graphslam Documentation

Release 0.0.7

Jeff Irion

Jan 13, 2022

CONTENTS

CHAPTER

ONE

GRAPHSLAM

1.1 graphslam package

1.1.1 Subpackages

graphslam.edge package

Submodules

graphslam.edge.base_edge module

A base class for edges.

class graphslam.edge.base_edge.**BaseEdge**(*vertex_ids*, *information*, *estimate*, *vertices=None*)
Bases: object

A class for representing edges in Graph SLAM.

Parameters

- **vertex_ids** (*list[int]*) – The IDs of all vertices constrained by this edge
- **information** (*np.ndarray*) – The information matrix Ω_j associated with the edge
- **estimate** (BasePose, *np.ndarray, float*) – The expected measurement $\mathbf{z}_j$
- **vertices** (*list[graphslam.vertex.Vertex], None*) – A list of the vertices constrained by the edge

estimate
The expected measurement $\mathbf{z}_j$

Type *BasePose*, np.ndarray, float

information
The information matrix Ω_j associated with the edge

Type np.ndarray

vertex_ids
The IDs of all vertices constrained by this edge

Type list[int]

vertices
A list of the vertices constrained by the edge

Type list[*graphslam.vertex.Vertex*], None

_calc_jacobian(*err*, *dim*, *vertex_index*)
Calculate the Jacobian of the edge with respect to the specified vertex's pose.

Parameters

- **err** (*np.ndarray*) – The current error for the edge (see *BaseEdge.calc_error()*)
- **dim** (*int*) – The dimensionality of the compact pose representation
- **vertex_index** (*int*) – The index of the vertex (pose) for which we are computing the Jacobian

Returns The Jacobian of the edge with respect to the specified vertex's pose

Return type np.ndarray

calc_chi2()
Calculate the χ^2 error for the edge.

$$\mathbf{e}_j^T \Omega_j \mathbf{e}_j$$

Returns The χ^2 error for the edge

Return type float

calc_chi2_gradient_hessian()
Calculate the edge's contributions to the graph's χ^2 error, gradient ($\mathbf{b}$), and Hessian (H).

Returns

- *float* – The χ^2 error for the edge
- *dict* – The edge's contribution(s) to the gradient
- *dict* – The edge's contribution(s) to the Hessian

calc_error()
Calculate the error for the edge: $\mathbf{e}_j \in \mathbb{R}^{\bullet}$.

Returns The error for the edge

Return type np.ndarray, float

calc_jacobians()
Calculate the Jacobian of the edge's error with respect to each constrained pose.

$$\frac{\partial}{\partial \Delta \mathbf{x}^k} \left[\mathbf{e}_j(\mathbf{x}^k \boxplus \Delta \mathbf{x}^k) \right]$$

Returns The Jacobian matrices for the edge with respect to each constrained pose

Return type list[np.ndarray]

plot(*color=''*)
Plot the edge.

Parameters **color** (*str*) – The color that will be used to plot the edge

to_g2o()
Export the edge to the .g2o format.

Returns The edge in .g2o format

Return type str

graphslam.edge.base_edge.**EPSILON = 1e-06**
The difference that will be used for numerical differentiation

graphslam.edge.edge_odometry module

A class for odometry edges.

class graphslam.edge.edge_odometry.**EdgeOdometry**(*vertex_ids*, *information*, *estimate*, *vertices=None*)

Bases: *graphslam.edge.base_edge.BaseEdge*

A class for representing odometry edges in Graph SLAM.

Parameters

- **vertices** (*list[graphslam.vertex.Vertex]*) – A list of the vertices constrained by the edge
- **information** (*np.ndarray*) – The information matrix Ω_j associated with the edge
- **estimate** (BasePose) – The expected measurement $\mathbf{z}_j$

vertices
A list of the vertices constrained by the edge

Type list[*graphslam.vertex.Vertex*]

information
The information matrix Ω_j associated with the edge

Type np.ndarray

estimate
The expected measurement $\mathbf{z}_j$

Type *BasePose*

calc_error()
Calculate the error for the edge: $\mathbf{e}_j \in \mathbb{R}^{\bullet}$.

$$\mathbf{e}_j = \mathbf{z}_j - (p_2 \ominus p_1)$$

Returns The error for the edge

Return type np.ndarray

calc_jacobians()
Calculate the Jacobian of the edge's error with respect to each constrained pose.

$$\frac{\partial}{\partial \Delta \mathbf{x}^k} \left[\mathbf{e}_j(\mathbf{x}^k \boxplus \Delta \mathbf{x}^k) \right]$$

Returns The Jacobian matrices for the edge with respect to each constrained pose

Return type list[np.ndarray]

plot(*color='b'*)
Plot the edge.

Parameters **color** (*str*) – The color that will be used to plot the edge

to_g2o()
Export the edge to the .g2o format.

