probreg Documentation

Release 0.3.6

neka-nat

Aug 21, 2023

CONTENTS:

1 Installation **3**

2 probreg package modules **5**

2.1 callbacks 5
2.2 cost_functions 6
2.3 cpd 6
2.4 features 9
2.5 filterreg 11
2.6 gauss_transform 13
2.7 gaussian_filtering 14
2.8 gmmtree 14
2.9 l2dist_regs 15
2.10 math_utils 17
2.11 se3_op 18
2.12 transformation 18
2.13 Module contents 20

3 Indices and tables **21**

Python Module Index **23**

Index **25**

CHAPTER ONE

INSTALLATION

We recommend to install probreg via pip:

```
$ pip install probreg
```

You can also install probreg from a Git repository:

```
$ git clone https://github.com/neka-nat/probreg.git --recurse
$ cd probreg
$ pip install -e .
```

CHAPTER

TWO

PROBREG PACKAGE MODULES

2.1 callbacks

probreg.callbacks.**asnumpy**(*x*)

class probreg.callbacks.**Plot2DCallback**(*source: ndarray*, *target: ndarray*, *save: bool = False*, *keep_window: bool = True*)

Bases: object

Display the 2D registration result of each iteration.

Parameters

- **source** (*numpy.ndarray*) – Source point cloud data.
- **target** (*numpy.ndarray*) – Target point cloud data.
- **save** (*bool, optional*) – If this flag is True, each iteration image is saved in a sequential number.

class probreg.callbacks.**Open3dVisualizerCallback**(*source: ndarray*, *target: ndarray*, *save: bool = False*, *keep_window: bool = True*, *fov: Optional[Any] = None*)

Bases: object

Display the 3D registration result of each iteration.

Parameters

- **source** (*numpy.ndarray*) – Source point cloud data.
- **target** (*numpy.ndarray*) – Target point cloud data.
- **save** (*bool, optional*) – If this flag is True, each iteration image is saved in a sequential number.
- **keep_window** (*bool, optional*) – If this flag is True, the drawing window blocks after registration is finished.
- **fov** – Field of view (degree).

2.2 cost_functions

class probreg.cost_functions.**CostFunction**(*tf_type: Type[*Transformation*]*)

Bases: object

abstract to_transformation(*theta: ndarray*)

abstract initial()

probreg.cost_functions.**compute_l2_dist**(*mu_source: ndarray*, *phi_source: ndarray*, *mu_target: ndarray*, *phi_target: ndarray*, *sigma: float*)

class probreg.cost_functions.**RigidCostFunction**

Bases: *CostFunction*

to_transformation(*theta: ndarray*) → *Transformation*

initial() → ndarray

class probreg.cost_functions.**TPSCostFunction**(*control_pts: ndarray*, *alpha: float = 1.0*, *beta: float = 0.1*)

Bases: *CostFunction*

to_transformation(*theta: ndarray*) → *Transformation*

initial() → ndarray

2.3 cpd

class probreg.cpd.**EstepResult**(*pt1*, *p1*, *px*, *n_p*)

Bases: tuple

property n_p

Alias for field number 3

property p1

Alias for field number 1

property pt1

Alias for field number 0

property px

Alias for field number 2

class probreg.cpd.**MstepResult**(*transformation*, *sigma2*, *q*)

Bases: tuple

Result of Maximization step.

transformation

Transformation from source to target.

Type

tf.Transformation

sigma2

Variance of Gaussian distribution.

Type
float

q

Result of likelihood.

Type
float

property q

Alias for field number 2

property sigma2

Alias for field number 1

property transformation

Alias for field number 0

class probreg.cpd.CoherentPointDrift(*source: Optional[ndarray] = None*, *use_cuda: bool = False*)

Bases: `object`

Coherent Point Drift algorithm. This is an abstract class. Based on this class, it is inherited by rigid, affine, nonrigid classes according to the type of transformation. In this class, Estimation step in EM algorithm is implemented and Maximazation step is implemented in the inherited classes.

Parameters

- **source** (*numpy.ndarray, optional*) – Source point cloud data.
- **use_cuda** (*bool, optional*) – Use CUDA.

set_source(*source: ndarray*) → None

set_callbacks(*callbacks: List[Callable]*) → None

expectation_step(*t_source: ndarray*, *target: ndarray*, *sigma2: float*, *w: float = 0.0*) → *EstepResult*

Expectation step for CPD

maximization_step(*target: ndarray*, *estep_res:* EstepResult, *sigma2_p: Optional[float] = None*) → Optional[*MstepResult*]

registration(*target: ndarray*, *w: float = 0.0*, *maxiter: int = 50*, *tol: float = 0.001*) → *MstepResult*

class probreg.cpd.RigidCPD(*source: Optional[ndarray] = None*, *update_scale: bool = True*, *tf_init_params: Dict = {}*, *use_cuda: bool = False*)

Bases: `CoherentPointDrift`

Coherent Point Drift for rigid transformation.

