *full_height=None*, *full_width=None*, *binning_x=1*, *binning_y=1*, *target_size=None*)

A Pinhole Camera Model

more detail, see http://wiki.ros.org/image_pipeline/CameraInfo http://docs.ros.org/api/sensor_msgs/html/msg/CameraInfo.html

Parameters

- **image_height** (*int*) – height of camera image.
- **image_width** (*int*) – width of camera image.
- **K** (*numpy.ndarray*) – 3x3 intrinsic matrix.
- **P** (*numpy.ndarray*) – 3x4 projection matrix
- **R** (*numpy.ndarray*) – 3x3 rectification matrix.
- **D** (*numpy.ndarray*) – distortion.
- **roi** (*None or list[float]*) – [top, left, bottom, right] order.
- **tf_frame** (*None or str*) – tf frame. This is for ROS compatibility.
- **stamp** (*None*) – timestamp. This is for ROS compatibility.
- **distortion_model** (*str*) – type of distortion model.
- **name** (*None or str*) – name of this camera.
- **full_K** (*numpy.ndarray or None*) – original intrinsic matrix of full resolution. If *None*, set copy of K.

- **full_P** (*numpy.ndarray or None*) – original projection matrix of full resolution. If *None*, set copy of P.
- **full_height** (*int or None*) – This value is indicating original image height.
- **full_width** (*int or None*) – This value is indicating original image width.

Methods

batch_project3d_to_pixel(*points*, *project_valid_depth_only=False*, *return_indices=False*)

Return project uv coordinates points

Returns the rectified pixel coordinates (u, v) of the 3D points using the camera P matrix. This is the inverse of *batch_project_pixel_to_3d_ray()*.

Parameters

- **points** (*numpy.ndarray*) – batch of xyz point (batch_size, 3)
- **project_valid_depth_only** (*bool*) – If True, return uvs which are in frame.
- **return_indices** (*bool*) – If this value and project_valid_depth_only are True, return valid indices.

Returns
points – shape of (batch_size, 2).

Return type
numpy.ndarray

batch_project_pixel_to_3d_ray(*uv*, *depth=None*)

Returns the ray vectors

This function is the batch version of *project_pixel_to_3d_ray()*. Returns the unit vector which passes from the camera center to through rectified pixel (u, v), using the camera P matrix. This is the inverse of *batch_project3d_to_pixel()*. If depth is specified, return 3d points.

Parameters

- **uv** (*numpy.ndarray*) – rectified pixel coordinates
- **depth** (*None or numpy.ndarray*) – depth value. If this value is specified, Return 3d points.

Returns
ret – calculated ray vectors or points(depth is given case). Shape of (batch_size, 3)

Return type
numpy.ndarray

static calc_f_from_fov(*fov*, *aperture*)

Return focal length.

Parameters

- **fov** (*float*) – field of view in degree.
- **aperture** (*float*) – aperture.

Returns
focal_length – calculated focal length.

Return type
float

static calc_fovx(*fovy*, *height*, *width*)

Return fovx from fovy, height and width.

Parameters

- **fovy** (*float*) – field of view in degree.
- **height** (*int*) – height of camera.
- **width** (*int*) – width of camera.

Returns
fovx – calculated fovx.

Return type
float

static calc_fovy(*fovx*, *height*, *width*)

Return fovy from fovx, height and width.

Parameters

- **fovx** (*float*) – horizontal field of view in degree.
- **height** (*int*) – height of camera.
- **width** (*int*) – width of camera.

Returns
fovy – calculated fovy.

Return type
float

crop_image(*img*, *copy=False*)

Crop input full resolution image considering roi.

Note that this function will not return resized image.

Parameters

- **img** (*numpy.ndarray*) – input image. (H, W, channel)
- **copy** (*bool*) – if *True*, return copy image.

Returns
cropped_img – cropped image.