Returns The edge in .g2o format

Return type str

Module contents

graphslam.pose package

Submodules

graphslam.pose.base_pose module

A base class for poses.

class graphslam.pose.base_pose.**BasePose**
Bases: numpy.ndarray

A base class for poses.

copy()
Return a copy of the pose.

Returns A copy of the pose

Return type *BasePose*

property inverse
Return the pose's inverse.

Returns The pose's inverse

Return type *BasePose*

jacobian_boxplus()
Compute the Jacobian of $p_1 \boxplus \Delta\mathbf{x}$ w.r.t. $\Delta\mathbf{x}$ evaluated at $\Delta\mathbf{x} = \mathbf{0}$.

Returns The Jacobian of $p_1 \boxplus \Delta\mathbf{x}$ w.r.t. $\Delta\mathbf{x}$ evaluated at $\Delta\mathbf{x} = \mathbf{0}$

Return type np.ndarray

jacobian_self_ominus_other_wrt_other(*other*)
Compute the Jacobian of $p_1 \ominus p_2$ w.r.t. p_2.

Parameters **other** (BasePose) – The pose that is being subtracted from self

Returns The Jacobian of $p_1 \ominus p_2$ w.r.t. p_2.

Return type np.ndarray

jacobian_self_ominus_other_wrt_other_compact(*other*)
Compute the Jacobian of $p_1 \ominus p_2$ w.r.t. p_2.

Parameters **other** (BasePose) – The pose that is being subtracted from self

Returns The Jacobian of $p_1 \ominus p_2$ w.r.t. p_2.

Return type np.ndarray

jacobian_self_ominus_other_wrt_self(*other*)
Compute the Jacobian of $p_1 \ominus p_2$ w.r.t. p_1.

Parameters **other** (BasePose) – The pose that is being subtracted from self

Returns The Jacobian of $p_1 \ominus p_2$ w.r.t. p_1.

Return type np.ndarray

jacobian_self_ominus_other_wrt_self_compact(*other*)
Compute the Jacobian of $p_1 \ominus p_2$ w.r.t. p_1.

Parameters **other** (BasePose) – The pose that is being subtracted from self

Returns The Jacobian of $p_1 \ominus p_2$ w.r.t. p_1.

Return type np.ndarray

jacobian_self_oplus_other_wrt_other(*other*)
Compute the Jacobian of $p_1 \oplus p_2$ w.r.t. p_2.

Parameters **other** (BasePose) – The pose that is being added to self

Returns The Jacobian of $p_1 \oplus p_2$ w.r.t. p_2.

Return type np.ndarray

jacobian_self_oplus_other_wrt_other_compact(*other*)
Compute the Jacobian of $p_1 \oplus p_2$ w.r.t. p_2.

Parameters **other** (BasePose) – The pose that is being added to self

Returns The Jacobian of $p_1 \oplus p_2$ w.r.t. p_2.

Return type np.ndarray

jacobian_self_oplus_other_wrt_self(*other*)
Compute the Jacobian of $p_1 \oplus p_2$ w.r.t. p_1.

Parameters **other** (BasePose) – The pose that is being added to self

Returns The Jacobian of $p_1 \oplus p_2$ w.r.t. p_1.

Return type np.ndarray

jacobian_self_oplus_other_wrt_self_compact(*other*)
Compute the Jacobian of $p_1 \oplus p_2$ w.r.t. p_1.

Parameters **other** (BasePose) – The pose that is being added to self

Returns The Jacobian of $p_1 \oplus p_2$ w.r.t. p_1.

Return type np.ndarray

property orientation
Return the pose's orientation.

Returns The pose's orientation

Return type float, np.ndarray

property position
Return the pose's position.

Returns The pose's position

Return type np.ndarray

to_array()
Return the pose as a numpy array.

Returns The pose as a numpy array

Return type np.ndarray

to_compact()
Return the pose as a compact numpy array.

Returns The pose as a compact numpy array

Return type np.ndarray

graphslam.pose.r2 module

Representation of a point in $\mathbb{R}^2$.

class graphslam.pose.r2.**PoseR2**
Bases: *graphslam.pose.base_pose.BasePose*

A representation of a 2-D point.

Parameters **position** (*np.ndarray, list*) – The position in $\mathbb{R}^2$

copy()
Return a copy of the pose.

Returns A copy of the pose

Return type *PoseR2*

property inverse
Return the pose's inverse.

Returns The pose's inverse

Return type *PoseR2*

jacobian_boxplus()
Compute the Jacobian of $p_1 \boxplus \Delta\mathbf{x}$ w.r.t. $\Delta\mathbf{x}$ evaluated at $\Delta\mathbf{x} = \mathbf{0}$.

Returns The Jacobian of $p_1 \boxplus \Delta\mathbf{x}$ w.r.t. $\Delta\mathbf{x}$ evaluated at $\Delta\mathbf{x} = \mathbf{0}$

Return type np.ndarray

jacobian_self_ominus_other_wrt_other(*other*)
Compute the Jacobian of $p_1 \ominus p_2$ w.r.t. p_2.