Parameters

- **source** (*numpy.ndarray, optional*) – Source point cloud data.
- **update_scale** (*bool, optional*) – If this flag is True, compute the scale parameter.
- **tf_init_params** (*dict, optional*) – Parameters to initialize transformation.
- **use_cuda** (*bool, optional*) – Use CUDA.

maximization_step(*target: ndarray*, *estep_res:* EstepResult, *sigma2_p: Optional[float] = None*) → *MstepResult*

class probreg.cpd.**AffineCPD**(*source: Optional[ndarray] = None*, *tf_init_params: Dict = {}*, *use_cuda: bool = False*)

Bases: `CoherentPointDrift`

Coherent Point Drift for affine transformation.

Parameters

- **source** (*numpy.ndarray, optional*) – Source point cloud data.
- **tf_init_params** (*dict, optional*) – Parameters to initialize transformation.
- **use_cuda** (*bool, optional*) – Use CUDA.

class probreg.cpd.**NonRigidCPD**(*source: Optional[ndarray] = None*, *beta: float = 2.0*, *lmd: float = 2.0*, *use_cuda: bool = False*)

Bases: `CoherentPointDrift`

Coherent Point Drift for nonrigid transformation.

Parameters

- **source** (*numpy.ndarray, optional*) – Source point cloud data.
- **beta** (*float, optional*) – Parameter of RBF kernel.
- **lmd** (*float, optional*) – Parameter for regularization term.
- **use_cuda** (*bool, optional*) – Use CUDA.

set_source(*source: ndarray*) → None

maximization_step(*target: ndarray*, *estep_res:* EstepResult, *sigma2_p: Optional[float] = None*) → *MstepResult*

class probreg.cpd.**ConstrainedNonRigidCPD**(*source: Optional[ndarray] = None*, *beta: float = 2.0*, *lmd: float = 2.0*, *alpha: float = 1e-08*, *use_cuda: bool = False*, *idx_source: Optional[ndarray] = None*, *idx_target: Optional[ndarray] = None*)

Bases: `CoherentPointDrift`

Extended Coherent Point Drift for nonrigid transformation. Like CoherentPointDrift, but allows to add point correspondance constraints See: https://people.mpi-inf.mpg.de/~golyanik/04_DRAFTS/ECPD2016.pdf

Parameters

- **source** (*numpy.ndarray, optional*) – Source point cloud data.
- **beta** (*float, optional*) – Parameter of RBF kernel.
- **lmd** (*float, optional*) – Parameter for regularization term.
- **alpha** (*float*) – Degree of reliability of priors. Approximately between 1e-8 (highly reliable) and 1 (highly unreliable)
- **use_cuda** (*bool, optional*) – Use CUDA.
- **idx_source** (*numpy.ndarray of ints, optional*) – Indices in source matrix for which a correspondance is known

- **idx_target** (*numpy.ndarray of ints, optional*) – Indices in target matrix for which a correspondance is known

set_source(*source: ndarray*) → None

maximization_step(*target: ndarray*, *estep_res: EstepResult*, *sigma2_p: Optional[float] = None*) → *MstepResult*

probreg.cpd.**registration_cpd**(*source: Union[ndarray, PointCloud]*, *target: Union[ndarray, PointCloud]*, *tf_type_name: str = 'rigid'*, *w: float = 0.0*, *maxiter: int = 50*, *tol: float = 0.001*, *callbacks: List[Callable] = []*, *use_cuda: bool = False*, ***kwargs: Any*) → *MstepResult*

CPD Registraion.

Parameters

- **source** (*numpy.ndarray*) – Source point cloud data.
- **target** (*numpy.ndarray*) – Target point cloud data.
- **tf_type_name** (*str, optional*) – Transformation type('rigid', 'affine', 'nonrigid', 'nonrigid_constrained')
- **w** (*float, optional*) – Weight of the uniform distribution, $0 < w < 1$.
- **maxitr** (*int, optional*) – Maximum number of iterations to EM algorithm.
- **tol** (*float, optional*) – Tolerance for termination.
- **callback** (list of function, optional) – Called after each iteration. *callback(probreg.Transformation)*
- **use_cuda** (*bool, optional*) – Use CUDA.

Keyword Arguments

- **update_scale** (*bool, optional*) – If this flag is true and tf_type is rigid transformation, then the scale is treated. The default is true.
- **tf_init_params** (*dict, optional*) – Parameters to initialize transformation (for rigid or affine).