Return type
numpy.ndarray

crop_resize_camera_info(*target_size*, *roi=None*)

Return cropped and resized region's camera model

Parameters

- **target_size** (*list[int]*) – [target_width, target_height] order.
- **roi** (*list[float]*) – [top, left, bottom, right] order, by default self.roi.

Returns
cameramodel – camera model of cropped and resised region.

Return type
cameramodels.PinholeCameraModel

crop_resize_image(*img*, *interpolation=2*, *use_cv2=True*)

Crop and resize input full resolution image.

Parameters

- **img** (*numpy.ndarray*) – input image. (H, W, channel)
- **interpolation** (*int*) – interpolation method. You can specify, PIL.Image.NEAREST, PIL.Image.BILINEAR, PIL.Image.BICUBIC and PIL.Image.LANCZOS.

Returns
out – cropped and resized image.

Return type
numpy.ndarray

depth_to_points(*depth*)

Convert depth image to point clouds.

Parameters
depth (*numpy.ndarray*) – depth image.

Returns
points – return shape is (width, height, 3).

Return type
numpy.ndarray

draw_roi(*bgr_img*, *color=(46, 204, 113)*, *box_width=None*, *copy=False*)

Draw Region of Interest

Parameters

- **bgr_img** (*numpy.ndarray*) – input image.
- **color** (*tuple(int)*) – RGB order color.
- **box_width** (*None or int*) – box width. If *None*, automatically set from image size.
- **copy** (*bool*) – If *True*, return copy image. If input image is gray image, this option will be ignored.

Returns
img – ROI drawn image.

Return type
numpy.ndarray

dump(*output_filepath*, *save_original=True*)

Dump this camera's parameter to yaml file.

Parameters

- **output_filepath** (*str or pathlib.Path*) – output path
- **save_original** (*bool*) – If *False*, save resized camera info.

flatten_uv(*uv*, *dtype=<class 'numpy.int64'>*)

Flattens uv coordinates to single dimensional tensor.

This is the inverse of *flattened_pixel_locations_to_uv()*.

Parameters

- **uv** (*numpy.ndarray or list[tuple(float, float)]*) – A pair of uv pixels. Shape of (batch_size, 2). [(u_1, v_1), (u_2, v_2) . . . , (u_n, v_n)].
- **dtype** (*type*) – data type. default is numpy.int64.

Returns
ret – Flattened uv tensor of shape (n,).

Return type
numpy.ndarray

Examples

```
>>> from cameramodels import PinholeCameraModel
>>> cm = PinholeCameraModel.from_fovy(45, 480, 640)
>>> cm.flatten_uv(np.array([(1, 0), (100, 1), (100, 2)]))
array([   1,  740, 1380])
```

flattened_pixel_locations_to_uv(*flat_pixel_locations*)

Flattens pixel locations(single dimension tensor) to uv coordinates.

This is the inverse of *flatten_uv()*.

Parameters
flat_pixel_locations (*numpy.ndarray or list[float]*) – Flattened pixel locations.

Returns
ret – UV coordinates.

Return type
numpy.ndarray

Examples

```
>>> from cameramodels import PinholeCameraModel
>>> cm = PinholeCameraModel.from_fovy(45, 480, 640)
>>> flatten_uv = [1, 740, 1380]
>>> cm.flattened_pixel_locations_to_uv(flatten_uv)
array([[  1,   0],
       [100,   1],
       [100,   2]])
```

static from_camera_info(*camera_info_msg*)

Return PinholeCameraModel from camera_info_msg

Parameters
camera_info_msg (*sensor_msgs.msg.CameraInfo*) – message of camera info.

Returns
cameramodel – camera model

Return type
cameramodels.PinholeCameraModel

static from_fov(*fovy*, *height*, *width*, ***kwargs*)

Return PinholeCameraModel from fovy.

Parameters

- **fovy** (*float*) – vertical field of view in degree.
- **height** (*int*) – height of camera.
- **width** (*int*) – width of camera.