Parameters **other** (BasePose) – The pose that is being subtracted from `self`

Returns The Jacobian of $p_1 \ominus p_2$ w.r.t. p_2.

Return type np.ndarray

jacobian_self_ominus_other_wrt_other_compact(*other*)
Compute the Jacobian of $p_1 \ominus p_2$ w.r.t. p_2.

Parameters **other** (BasePose) – The pose that is being subtracted from `self`

Returns The Jacobian of $p_1 \ominus p_2$ w.r.t. p_2.

Return type np.ndarray

jacobian_self_ominus_other_wrt_self(*other*)
Compute the Jacobian of $p_1 \ominus p_2$ w.r.t. p_1.

Parameters **other** (BasePose) – The pose that is being subtracted from `self`

Returns The Jacobian of $p_1 \ominus p_2$ w.r.t. p_1.

Return type np.ndarray

jacobian_self_ominus_other_wrt_self_compact(*other*)
Compute the Jacobian of $p_1 \ominus p_2$ w.r.t. p_1.

Parameters **other** (BasePose) – The pose that is being subtracted from self

Returns The Jacobian of $p_1 \ominus p_2$ w.r.t. p_1.

Return type np.ndarray

jacobian_self_oplus_other_wrt_other(*other*)
Compute the Jacobian of $p_1 \oplus p_2$ w.r.t. p_2.

Parameters **other** (BasePose) – The pose that is being added to self

Returns The Jacobian of $p_1 \oplus p_2$ w.r.t. p_2.

Return type np.ndarray

jacobian_self_oplus_other_wrt_other_compact(*other*)
Compute the Jacobian of $p_1 \oplus p_2$ w.r.t. p_2.

Parameters **other** (BasePose) – The pose that is being added to self

Returns The Jacobian of $p_1 \oplus p_2$ w.r.t. p_2.

Return type np.ndarray

jacobian_self_oplus_other_wrt_self(*other*)
Compute the Jacobian of $p_1 \oplus p_2$ w.r.t. p_1.

Parameters **other** (BasePose) – The pose that is being added to self

Returns The Jacobian of $p_1 \oplus p_2$ w.r.t. p_1.

Return type np.ndarray

jacobian_self_oplus_other_wrt_self_compact(*other*)
Compute the Jacobian of $p_1 \oplus p_2$ w.r.t. p_1.

Parameters **other** (BasePose) – The pose that is being added to self

Returns The Jacobian of $p_1 \oplus p_2$ w.r.t. p_1.

Return type np.ndarray

property orientation
Return the pose's orientation.

Returns A PoseR2 object has no orientation, so this will always return 0.

Return type float

property position
Return the pose's position.

Returns The position portion of the pose

Return type np.ndarray

to_array()
Return the pose as a numpy array.

Returns The pose as a numpy array

Return type np.ndarray

to_compact()
Return the pose as a compact numpy array.

Returns The pose as a compact numpy array

Return type np.ndarray

graphslam.pose.r3 module

Representation of a point in $\mathbb{R}^3$.

class graphslam.pose.r3.**PoseR3**
Bases: *graphslam.pose.base_pose.BasePose*

A representation of a 3-D point.

Parameters **position** (*np.ndarray, list*) – The position in $\mathbb{R}^3$

copy()
Return a copy of the pose.

Returns A copy of the pose

Return type *PoseR3*

property inverse
Return the pose's inverse.

Returns The pose's inverse

Return type *PoseR3*

jacobian_boxplus()
Compute the Jacobian of $p_1 \boxplus \Delta\mathbf{x}$ w.r.t. $\Delta\mathbf{x}$ evaluated at $\Delta\mathbf{x} = \mathbf{0}$.

Returns The Jacobian of $p_1 \boxplus \Delta\mathbf{x}$ w.r.t. $\Delta\mathbf{x}$ evaluated at $\Delta\mathbf{x} = \mathbf{0}$

Return type np.ndarray

jacobian_self_ominus_other_wrt_other(*other*)
Compute the Jacobian of $p_1 \ominus p_2$ w.r.t. p_2.

Parameters **other** (BasePose) – The pose that is being subtracted from `self`

Returns The Jacobian of $p_1 \ominus p_2$ w.r.t. p_2.

Return type np.ndarray

jacobian_self_ominus_other_wrt_other_compact(*other*)
Compute the Jacobian of $p_1 \ominus p_2$ w.r.t. p_2.

Parameters **other** (BasePose) – The pose that is being subtracted from `self`

Returns The Jacobian of $p_1 \ominus p_2$ w.r.t. p_2.

Return type np.ndarray

jacobian_self_ominus_other_wrt_self(*other*)
Compute the Jacobian of $p_1 \ominus p_2$ w.r.t. p_1.

Parameters **other** (BasePose) – The pose that is being subtracted from `self`

Returns The Jacobian of $p_1 \ominus p_2$ w.r.t. p_1.