Returns

Result of the registration (transformation, sigma2, q)

Return type

MstepResult

2.4 features

class probreg.features.**Feature**

Bases: object

abstract init()

abstract compute(*data*)

annealing()

class probreg.features.**FPFH**(*radius_normal: float = 0.1*, *radius_feature: float = 0.5*)

Bases: *Feature*

Fast Point Feature Histograms

Parameters

- **radius_normal** (*float*) – Radius search parameter for computing normal vectors
- **radius_feature** (*float*) – Radius search parameter for computing FPFH.

init()

estimate_normals(*pcd: PointCloud*)

compute(*data: ndarray*)

class probreg.features.**GMM**(*n_gmm_components: int = 800*)

Bases: *Feature*

Feature points extraction using Gaussian mixture model

Parameters

n_gmm_components (*int*) – The number of mixture components.

init()

compute(*data: ndarray*)

class probreg.features.**OneClassSVM**(*dim: int*, *sigma: float*, *gamma: float = 0.5*, *nu: float = 0.05*, *delta: float = 10.0*)

Bases: *Feature*

Feature points extraction using One class SVM

Parameters

- **dim** (*int*) – The dimension of samples.
- **sigma** (*float*) – Veriance of the gaussian distribution made from parameters of SVM.
- **gamma** (*float, optional*) – Coefficient for RBF kernel.
- **nu** (*float, optional*) – An upper bound on the fraction of training errors and a lower bound of the fraction of support vectors.
- **delta** (*float, optional*) – Anealing parameter for optimization.

init()

compute(*data: ndarray*)

annealing()

2.5 filterreg

`class probreg.filterreg.EstepResult`(*m0*, *m1*, *m2*, *nx*)

Bases: `tuple`

property m0

Alias for field number 0

property m1

Alias for field number 1

property m2

Alias for field number 2

property nx

Alias for field number 3

`class probreg.filterreg.MstepResult`(*transformation*, *sigma2*, *q*)

Bases: `tuple`

Result of Maximization step.

transformation

Transformation from source to target.

Type
tf.Transformation

sigma2

Variance of Gaussian distribution.

Type
float

q

Result of likelihood.

Type
float

property q

Alias for field number 2

property sigma2

Alias for field number 1

property transformation

Alias for field number 0

`probreg.filterreg.dualquat_from_twist`(*tw*)

`class probreg.filterreg.FilterReg`(*source=None*, *target_normals=None*, *sigma2=None*, *update_sigma2=False*)

Bases: `object`

FilterReg is similar to CPD, and the speed performance is improved. In this algorithm, not only point-to-point alignment but also point-to-plane alignment are implemented.

Parameters

- **source** (*numpy.ndarray, optional*) – Source point cloud data.
- **target_normals** (*numpy.ndarray, optional*) – Normals of target points.
- **sigma2** (*Float, optional*) – Variance parameter. If this variable is None, the variance is updated in Mstep.
- **update_sigma2** (*bool, optional*) – If this variable is True, Update sigma2 in the registration iteration.

set_source(*source*)

set_target_normals(*target_normals*)

set_callbacks(*callbacks*)

expectation_step(*t_source*, *target*, *y*, *sigma2*, *update_sigma2*, *objective_type='pt2pt'*, *alpha=0.015*)

Expectation step

maximization_step(*t_source*, *target*, *estep_res*, *w=0.0*, *objective_type='pt2pt'*)

registration(*target*, *w=0.0*, *objective_type='pt2pt'*, *maxiter=50*, *tol=0.001*, *min_sigma2=0.0001*, *feature_fn=<function FilterReg.<lambda>>*)

class probreg.filterreg.**RigidFilterReg**(*source=None*, *target_normals=None*, *sigma2=None*, *update_sigma2=False*, *tf_init_params={}*)

Bases: *FilterReg*

class probreg.filterreg.**DeformableKinematicFilterReg**(*source=None*, *skinning_weight=None*, *sigma2=None*)

Bases: *FilterReg*

probreg.filterreg.**registration_filterreg**(*source: ~typing.Union[~numpy.ndarray, ~open3d.cpu.pybind.geometry.PointCloud], target: ~typing.Union[~numpy.ndarray, ~open3d.cpu.pybind.geometry.PointCloud], target_normals: ~typing.Optional[~numpy.ndarray] = None, sigma2: ~typing.Optional[float] = None, update_sigma2: bool = False, w: float = 0, objective_type: str = 'pt2pt', maxiter: int = 50, tol: float = 0.001, min_sigma2: float = 0.0001, feature_fn: ~typing.Callable = <function <lambda>>, callbacks: ~typing.List[~typing.Callable] = [], **kwargs: ~typing.Any*)

FilterReg registration

Parameters

- **source** (*numpy.ndarray*) – Source point cloud data.
- **target** (*numpy.ndarray*) – Target point cloud data.
- **target_normals** (*numpy.ndarray, optional*) – Normal vectors of target point cloud.
- **sigma2** (*float, optional*) – Variance of GMM. If *sigma2* is *None*, *sigma2* is automatically updated.
- **w** (*float, optional*) – Weight of the uniform distribution, $0 < w < 1$.
- **objective_type** (*str, optional*) – The type of objective function selected by ‘pt2pt’ or ‘pt2pl’.
- **maxitr** (*int, optional*) – Maximum number of iterations to EM algorithm.

- **tol** (*float, optional*) – Tolerance for termination.
- **min_sigma2** (*float, optional*) – Minimum variance of GMM.
- **feature_fn** (*function, optional*) – Feature function. If you use FPFH feature, set *feature_fn=probreg.feature.FPFH()*.
- **callback** (list of function, optional) – Called after each iteration. *callback(probreg.Transformation)*

Keyword Arguments
tf_init_params (*dict, optional*) – Parameters to initialize transformation (for rigid).