Returns
cameramodel – camera model

Return type
cameramodels.PinholeCameraModel

static from_fovx(*fovx*, *height*, *width*, ***kwargs*)

Return PinholeCameraModel from fovx.

Parameters

- **fovx** (*float*) – horizontal field of view in degree.
- **height** (*int*) – height of camera.
- **width** (*int*) – width of camera.

Returns
cameramodel – camera model

Return type
cameramodels.PinholeCameraModel

static from_fovy(*fovy*, *height*, *width*, ***kwargs*)

Return PinholeCameraModel from fovy.

Parameters

- **fovy** (*float*) – vertical field of view in degree.
- **height** (*int*) – height of camera.
- **width** (*int*) – width of camera.

Returns
cameramodel – camera model

Return type
cameramodels.PinholeCameraModel

static from_intrinsic_matrix(*intrinsic_matrix*, *height*, *width*, ***kwargs*)

Return PinholeCameraModel from intrinsic_matrix.

Parameters

- **intrinsic_matrix** (*numpy.ndarray*) – [3, 3] intrinsic matrix.
- **height** (*int*) – height of camera.
- **width** (*int*) – width of camera.
- **kwargs** (*dict*) – keyword args. These values are passed to cameramodels.PinholeCameraModel

Returns
cameramodel – camera model

Return type
cameramodels.PinholeCameraModel

`static from_open3d_intrinsic`(*open3d_pinhole_intrinsic*)

Return PinholeCameraModel from open3d's pinhole camera intrinsic.

Parameters
open3d_pinhole_intrinsic (*open3d.camera.PinholeCameraIntrinsic*) – open3d PinholeCameraIntrinsic

Returns
cameramodel – camera model

Return type
cameramodels.PinholeCameraModel

`static from_yaml_file`(*filename*)

Create instance of PinholeCameraModel from yaml file.

This function is supporting OpenCV calibration program's YAML format and sensor_msgs/CameraInfo's YAML format in ROS.

Parameters
filename (*str*) – path of yaml file.

Returns
cameramodel – camera model

Return type
cameramodels.PinholeCameraModel

`get_camera_matrix`()

Return camera matrix

Returns
camera_matrix – camera matrix from Projection matrix.

Return type
numpy.ndarray

`get_view_frustum`(*max_depth=1.0*, *translation=array([0., 0., 0.])*, *rotation=array([[1., 0., 0.], [0., 1., 0.], [0., 0., 1.]])*)

Return View Frustsum of this camera model.

Parameters

- **max_depth** (*float*) – max depth of frustsum.
- **translation** (*numpy.ndarray*) – translation vector
- **rotation** (*numpy.ndarray*) – rotation matrix

Returns
view_frust_pts – view frust points shape of (5, 3).

Return type
numpy.ndarray

Examples

```
>>> from cameramodels import Xtion
>>> cameramodel = Xtion()
>>> cameramodel.get_view_frustum(max_depth=1.0)
array([[ 0.        ,  0.        ,  0.        ],
       [-0.41421356, -0.41421356,  1.        ],
       [-0.41421356,  0.41421356,  1.        ],
       [ 0.41421356,  0.41421356,  1.        ],
       [ 0.41421356, -0.41421356,  1.        ]])
```

in_view_frustum(*points*, *max_depth=100.0*)

Determine if points in the view frustum.

Parameters

- **points** (*numpy.ndarray or list[tuple(float, float)]*) – 3D point (x, y, z).
- **max_depth** (*float*) – max depth of frustsum.

Returns

ret – bool array. True indicates point in this view frustsum.

Return type

numpy.ndarray or bool

points_in_roi(*points*)

Check if input points are in roi.

Parameters

points (*list[list[float, float]]*) – [[x_1, y_1], [x_2, y_2] . . . , [x_n, y_n]].

Returns

True if the point is in roi, False otherwise.

Return type

result list[bool]

points_to_depth(*points*, *depth_value=0.0*)

Return depth image from 3D points.