Return type np.ndarray

jacobian_self_ominus_other_wrt_self_compact(*other*)
Compute the Jacobian of $p_1 \ominus p_2$ w.r.t. p_1.

Parameters **other** (BasePose) – The pose that is being subtracted from self

Returns The Jacobian of $p_1 \ominus p_2$ w.r.t. p_1.

Return type np.ndarray

jacobian_self_oplus_other_wrt_other(*other*)
Compute the Jacobian of $p_1 \oplus p_2$ w.r.t. p_2.

Parameters **other** (BasePose) – The pose that is being added to self

Returns The Jacobian of $p_1 \oplus p_2$ w.r.t. p_2.

Return type np.ndarray

jacobian_self_oplus_other_wrt_other_compact(*other*)
Compute the Jacobian of $p_1 \oplus p_2$ w.r.t. p_2.

Parameters **other** (BasePose) – The pose that is being added to self

Returns The Jacobian of $p_1 \oplus p_2$ w.r.t. p_2.

Return type np.ndarray

jacobian_self_oplus_other_wrt_self(*other*)
Compute the Jacobian of $p_1 \oplus p_2$ w.r.t. p_1.

Parameters **other** (BasePose) – The pose that is being added to self

Returns The Jacobian of $p_1 \oplus p_2$ w.r.t. p_1.

Return type np.ndarray

jacobian_self_oplus_other_wrt_self_compact(*other*)
Compute the Jacobian of $p_1 \oplus p_2$ w.r.t. p_1.

Parameters **other** (BasePose) – The pose that is being added to self

Returns The Jacobian of $p_1 \oplus p_2$ w.r.t. p_1.

Return type np.ndarray

property orientation
Return the pose's orientation.

Returns A PoseR3 object has no orientation, so this will always return 0.

Return type float

property position
Return the pose's position.

Returns The position portion of the pose

Return type np.ndarray

to_array()
Return the pose as a numpy array.

Returns The pose as a numpy array

Return type np.ndarray

to_compact()
Return the pose as a compact numpy array.

Returns The pose as a compact numpy array

Return type np.ndarray

graphslam.pose.se2 module

Representation of a pose in $SE(2)$.

class graphslam.pose.se2.**PoseSE2**
Bases: *graphslam.pose.base_pose.BasePose*

A representation of a pose in $SE(2)$.

Parameters

- **position** (*np.ndarray, list*) – The position in $\mathbb{R}^2$
- **orientation** (*float*) – The angle of the pose (in radians)

copy()
Return a copy of the pose.

Returns A copy of the pose

Return type *PoseSE2*

classmethod from_matrix(*matrix*)
Return the pose as an $SE(2)$ matrix.

Parameters **matrix** (*np.ndarray*) – The $SE(2)$ matrix that will be converted to a *PoseSE2* instance

Returns The matrix as a *PoseSE2* object

Return type *PoseSE2*

property inverse
Return the pose's inverse.

Returns The pose's inverse

Return type *PoseSE2*

jacobian_boxplus()
Compute the Jacobian of $p_1 \boxplus \Delta\mathbf{x}$ w.r.t. $\Delta\mathbf{x}$ evaluated at $\Delta\mathbf{x} = \mathbf{0}$.

Returns The Jacobian of $p_1 \boxplus \Delta\mathbf{x}$ w.r.t. $\Delta\mathbf{x}$ evaluated at $\Delta\mathbf{x} = \mathbf{0}$

Return type np.ndarray

jacobian_self_ominus_other_wrt_other(*other*)
Compute the Jacobian of $p_1 \ominus p_2$ w.r.t. p_2.

Parameters **other** (BasePose) – The pose that is being subtracted from `self`

Returns The Jacobian of $p_1 \ominus p_2$ w.r.t. p_2.

Return type np.ndarray

jacobian_self_ominus_other_wrt_other_compact(*other*)
Compute the Jacobian of $p_1 \ominus p_2$ w.r.t. p_2.

Parameters **other** (BasePose) – The pose that is being subtracted from `self`

Returns The Jacobian of $p_1 \ominus p_2$ w.r.t. p_2.

Return type np.ndarray

jacobian_self_ominus_other_wrt_self(*other*)
Compute the Jacobian of $p_1 \ominus p_2$ w.r.t. p_1.

Parameters **other** (BasePose) – The pose that is being subtracted from `self`

Returns The Jacobian of $p_1 \ominus p_2$ w.r.t. p_1.

Return type np.ndarray

jacobian_self_ominus_other_wrt_self_compact(*other*)
Compute the Jacobian of $p_1 \ominus p_2$ w.r.t. p_1.

Parameters **other** (BasePose) – The pose that is being subtracted from `self`

Returns The Jacobian of $p_1 \ominus p_2$ w.r.t. p_1.

Return type np.ndarray

jacobian_self_oplus_other_wrt_other(*other*)
Compute the Jacobian of $p_1 \oplus p_2$ w.r.t. p_2.