Returns
Result of the registration (transformation, sigma2, q)

Return type
MstepResult

2.6 gauss_transform

class probreg.gauss_transform.**Direct**(*source*, *h*)

Bases: object

compute(*target: ndarray*, *weights: ndarray*) → ndarray

class probreg.gauss_transform.**GaussTransform**(*source: ndarray*, *h: float*, *eps: float = 0.0001*, *sw_h: float = 0.01*)

Bases: object

Calculate Gauss Transform

Parameters

- **source** (*numpy.ndarray*) – Source data.
- **h** (*float*) – Bandwidth parameter of the Gaussian.
- **eps** (*float*) – Small floating point used in Gauss Transform.
- **sw_h** (*float*) – Value of the bandwidth parameter to switch between direct method and IFGT.

compute(*target: ndarray*, *weights: Optional[ndarray] = None*)

Compute gauss transform

Parameters

- **target** (*numpy.ndarray*) – Target data.
- **weights** (*numpy.ndarray*) – Weights of Gauss Transform.

2.7 gaussian_filtering

class probreg.gaussian_filtering.**Permutohedral**(*p: ndarray*, *with_blur: bool = True*)

Bases: `object`

get_lattice_size()

filter(*v: ndarray*, *start: int = 0*)

2.8 gmmtree

class probreg.gmmtree.**EstepResult**(*moments*)

Bases: `tuple`

property moments

Alias for field number 0

class probreg.gmmtree.**MstepResult**(*transformation*, *q*)

Bases: `tuple`

Result of Maximization step.

transformation

Transformation from source to target.

Type

tf.Transformation

q

Result of likelihood.

Type

float

property q

Alias for field number 1

property transformation

Alias for field number 0

class probreg.gmmtree.**GMMTree**(*source: Optional[ndarray] = None*, *tree_level: int = 2*, *lambda_c: float = 0.01*, *lambda_s: float = 0.001*, *tf_init_params: Dict = {}*)

Bases: `object`

GMM Tree

Parameters

- **source** (*numpy.ndarray, optional*) – Source point cloud data.
- **tree_level** (*int, optional*) – Maximum depth level of GMM tree.
- **lambda_c** (*float, optional*) – Parameter that determine the pruning of GMM tree.
- **lambda_s** (*float, optional*) – Parameter that tolerance for building GMM tree.
- **tf_init_params** (*dict, optional*) – Parameters to initialize transformation.

set_source(*source: ndarray*) → None

set_callbacks(*callbacks*)

expectation_step(*target: ndarray*) → *EstepResult*

maximization_step(*estep_res:* EstepResult, *trans_p:* Transformation) → *MstepResult*

registration(*target: ndarray*, *maxiter: int = 20*, *tol: float = 0.0001*) → *MstepResult*

probreg.gmmtree.**registration_gmmtree**(*source: Union[ndarray, PointCloud]*, *target: Union[ndarray, PointCloud]*, *maxiter: int = 20*, *tol: float = 0.0001*, *callbacks: List[Callable] = []*, ***kwargs: Any*) → *MstepResult*

GMMTree registration

Parameters

- **source** (*numpy.ndarray*) – Source point cloud data.
- **target** (*numpy.ndarray*) – Target point cloud data.
- **maxitr** (*int, optional*) – Maximum number of iterations to EM algorithm.
- **tol** (*float, optional*) – Tolerance for termination.
- **callback** (list of function, optional) – Called after each iteration. *callback(probreg.Transformation)*

Keyword Arguments

- **tree_level** (*int, optional*) – Maximum depth level of GMM tree.
- **lambda_c** (*float, optional*) – Parameter that determine the pruning of GMM tree.
- **lambda_s** (*float, optional*) – Parameter that tolerance for building GMM tree.
- **tf_init_params** (*dict, optional*) – Parameters to initialize transformation.

Returns

Result of the registration (transformation, q)

Return type

MstepResult

2.9 l2dist_regs

class probreg.l2dist_regs.**L2DistRegistration**(*source: ndarray*, *feature_gen:* Feature, *cost_fn:* CostFunction, *sigma: float = 1.0*, *delta: float = 0.9*, *use_estimated_sigma: bool = True*)

Bases: `object`

L2 distance registration class This algorithm expresses point clouds as mixture gaussian distributions and performs registration by minimizing the distance between two distributions.

Parameters

- **source** (*numpy.ndarray*) – Source point cloud data.
- **feature_gen** (`probreg.features.Feature`) – Generator of mixture gaussian distribution.