Parameters

- **points** (*numpy.ndarray*) – batch of xyz point (batch_size, 3) or (height, width, 3).
- **depth_value** (*float*) – default depth value.

Returns

depth – projected depth image.

Return type

numpy.ndarray

project3d_to_pixel(*point*)

Returns the rectified pixel coordinates

Returns the rectified pixel coordinates (u, v) of the 3D point, using the camera P matrix. This is the inverse of :meth *project_pixel_to_3d_ray*.

Parameters

point (*numpy.ndarray*) – 3D point (x, y, z)

Returns
uv – uv coordinates. If point is not in range of this camera model, return tuple(float('nan'), float('nan')).

Return type
tuple(float)

`project_pixel_to_3d_ray`(*uv*, *normalize=False*)

Returns the ray vector

Returns the unit vector which passes from the camera center to through rectified pixel (u, v), using the camera P matrix. This is the inverse of `project3d_to_pixel()`.

Parameters

- **uv** (*numpy.ndarray*) – rectified pixel coordinates
- **normalize** (*bool*) – if True, return normalized ray vector (unit vector).

Returns
ray_vector – ray vector.

Return type
tuple(float)

`rectify_image`(*raw_img*, *interpolation=2*)

Rectify input raw image.

Parameters

- **raw_img** (*numpy.ndarray*) – raw image.
- **interpolation** (*int*) – interpolation method. You can specify, PIL.Image.NEAREST, PIL.Image.BILINEAR, PIL.Image.BICUBIC and PIL.Image.LANCZOS.

Returns
rectified_img – rectified image.

Return type
numpy.ndarray

`rectify_point`(*uv_raw*)

Rectify input raw points.

Parameters
uv_raw (*numpy.ndarray or tuple[float] or list[float]*) – raw uv points.

Returns
rectified_uv – rectified point.

Return type
numpy.ndarray

`resize_bbox`(*bbox*)

Resize input full resolution bbox.

Parameters
bbox (*numpy.ndarray or list[float]*) – input bbox. Input shape can be (4,) or (N, 4). [top, left, bottom, right] order.

Returns
out_bbox – resized bbox.

Return type
numpy.ndarray

resize_point(*uv_point*)

Resize input full resolution uv point.

Parameters
uv_point (*numpy.ndarray or list[float]*) – input point. Input shape can be (2,) or (N, 2). [u, v] order.

Returns
out_point – resized point.

Return type
numpy.ndarray

unrectify_point(*uv_points*)

Return distorted points from rectified points.

Parameters
uv_points (*numpy.ndarray or list*) – (u, v) point.

Returns
distorted_points – distorted points.

Return type
numpy.ndarray or tuple(int)

__eq__(*value*, /)

Return self==value.

__ne__(*value*, /)

Return self!=value.

__lt__(*value*, /)

Return self<value.

__le__(*value*, /)

Return self<=value.

__gt__(*value*, /)

Return self>value.

__ge__(*value*, /)

Return self>=value.

Attributes

D

Property of distortion parameters

The distortion parameters, size depending on the distortion model. For "plumb_bob", the 5 parameters are: (k1, k2, t1, t2, k3).

Returns
self._D – distortion array.

Return type
numpy.ndarray

K

Intrinsic camera matrix for the raw (distorted) images.

$$K = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}$$

Projects 3D points in the camera coordinate frame to 2D pixel coordinates using the focal lengths (fx, fy) and principal point (cx, cy).

Returns
self._K – 3x3 intrinsic matrix.

Return type
numpy.ndarray

K_inv

Inverse of Intrinsic camera matrix for the raw (distorted) images.

$$K^{-1} = \begin{pmatrix} 1/f_x & 0 & -c_x/f_x \\ 0 & 1/f_y & -c_y/f_y \\ 0 & 0 & 1 \end{pmatrix}$$

Projects 3D points in the camera coordinate frame to 2D pixel coordinates using the focal lengths (fx, fy) and principal point (cx, cy).