Parameters **other** (BasePose) – The pose that is being added to `self`

Returns The Jacobian of $p_1 \oplus p_2$ w.r.t. p_2.

Return type np.ndarray

jacobian_self_oplus_other_wrt_other_compact(*other*)
Compute the Jacobian of $p_1 \oplus p_2$ w.r.t. p_2.

Parameters **other** (BasePose) – The pose that is being added to `self`

Returns The Jacobian of $p_1 \oplus p_2$ w.r.t. p_2.

Return type np.ndarray

jacobian_self_oplus_other_wrt_self(*other*)
Compute the Jacobian of $p_1 \oplus p_2$ w.r.t. p_1.

Parameters **other** (BasePose) – The pose that is being added to `self`

Returns The Jacobian of $p_1 \oplus p_2$ w.r.t. p_1.

Return type np.ndarray

jacobian_self_oplus_other_wrt_self_compact(*other*)
Compute the Jacobian of $p_1 \oplus p_2$ w.r.t. p_1.

Parameters **other** (BasePose) – The pose that is being added to `self`

Returns The Jacobian of $p_1 \oplus p_2$ w.r.t. p_1.

Return type np.ndarray

property orientation
Return the pose's orientation.

Returns The angle of the pose

Return type float

property position
Return the pose's position.

Returns The position portion of the pose

Return type np.ndarray

to_array()
Return the pose as a numpy array.

Returns The pose as a numpy array

Return type np.ndarray

to_compact()
Return the pose as a compact numpy array.

Returns The pose as a compact numpy array

Return type np.ndarray

to_matrix()
Return the pose as an $SE(2)$ matrix.

Returns The pose as an $SE(2)$ matrix

Return type np.ndarray

graphslam.pose.se3 module

Representation of a pose in $SE(3)$.

class graphslam.pose.se3.**PoseSE3**
Bases: *graphslam.pose.base_pose.BasePose*

A representation of a pose in $SE(3)$.

Parameters

- **position** (*np.ndarray, list*) – The position in $\mathbb{R}^3$
- **orientation** (*np.ndarray, list*) – The orientation of the pose as a unit quaternion: $[q_x, q_y, q_z, q_w]$

copy()
Return a copy of the pose.

Returns A copy of the pose

Return type *PoseSE3*

property inverse
Return the pose's inverse.

Returns The pose's inverse

Return type *PoseSE3*

jacobian_boxplus()
Compute the Jacobian of $p_1 \boxplus \Delta\mathbf{x}$ w.r.t. $\Delta\mathbf{x}$ evaluated at $\Delta\mathbf{x} = \mathbf{0}$.

Returns The Jacobian of $p_1 \boxplus \Delta\mathbf{x}$ w.r.t. $\Delta\mathbf{x}$ evaluated at $\Delta\mathbf{x} = \mathbf{0}$

Return type np.ndarray

jacobian_self_ominus_other_wrt_other(*other*)
Compute the Jacobian of $p_1 \ominus p_2$ w.r.t. p_2.

Parameters **other** (BasePose) – The pose that is being subtracted from self

Returns The Jacobian of $p_1 \ominus p_2$ w.r.t. p_2.

Return type np.ndarray

jacobian_self_ominus_other_wrt_other_compact(*other*)
Compute the Jacobian of $p_1 \ominus p_2$ w.r.t. p_2.

Parameters **other** (BasePose) – The pose that is being subtracted from self

Returns The Jacobian of $p_1 \ominus p_2$ w.r.t. p_2.

Return type np.ndarray

jacobian_self_ominus_other_wrt_self(*other*)
Compute the Jacobian of $p_1 \ominus p_2$ w.r.t. p_1.

Parameters **other** (BasePose) – The pose that is being subtracted from self

Returns The Jacobian of $p_1 \ominus p_2$ w.r.t. p_1.

Return type np.ndarray

jacobian_self_ominus_other_wrt_self_compact(*other*)
Compute the Jacobian of $p_1 \ominus p_2$ w.r.t. p_1.

Parameters **other** (BasePose) – The pose that is being subtracted from self

Returns The Jacobian of $p_1 \ominus p_2$ w.r.t. p_1.

Return type np.ndarray

jacobian_self_oplus_other_wrt_other(*other*)
Compute the Jacobian of $p_1 \oplus p_2$ w.r.t. p_2.

Parameters **other** (BasePose) – (Unused) The pose that is being added to self

Returns The Jacobian of $p_1 \oplus p_2$ w.r.t. p_2.

Return type np.ndarray

jacobian_self_oplus_other_wrt_other_compact(*other*)
Compute the Jacobian of $p_1 \oplus p_2$ w.r.t. p_2.

Parameters **other** (BasePose) – (Unused) The pose that is being added to self

Returns The Jacobian of $p_1 \oplus p_2$ w.r.t. p_2.