- **cost_fn** (probreg.cost_functions.CostFunction) – Cost function to caliculate L2 distance.
- **sigma** (*float, optional*) – Scaling parameter for L2 distance.
- **delta** (*float, optional*) – Annealing parameter for optimization.
- **use_estimated_sigma** (*float, optional*) – If this flag is True, sigma estimates from the source point cloud.

set_source(*source: ndarray*)

set_callbacks(*callbacks*)

optimization_cb(*x: ndarray*)

registration(*target: ndarray*, *maxiter: int = 1*, *tol: float = 0.001*, *opt_maxiter: int = 50*, *opt_tol: float = 0.001*) → *Transformation*

class probreg.l2dist_regs.**RigidGMMReg**(*source*, *sigma=1.0*, *delta=0.9*, *n_gmm_components=800*, *use_estimated_sigma=True*)

Bases: *L2DistRegistration*

class probreg.l2dist_regs.**TPSGMMReg**(*source*, *sigma=1.0*, *delta=0.9*, *n_gmm_components=800*, *alpha=1.0*, *beta=0.1*, *use_estimated_sigma=True*)

Bases: *L2DistRegistration*

class probreg.l2dist_regs.**RigidSVR**(*source*, *sigma=1.0*, *delta=0.9*, *gamma=0.5*, *nu=0.1*, *use_estimated_sigma=True*)

Bases: *L2DistRegistration*

class probreg.l2dist_regs.**TPSSVR**(*source*, *sigma=1.0*, *delta=0.9*, *gamma=0.5*, *nu=0.1*, *alpha=1.0*, *beta=0.1*, *use_estimated_sigma=True*)

Bases: *L2DistRegistration*

probreg.l2dist_regs.**registration_gmmreg**(*source: ndarray*, *target: ndarray*, *tf_type_name: str = 'rigid'*, *callbacks: List = []*, ***kargs*)

GMMReg.

Parameters

- **source** (*numpy.ndarray*) – Source point cloud data.
- **target** (*numpy.ndarray*) – Target point cloud data.
- **tf_type_name** (*str, optional*) – Transformation type('rigid', 'nonrigid')
- **callback** (list of function, optional) – Called after each iteration. *callback(probreg.Transformation)*

Returns
Transformation from source to target.

Return type
probreg.Transformation

probreg.l2dist_regs.**registration_svr**(*source: Union[ndarray, PointCloud]*, *target: Union[ndarray, PointCloud]*, *tf_type_name: str = 'rigid'*, *maxiter: int = 1*, *tol: float = 0.001*, *opt_maxiter: int = 50*, *opt_tol: float = 0.001*, *callbacks: List[Callable] = []*, ***kwargs: Any*)

Support Vector Registration.

Parameters

- **source** (*numpy.ndarray*) – Source point cloud data.
- **target** (*numpy.ndarray*) – Target point cloud data.
- **tf_type_name** (*str, optional*) – Transformation type('rigid', 'nonrigid')
- **maxitr** (*int, optional*) – Maximum number of iterations for outer loop.
- **tol** (*float, optional*) – Tolerance for termination of outer loop.
- **opt_maxitr** (*int, optional*) – Maximum number of iterations for inner loop.
- **opt_tol** (*float, optional*) – Tolerance for termination of inner loop.
- **callback** (list of function, optional) – Called after each iteration. *callback(probreg.Transformation)*

Returns
Transformation from source to target.

Return type
probreg.Transformation

2.10 math_utils

class probreg.math_utils.**Normalizer**(*scale: float = 1.0*, *centroid: float = 0.0*)

Bases: object

Parameters

- **scale** (*float, optional*) – Scale factor.
- **centroid** (*numpy.array, optional*) – Central point.

normalize(*x: ndarray*) → ndarray

denormalize(*x: ndarray*) → ndarray

probreg.math_utils.**squared_kernel_sum**(*x: ndarray*, *y: ndarray*) → float

probreg.math_utils.**compute_rmse**(*source: ndarray*, *target_tree: cKDTree*) → float

probreg.math_utils.**rbf_kernel**(*x: ndarray*, *y: ndarray*, *beta: float*) → float

probreg.math_utils.**tps_kernel**(*x: ndarray*, *y: ndarray*) → float

probreg.math_utils.**inverse_multiquadric_kernel**(*x: ndarray*, *y: ndarray*, *c: float = 1.0*) → float

2.11 se3_op

probreg.se3_op.**skew**(*x: ndarray*) → ndarray

skew-symmetric matrix, that represent cross products as matrix multiplications.

Parameters
x (*numpy.ndarray*) – 3D vector.

Returns
3x3 skew-symmetric matrix.

probreg.se3_op.**twist_trans**(*tw: np.ndarray*, *linear: bool = False*) → tuple[np.ndarray, np.ndarray]

Convert from twist representation to transformation matrix.

Parameters

- **tw** (*numpy.ndarray*) – Twist vector.
- **linear** (*bool, optional*) – Linear approximation.

probreg.se3_op.**twist_mul**(*tw: np.ndarray*, *rot: np.ndarray*, *t: np.ndarray*, *linear: bool = False*) → tuple[np.ndarray, np.ndarray]

Multiply twist vector and transformation matrix.