Returns
self._K – 3x3 intrinsic matrix.

Return type
numpy.ndarray

P

Projection camera_matrix

By convention, this matrix specifies the intrinsic (camera) matrix of the processed (rectified) image.

$$P = \begin{pmatrix} {f_x}' & 0 & {c_x}' & T_x \\ 0 & {f_y}' & {c_y}' & T_y \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Returns
self._P – 4x3 projection matrix.

Return type
numpy.ndarray

R

Rectification matrix (stereo cameras only)

A rotation matrix aligning the camera coordinate system to the ideal stereo image plane so that epipolar lines in both stereo images are parallel.

Returns
self._R – rectification matrix.

Return type
numpy.ndarray

Tx

Return Tx.

For monocular cameras, Tx = Ty = Tz = 0. For a stereo pair, the fourth column [Tx Ty Tz]' is related to the position of the optical center of the second camera in the first camera's frame.

Returns
Tx

Return type
numpy.float32

Ty

Return Ty.

For monocular cameras, Tx = Ty = Tz = 0. For a stereo pair, the fourth column [Tx Ty Tz]' is related to the position of the optical center of the second camera in the first camera's frame.

Returns
Ty

Return type
numpy.float32

Tz

Return Tz.

For monocular cameras, Tx = Ty = Tz = 0. For a stereo pair, the fourth column [Tx Ty Tz]' is related to the position of the optical center of the second camera in the first camera's frame.

Returns
Tz

Return type
numpy.float32

aspect

Return aspect ratio

Returns
self._aspect – ascpect ratio of this camera.

Return type
float

binning_x

Return number of pixels to decimate to one horizontally.

Returns
self._binning_x – binning x.

Return type
int

binning_y

Return number of pixels to decimate to one vertically.

Returns
self._binning_y – binning y.

Return type
int

cx

Returns x center

Returns
cx

Return type
numpy.float32

cy

Returns y center

Returns
cy

Return type
numpy.float32

fov

Property of fov.

Returns
fov – tuple of (fovx, fovy).

Return type
tuple(float)

fovx

Property of horizontal fov.

Returns
self._fovx – horizontal fov of this camera.

Return type
float

fovy

Property of vertical fov.

Returns
self._fovy – vertical fov of this camera.

Return type
float

full_K

Return the original camera matrix for full resolution

Returns
self.full_K – intrinsic matrix.

Return type
numpy.ndarray

full_P

Return the projection matrix for full resolution

Returns
self.full_P – projection matrix.

Return type
numpy.ndarray

fx

Returns x focal length

Returns
fx

Return type
numpy.float32

fy

Returns y focal length

Returns
fy

Return type
numpy.float32

height

Returns image height

Returns
self._height – image height

Return type
int

open3d_intrinsic

Return open3d.camera.PinholeCameraIntrinsic instance.

Returns
intrinsic – open3d PinholeCameraIntrinsic

Return type
open3d.camera.PinholeCameraIntrinsic

roi

Return roi

Returns
self._roi – [top, left, bottom, right] order.

Return type
None or list[float]

target_size

Return target_size

Returns
self._target_size – (width, height). If this value is *None*, target size is not specified.

Return type
None or tuple(int)

width

Returns image width

Returns
self._width – image width

Return type
int

1.2 StereoCamera

1.2.1 StereoCameraModel class

cameramodels.stereo_camera.StereoCameraModel	A Stereo Camera Model

cameramodels.stereo_camera.StereoCameraModel

class cameramodels.stereo_camera.**StereoCameraModel**(*left=None*, *right=None*)

A Stereo Camera Model

Parameters

- **left** (cameramodels.PinholeCameraModel) – left camera model.
- **right** (cameramodels.PinholeCameraModel) – right camera model.

Methods

static from_camera_info(*left_camera_info_msg*, *right_camera_info_msg*)

Return StereoCameraModel from camera info msgs.