Return type np.ndarray

jacobian_self_oplus_other_wrt_self(*other*)
Compute the Jacobian of $p_1 \oplus p_2$ w.r.t. p_1.

Parameters **other** (BasePose) – The pose that is being added to self

Returns The Jacobian of $p_1 \oplus p_2$ w.r.t. p_1.

Return type np.ndarray

jacobian_self_oplus_other_wrt_self_compact(*other*)
Compute the Jacobian of $p_1 \oplus p_2$ w.r.t. p_1.

Parameters **other** (BasePose) – The pose that is being added to self

Returns The Jacobian of $p_1 \oplus p_2$ w.r.t. p_1.

Return type np.ndarray

normalize()
Normalize the quaternion portion of the pose.

property orientation
Return the pose's orientation.

Returns The pose's quaternion

Return type float

property position
Return the pose's position.

Returns The position portion of the pose

Return type np.ndarray

`to_array`()
Return the pose as a numpy array.

Returns The pose as a numpy array

Return type np.ndarray

`to_compact`()
Return the pose as a compact numpy array.

Returns The pose as a compact numpy array

Return type np.ndarray

`to_matrix`()
Return the pose as an $SE(3)$ matrix.

Returns The pose as an $SE(3)$ matrix

Return type np.ndarray

Module contents

1.1.2 Submodules

graphslam.graph module

A `Graph` class that stores the edges and vertices required for Graph SLAM.

Given:

- A set of M edges (i.e., constraints) $\mathcal{E}$
  - $e_j \in \mathcal{E}$ is an edge
  - $\mathbf{e}_j \in \mathbb{R}^\bullet$ is the error associated with that edge, where $\bullet$ is a scalar that depends on the type of edge
  - Ω_j is the $\bullet \times \bullet$ information matrix associated with edge e_j
- A set of N vertices $\mathcal{V}$
  - $v_i \in \mathcal{V}$ is a vertex
  - $\mathbf{x}_i \in \mathbb{R}^c$ is the compact pose associated with v_i
  - $\boxplus$ is the pose composition operator that yields a (non-compact) pose that lies in (a subspace of) $\mathbb{R}^d$

We want to optimize

$$\chi^2 = \sum_{e_j \in \mathcal{E}} \mathbf{e}_j^T \Omega_j \mathbf{e}_j.$$

Let

$$\mathbf{x} := \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_N \end{bmatrix} \in \mathbb{R}^{cN}.$$

We will solve this optimization problem iteratively. Let

$$\mathbf{x}^{k+1} := \mathbf{x}^k \boxplus \Delta\mathbf{x}^k.$$

The χ^2 error at iteration $k+1$ is

$$\chi^2_{k+1} = \sum_{e_j \in \mathcal{E}} \underbrace{\left[\mathbf{e}_j(\mathbf{x}^{k+1})\right]^T}_{1\times\bullet} \underbrace{\Omega_j}_{\bullet\times\bullet} \underbrace{\mathbf{e}_j(\mathbf{x}^{k+1})}_{\bullet\times 1}.$$

We will linearize the errors as:

$$\begin{aligned}
\mathbf{e}_j(\mathbf{x}^{k+1}) &= \mathbf{e}_j(\mathbf{x}^k \boxplus \Delta\mathbf{x}^k) \\
&\approx \mathbf{e}_j(\mathbf{x}^k) + \frac{\partial}{\partial \Delta\mathbf{x}^k}\left[\mathbf{e}_j(\mathbf{x}^k \boxplus \Delta\mathbf{x}^k)\right]\Delta\mathbf{x}^k \\
&= \mathbf{e}_j(\mathbf{x}^k) + \left(\left.\frac{\partial \mathbf{e}_j(\mathbf{x}^k \boxplus \Delta\mathbf{x}^k)}{\partial(\mathbf{x}^k \boxplus \Delta\mathbf{x}^k)}\right|_{\Delta\mathbf{x}^k=\mathbf{0}}\right)\frac{\partial(\mathbf{x}^k \boxplus \Delta\mathbf{x}^k)}{\partial\Delta\mathbf{x}^k}\Delta\mathbf{x}^k.
\end{aligned}$$

Plugging this into the formula for χ^2, we get:

$$\begin{aligned}
\chi^2_{k+1} \approx \quad & \sum_{e_j\in\mathcal{E}} \underbrace{[\mathbf{e}_j(\mathbf{x}^k)]^T}_{1\times\bullet} \underbrace{\Omega_j}_{\bullet\times\bullet} \underbrace{\mathbf{e}_j(\mathbf{x}^k)}_{\bullet\times 1} \\
& + \sum_{e_j\in\mathcal{E}} \underbrace{[\mathbf{e}_j(\mathbf{x}^k)]^T}_{1\times\bullet} \underbrace{\Omega_j}_{\bullet\times\bullet} \underbrace{\left(\left.\frac{\partial \mathbf{e}_j(\mathbf{x}^k \boxplus \Delta\mathbf{x}^k)}{\partial(\mathbf{x}^k \boxplus \Delta\mathbf{x}^k)}\right|_{\Delta\mathbf{x}^k=\mathbf{0}}\right)}_{\bullet\times dN} \underbrace{\frac{\partial(\mathbf{x}^k \boxplus \Delta\mathbf{x}^k)}{\partial\Delta\mathbf{x}^k}}_{dN\times cN} \underbrace{\Delta\mathbf{x}^k}_{cN\times 1} \\
& + \sum_{e_j\in\mathcal{E}} \underbrace{(\Delta\mathbf{x}^k)^T}_{1\times cN} \underbrace{\left(\frac{\partial(\mathbf{x}^k \boxplus \Delta\mathbf{x}^k)}{\partial\Delta\mathbf{x}^k}\right)^T}_{cN\times dN} \underbrace{\left(\left.\frac{\partial \mathbf{e}_j(\mathbf{x}^k \boxplus \Delta\mathbf{x}^k)}{\partial(\mathbf{x}^k \boxplus \Delta\mathbf{x}^k)}\right|_{\Delta\mathbf{x}^k=\mathbf{0}}\right)^T}_{dN\times\bullet} \underbrace{\Omega_j}_{\bullet\times\bullet} \underbrace{\left(\left.\frac{\partial \mathbf{e}_j(\mathbf{x}^k \boxplus \Delta\mathbf{x}^k)}{\partial(\mathbf{x}^k \boxplus \Delta\mathbf{x}^k)}\right|_{\Delta\mathbf{x}^k=\mathbf{0}}\right)}_{\bullet\times dN} \underbrace{\frac{\partial(\mathbf{x}^k \boxplus \Delta\mathbf{x}^k)}{\partial\Delta\mathbf{x}^k}}_{dN\times cN} \underbrace{\Delta\mathbf{x}^k}_{cN\times 1} \\
= \chi^2_k + 2\mathbf{b}^T\Delta\mathbf{x}^k + (\Delta\mathbf{x}^k)^T H \Delta\mathbf{x}^k,
\end{aligned}$$

where

$$\begin{aligned}
\mathbf{b}^T &= \sum_{e_j\in\mathcal{E}} \underbrace{[\mathbf{e}_j(\mathbf{x}^k)]^T}_{1\times\bullet} \underbrace{\Omega_j}_{\bullet\times\bullet} \underbrace{\left(\left.\frac{\partial \mathbf{e}_j(\mathbf{x}^k \boxplus \Delta\mathbf{x}^k)}{\partial(\mathbf{x}^k \boxplus \Delta\mathbf{x}^k)}\right|_{\Delta\mathbf{x}^k=\mathbf{0}}\right)}_{\bullet\times dN} \underbrace{\frac{\partial(\mathbf{x}^k \boxplus \Delta\mathbf{x}^k)}{\partial\Delta\mathbf{x}^k}}_{dN\times cN} \\
H &= \sum_{e_j\in\mathcal{E}} \underbrace{\left(\frac{\partial(\mathbf{x}^k \boxplus \Delta\mathbf{x}^k)}{\partial\Delta\mathbf{x}^k}\right)^T}_{cN\times dN} \underbrace{\left(\left.\frac{\partial \mathbf{e}_j(\mathbf{x}^k \boxplus \Delta\mathbf{x}^k)}{\partial(\mathbf{x}^k \boxplus \Delta\mathbf{x}^k)}\right|_{\Delta\mathbf{x}^k=\mathbf{0}}\right)^T}_{dN\times\bullet} \underbrace{\Omega_j}_{\bullet\times\bullet} \underbrace{\left(\left.\frac{\partial \mathbf{e}_j(\mathbf{x}^k \boxplus \Delta\mathbf{x}^k)}{\partial(\mathbf{x}^k \boxplus \Delta\mathbf{x}^k)}\right|_{\Delta\mathbf{x}^k=\mathbf{0}}\right)}_{\bullet\times dN} \underbrace{\frac{\partial(\mathbf{x}^k \boxplus \Delta\mathbf{x}^k)}{\partial\Delta\mathbf{x}^k}}_{dN\times cN}.
\end{aligned}$$

Using this notation, we obtain the optimal update as

$$\Delta\mathbf{x}^k = -H^{-1}\mathbf{b}.$$

We apply this update to the poses and repeat until convergence.

class graphslam.graph.**Graph**(*edges*, *vertices*)

Bases: object

A graph that will be optimized via Graph SLAM.