Parameters

- **tw** (*numpy.ndarray*) – Twist vector.
- **rot** (*numpy.ndarray*) – Rotation matrix.
- **t** (*numpy.ndarray*) – Translation vector.
- **linear** (*bool, optional*) – Linear approximation.

probreg.se3_op.**diff_x_from_twist**(*x: ndarray*) → ndarray

probreg.se3_op.**diff_rot_from_quaternion**(*q: ndarray*) → ndarray

Differencial rotation matrix from quaternion.

dR(q)/dq = [dR(q)/dq0, dR(q)/dq1, dR(q)/dq2, dR(q)/dq3]

Parameters
q (*numpy.ndarray*) – Quaternion.

2.12 transformation

class probreg.transformation.**Transformation**(*xp=<module 'numpy' from '/home/docs/checkouts/readthedocs.org/user_builds/probreg/envs/latest/lib/py packages/numpy/__init__.py'>*)

Bases: object

transform(*points*, *array_type=<class 'open3d.cpu.pybind.utility.Vector3dVector'>*)

class probreg.transformation.**RigidTransformation**(*rot=array([[1., 0., 0.], [0., 1., 0.], [0., 0., 1.]])*, *t=array([0., 0., 0.])*, *scale=1.0*, *xp=<module 'numpy' from '/home/docs/checkouts/readthedocs.org/user_builds/probreg/envs/latest packages/numpy/__init__.py'>*)

Bases: `Transformation`

Rigid Transformation

Parameters

- **rot** (*numpy.ndarray, optional*) – Rotation matrix.
- **t** (*numpy.ndarray, optional*) – Translation vector.
- **scale** (*Float, optional*) – Scale factor.
- **xp** (*module, optional*) – Numpy or Cupy.

inverse()

class probreg.transformation.**AffineTransformation**(*b=array([[1., 0., 0.], [0., 1., 0.], [0., 0., 1.]]), t=array([0., 0., 0.]), xp=<module 'numpy' from '/home/docs/checkouts/readthedocs.org/user_builds/probreg/envs/late... packages/numpy/__init__.py'>*)

Bases: `Transformation`

Affine Transformation

Parameters

- **b** (*numpy.ndarray, optional*) – Affine matrix.
- **t** (*numpy.ndarray, optional*) – Translation vector.
- **xp** (*module, optional*) – Numpy or Cupy.

class probreg.transformation.**NonRigidTransformation**(*w, points, beta=2.0, xp=<module 'numpy' from '/home/docs/checkouts/readthedocs.org/user_builds/probreg/envs/la... packages/numpy/__init__.py'>*)

Bases: `Transformation`

Nonrigid Transformation

Parameters

- **w** (*numpy.array*) – Weights for kernel.
- **points** (*numpy.array*) – Source point cloud data.
- **beta** (*float, optional*) – Parameter for gaussian kernel.
- **xp** (*module*) – Numpy or Cupy.

class probreg.transformation.**CombinedTransformation**(*rot=array([[1.0, 0.0, 0.0], [0.0, 1.0, 0.0], [0.0, 0.0, 1.0]]), t=array([0.0, 0.0, 0.0]), scale=1.0, v=0.0*)

Bases: `Transformation`

Combined Transformation

Parameters

- **rot** (*numpy.array, optional*) – Rotation matrix.
- **t** (*numpy.array, optional*) – Translation vector.
- **scale** (*float, optional*) – Scale factor.
- **v** (*numpy.array, optional*) – Nonrigid term.

class probreg.transformation.**TPSTransformation**(*a*, *v*, *control_pts*, *kernel=<function tps_kernel>*)

Bases: *Transformation*

Thin Plate Spline transformaion.

Parameters

- **a** (*numpy.array*) – Affine matrix.
- **v** (*numpy.array*) – Translation vector.
- **control_pts** (*numpy.array*) – Control points.
- **kernel** (*function, optional*) – Kernel function.

prepare(*landmarks*)

transform_basis(*basis*)

class probreg.transformation.**DeformableKinematicModel**(*dualquats*, *weights*)

Bases: *Transformation*

Deformable Kinematic Transformation

Parameters

- **dualquats** (list of dq3d.dualquat) – Transformations for each link.
- **weights** (DeformableKinematicModel.SkinningWeight) – Skinning weight.

class SkinningWeight(*n_points*)

Bases: ndarray

Transformations and weights for each point.

. tf = SkinningWeight['val'][0] * dualquats[SkinningWeight['pair'][0]] + SkinningWeight['val'][1] * dualquats[SkinningWeight['pair'][1]]

property n_nodes

pairs_set()

in_pair(*pair*)

Return indices of the pairs equal to the given pair.

classmethod make_weight(*pairs*, *vals*)

2.13 Module contents

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

probreg, 20
probreg.callbacks, 5
probreg.cost_functions, 6
probreg.cpd, 6
probreg.features, 9
probreg.filterreg, 11
probreg.gauss_transform, 13
probreg.gaussian_filtering, 14
probreg.gmmtree, 14
probreg.l2dist_regs, 15
probreg.math_utils, 17
probreg.se3_op, 18
probreg.transformation, 18