Parameters

- **left_camera_info_msg** (*sensor_msgs.msg.CameraInfo*) – left message of camera info.
- **right_camera_info_msg** (*sensor_msgs.msg.CameraInfo*) – right message of camera info.

Returns
cameramodel – stereo camera model

Return type
cameramodels.StereoCameraModel

static from_yaml_file(*left_yaml*, *right_yaml*)

Create instance of StereoCameraModel from yaml file.

This function is supporting sensor_msgs/CameraInfo's YAML format in ROS.

left_depth_to_right_depth(*depth*)

Return right camera depth image from left camera depth image.

Parameters
depth (*numpy.ndarray*) – depth image in meters.

Returns
right_depth – right camera's depth image in meters.

Return type
numpy.ndarray

points_from_keypoints(*left_uv*, *right_uv*, *return_pixel_error=False*)

Calculate points from left and right camera's keypoints.

Parameters

- **left_uv** (*numpy.ndarray or tuple*) – (x, y) coordinates.
- **right_uv** (*numpy.ndarray or tuple*) – (x, y) coordinates.
- **return_pixel_error** (*bool*) – if *True*, teturns the distance between the reprojected point and the input point.

Returns

points – (x, y, z) points.

Return type

numpy.ndarray

update_q()

Update variable fields of reprojection matrix

From Springer Handbook of Robotics, p. 524:

$$P = \begin{pmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

$$P' = \begin{pmatrix} f_x & 0 & c'_x & f_x T_x \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

where primed parameters are from the left projection matrix, unprimed from the right.

```
[u   v 1]^T = P  * [x y z 1]^T
[(u-d) v 1]^T = P' * [x y z 1]^T
```

Combining the two equations above results in the following equation

```
[u v u-d 1]^T = [ Fx   0    Cx   0    ] * [ x y z 1]^T
                [ 0    Fy   Cy   0    ]
                [ Fx   0    Cx'  FxTx ]
                [ 0    0    1    0    ]
```

Subtracting the 3rd from from the first and inverting the expression results in the following equation.

```
[x y z 1]^T = Q * [u v d 1]^T
```

Where Q is defined as

$$Q = \begin{pmatrix} f_y T_x & 0 & 0 & -f_y c_x T_x \\ 0 & f_x T_x & 0 & -f_x c_y T_x \\ 0 & 0 & 0 & f_x f_y T_x \\ 0 & 0 & -f_y & f_y(c_x - c'_x) \end{pmatrix}$$

Using the assumption f_x = f_y Q can be simplified to the following. But for compatibility with stereo cameras with different focal lengths we will use the full Q matrix.

$$Q = \begin{pmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 0 & f_x \\ 0 & 0 & -1/T_x & (c_x - c'_x)/T_x \end{pmatrix}$$

$$Disparity = x_{left} - x_{right}$$

For compatibility with stereo cameras with different focal lengths we will use the full Q matrix.

__eq__(*value*, /)

Return self==value.

__ne__(*value*, /)

Return self!=value.

__lt__(*value*, /)

Return self<value.

__le__(*value*, /)

Return self<=value.

__gt__(*value*, /)

Return self>value.

__ge__(*value*, /)

Return self>=value.

Attributes

Q

Return Q matrix.

$$Q = \begin{pmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 0 & f_x \\ 0 & 0 & -1/T_x & (c_x - c'_x)/T_x \end{pmatrix}$$

Returns
self._Q – Q matrix.

Return type
numpy.ndarray

baseline

Return left to right baseline.

Currently assuming horizontal baseline

Returns
baseline – left to right translation.

Return type
float

left_camera

Getter of left camera.

Returns
self._left_camera – left camera model

Return type
cameramodels.PinholeCameraModel

right_camera

Getter of right camera.

Returns

self._right_camera – right camera model

Return type

cameramodels.PinholeCameraModel

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

C

INDEX

Symbols

A

B

C

D

F

G

H

I

K

L

M

O

P

Q

R

S

T

U

W