INDEX

A

AffineCPD (*class in probreg.cpd*), 8
AffineTransformation (*class in probreg.transformation*), 19
annealing() (*probreg.features.Feature method*), 9
annealing() (*probreg.features.OneClassSVM method*), 10
asnumpy() (*in module probreg.callbacks*), 5

C

CoherentPointDrift (*class in probreg.cpd*), 7
CombinedTransformation (*class in probreg.transformation*), 19
compute() (*probreg.features.Feature method*), 9
compute() (*probreg.features.FPFH method*), 10
compute() (*probreg.features.GMM method*), 10
compute() (*probreg.features.OneClassSVM method*), 10
compute() (*probreg.gauss_transform.Direct method*), 13
compute() (*probreg.gauss_transform.GaussTransform method*), 13
compute_l2_dist() (*in module probreg.cost_functions*), 6
compute_rmse() (*in module probreg.math_utils*), 17
ConstrainedNonRigidCPD (*class in probreg.cpd*), 8
CostFunction (*class in probreg.cost_functions*), 6

D

DeformableKinematicFilterReg (*class in probreg.filterreg*), 12
DeformableKinematicModel (*class in probreg.transformation*), 20
DeformableKinematicModel.SkinningWeight (*class in probreg.transformation*), 20
denormalize() (*probreg.math_utils.Normalizer method*), 17
diff_rot_from_quaternion() (*in module probreg.se3_op*), 18
diff_x_from_twist() (*in module probreg.se3_op*), 18
Direct (*class in probreg.gauss_transform*), 13
dualquat_from_twist() (*in module probreg.filterreg*), 11

E

EstepResult (*class in probreg.cpd*), 6
EstepResult (*class in probreg.filterreg*), 11
EstepResult (*class in probreg.gmmtree*), 14
estimate_normals() (*probreg.features.FPFH method*), 10
expectation_step() (*probreg.cpd.CoherentPointDrift method*), 7
expectation_step() (*probreg.filterreg.FilterReg method*), 12
expectation_step() (*probreg.gmmtree.GMMTree method*), 15

F

Feature (*class in probreg.features*), 9
filter() (*probreg.gaussian_filtering.Permutohedral method*), 14
FilterReg (*class in probreg.filterreg*), 11
FPFH (*class in probreg.features*), 9

G

GaussTransform (*class in probreg.gauss_transform*), 13
get_lattice_size() (*probreg.gaussian_filtering.Permutohedral method*), 14
GMM (*class in probreg.features*), 10
GMMTree (*class in probreg.gmmtree*), 14

I

in_pair() (*probreg.transformation.DeformableKinematicModel.SkinningW method*), 20
init() (*probreg.features.Feature method*), 9
init() (*probreg.features.FPFH method*), 10
init() (*probreg.features.GMM method*), 10
init() (*probreg.features.OneClassSVM method*), 10
initial() (*probreg.cost_functions.CostFunction method*), 6
initial() (*probreg.cost_functions.RigidCostFunction method*), 6
initial() (*probreg.cost_functions.TPSCostFunction method*), 6

inverse() (probreg.transformation.RigidTransformation method), 19
inverse_multiquadric_kernel() (in module probreg.math_utils), 17

L

L2DistRegistration (class in probreg.l2dist_regs), 15

M

m0 (probreg.filterreg.EstepResult property), 11
m1 (probreg.filterreg.EstepResult property), 11
m2 (probreg.filterreg.EstepResult property), 11
make_weight() (probreg.transformation.DeformableKinematicModel class method), 20
maximization_step() (probreg.cpd.CoherentPointDrift method), 7
maximization_step() (probreg.cpd.ConstrainedNonRigidCPD method), 9
maximization_step() (probreg.cpd.NonRigidCPD method), 8
maximization_step() (probreg.cpd.RigidCPD method), 7
maximization_step() (probreg.filterreg.FilterReg method), 12
maximization_step() (probreg.gmmtree.GMMTree method), 15
module
  probreg, 20
  probreg.callbacks, 5
  probreg.cost_functions, 6
  probreg.cpd, 6
  probreg.features, 9
  probreg.filterreg, 11
  probreg.gauss_transform, 13
  probreg.gaussian_filtering, 14
  probreg.gmmtree, 14
  probreg.l2dist_regs, 15
  probreg.math_utils, 17
  probreg.se3_op, 18
  probreg.transformation, 18
moments (probreg.gmmtree.EstepResult property), 14
MstepResult (class in probreg.cpd), 6
MstepResult (class in probreg.filterreg), 11
MstepResult (class in probreg.gmmtree), 14

N

n_nodes (probreg.transformation.DeformableKinematicModel.SkinningWeight property), 20
n_p (probreg.cpd.EstepResult property), 6
NonRigidCPD (class in probreg.cpd), 8
NonRigidTransformation (class in probreg.transformation), 19
normalize() (probreg.math_utils.Normalizer method), 17
Normalizer (class in probreg.math_utils), 17
nx (probreg.filterreg.EstepResult property), 11

O

OneClassSVM (class in probreg.features), 10
Open3dVisualizerCallback (class in probreg.callbacks), 5
optimization_cb() (probreg.l2dist_regs.L2DistRegistration method), 16

P

p1 (probreg.cpd.EstepResult property), 6
pairs_set() (probreg.transformation.DeformableKinematicModel.Skinni method), 20
Permutohedral (class in probreg.gaussian_filtering), 14
Plot2DCallback (class in probreg.callbacks), 5
prepare() (probreg.transformation.TPSTransformation method), 20
probreg
  module, 20
probreg.callbacks
  module, 5
probreg.cost_functions
  module, 6
probreg.cpd
  module, 6
probreg.features
  module, 9
probreg.filterreg
  module, 11
probreg.gauss_transform
  module, 13
probreg.gaussian_filtering
  module, 14
probreg.gmmtree
  module, 14
probreg.l2dist_regs
  module, 15
probreg.math_utils
  module, 17
probreg.se3_op
  module, 18
probreg.transformation
  module, 18
pt1 (probreg.cpd.EstepResult property), 6
px (probreg.cpd.EstepResult property), 6

Q

q (probreg.cpd.MstepResult attribute), 7
q (probreg.cpd.MstepResult property), 7
q (probreg.filterreg.MstepResult attribute), 11

q (*probreg.filterreg.MstepResult property*), 11
q (*probreg.gmmtree.MstepResult attribute*), 14
q (*probreg.gmmtree.MstepResult property*), 14

R

rbf_kernel() (*in module probreg.math_utils*), 17
registration() (*probreg.cpd.CoherentPointDrift method*), 7
registration() (*probreg.filterreg.FilterReg method*), 12
registration() (*probreg.gmmtree.GMMTree method*), 15
registration() (*probreg.l2dist_regs.L2DistRegistration method*), 16
registration_cpd() (*in module probreg.cpd*), 9
registration_filterreg() (*in module probreg.filterreg*), 12
registration_gmmreg() (*in module probreg.l2dist_regs*), 16
registration_gmmtree() (*in module probreg.gmmtree*), 15
registration_svr() (*in module probreg.l2dist_regs*), 16
RigidCostFunction (*class in probreg.cost_functions*), 6
RigidCPD (*class in probreg.cpd*), 7
RigidFilterReg (*class in probreg.filterreg*), 12
RigidGMMReg (*class in probreg.l2dist_regs*), 16
RigidSVR (*class in probreg.l2dist_regs*), 16
RigidTransformation (*class in probreg.transformation*), 18

S

set_callbacks() (*probreg.cpd.CoherentPointDrift method*), 7
set_callbacks() (*probreg.filterreg.FilterReg method*), 12
set_callbacks() (*probreg.gmmtree.GMMTree method*), 15
set_callbacks() (*probreg.l2dist_regs.L2DistRegistration method*), 16
set_source() (*probreg.cpd.CoherentPointDrift method*), 7
set_source() (*probreg.cpd.ConstrainedNonRigidCPD method*), 9
set_source() (*probreg.cpd.NonRigidCPD method*), 8
set_source() (*probreg.filterreg.FilterReg method*), 12
set_source() (*probreg.gmmtree.GMMTree method*), 14
set_source() (*probreg.l2dist_regs.L2DistRegistration method*), 16
set_target_normals() (*probreg.filterreg.FilterReg method*), 12
sigma2 (*probreg.cpd.MstepResult attribute*), 6
sigma2 (*probreg.cpd.MstepResult property*), 7
sigma2 (*probreg.filterreg.MstepResult attribute*), 11
sigma2 (*probreg.filterreg.MstepResult property*), 11
skew() (*in module probreg.se3_op*), 18
squared_kernel_sum() (*in module probreg.math_utils*), 17

T

to_transformation() (*probreg.cost_functions.CostFunction method*), 6
to_transformation() (*probreg.cost_functions.RigidCostFunction method*), 6
to_transformation() (*probreg.cost_functions.TPSCostFunction method*), 6
tps_kernel() (*in module probreg.math_utils*), 17
TPSCostFunction (*class in probreg.cost_functions*), 6
TPSGMMReg (*class in probreg.l2dist_regs*), 16
TPSSVR (*class in probreg.l2dist_regs*), 16
TPSTransformation (*class in probreg.transformation*), 19
transform() (*probreg.transformation.Transformation method*), 18
transform_basis() (*probreg.transformation.TPSTransformation method*), 20
Transformation (*class in probreg.transformation*), 18
transformation (*probreg.cpd.MstepResult attribute*), 6
transformation (*probreg.cpd.MstepResult property*), 7
transformation (*probreg.filterreg.MstepResult attribute*), 11
transformation (*probreg.filterreg.MstepResult property*), 11
transformation (*probreg.gmmtree.MstepResult attribute*), 14
transformation (*probreg.gmmtree.MstepResult property*), 14
twist_mul() (*in module probreg.se3_op*), 18
twist_trans() (*in module probreg.se3_op*), 18