Parameters

- **edges** (*list[graphslam.edge.base_edge.BaseEdge]*) – A list of the vertices in the graph

- **vertices** (*list[graphslam.vertex.Vertex]*) – A list of the vertices in the graph

_chi2
The current χ^2 error, or `None` if it has not yet been computed

Type float, None

_edges
A list of the edges (i.e., constraints) in the graph

Type list[*graphslam.edge.base_edge.BaseEdge*]

_gradient
The gradient $\mathbf{b}$ of the χ^2 error, or `None` if it has not yet been computed

Type numpy.ndarray, None

_hessian
The Hessian matrix H, or `None` if it has not yet been computed

Type scipy.sparse.lil_matrix, None

_vertices
A list of the vertices in the graph

Type list[*graphslam.vertex.Vertex*]

_calc_chi2_gradient_hessian()
Calculate the χ^2 error, the gradient $\mathbf{b}$, and the Hessian H.

_link_edges()
Fill in the `vertices` attributes for the graph's edges.

calc_chi2()
Calculate the χ^2 error for the `Graph`.

Returns The χ^2 error

Return type float

optimize(*tol=0.0001*, *max_iter=20*, *fix_first_pose=True*)
Optimize the χ^2 error for the `Graph`.

Parameters

- **tol** (*float*) – If the relative decrease in the χ^2 error between iterations is less than `tol`, we will stop
- **max_iter** (*int*) – The maximum number of iterations
- **fix_first_pose** (*bool*) – If `True`, we will fix the first pose

plot(*vertex_color='r'*, *vertex_marker='o'*, *vertex_markersize=3*, *edge_color='b'*, *title=None*)
Plot the graph.

Parameters

- **vertex_color** (*str*) – The color that will be used to plot the vertices
- **vertex_marker** (*str*) – The marker that will be used to plot the vertices
- **vertex_markersize** (*int*) – The size of the plotted vertices
- **edge_color** (*str*) – The color that will be used to plot the edges
- **title** (*str, None*) – The title that will be used for the plot

to_g2o(*outfile*)
Save the graph in .g2o format.

Parameters **outfile** (*str*) – The path where the graph will be saved

class graphslam.graph.**_Chi2GradientHessian**(*dim*)
Bases: object

A class that is used to aggregate the χ^2 error, gradient, and Hessian.

Parameters **dim** (*int*) – The compact dimensionality of the poses

chi2
The χ^2 error

Type float

dim
The compact dimensionality of the poses

Type int

gradient
The contributions to the gradient vector

Type defaultdict

hessian
The contributions to the Hessian matrix

Type defaultdict

static update(*chi2_grad_hess*, *incoming*)
Update the χ^2 error and the gradient and Hessian dictionaries.

Parameters

- **chi2_grad_hess** (_Chi2GradientHessian) – The _Chi2GradientHessian that will be updated
- **incoming** (*tuple*) – TODO

graphslam.load module

Functions for loading graphs.

graphslam.load.**load_g2o_se2**(*infile*)
Load an $SE(2)$ graph from a .g2o file.

Parameters **infile** (*str*) – The path to the .g2o file

Returns The loaded graph

Return type *Graph*

graphslam.load.**load_g2o_se3**(*infile*)
Load an $SE(3)$ graph from a .g2o file.

Parameters **infile** (*str*) – The path to the .g2o file

Returns The loaded graph

Return type *Graph*

graphslam.util module

Utility functions used throughout the package.

graphslam.util.**neg_pi_to_pi**(*angle*)
Normalize `angle` to be in $[-\pi, \pi)$.

Parameters **angle** (*float*) – An angle (in radians)

Returns The angle normalized to $[-\pi, \pi)$

Return type float

graphslam.util.**solve_for_edge_dimensionality**(*n*)
Solve for the dimensionality of an edge.

In a .g2o file, an edge is specified as `<estimate> <information matrix>`, where only the upper triangular portion of the matrix is provided.

This solves the problem:

$$d + \frac{d(d+1)}{2} = n$$

Returns The dimensionality of the edge

Return type int

graphslam.util.**upper_triangular_matrix_to_full_matrix**(*arr*, *n*)
Given an upper triangular matrix, return the full matrix.

Parameters

- **arr** (*np.ndarray*) – The upper triangular portion of the matrix
- **n** (*int*) – The size of the matrix

Returns **mat** – The full matrix

Return type np.ndarray

graphslam.vertex module

A `Vertex` class.

class graphslam.vertex.**Vertex**(*vertex_id*, *pose*, *vertex_index=None*)
Bases: `object`

A class for representing a vertex in Graph SLAM.

Parameters

- **vertex_id** (*int*) – The vertex's unique ID
- **pose** (graphslam.pose.base_pose.BasePose) – The pose associated with the vertex
- **vertex_index** (*int, None*) – The vertex's index in the graph's `vertices` list

id
The vertex's unique ID

Type int

index
The vertex's index in the graph's `vertices` list

Type int, None

pose
The pose associated with the vertex

Type *graphslam.pose.base_pose.BasePose*

plot(*color='r'*, *marker='o'*, *markersize=3*)
Plot the vertex.

Parameters

- **color** (*str*) – The color that will be used to plot the vertex
- **marker** (*str*) – The marker that will be used to plot the vertex
- **markersize** (*int*) – The size of the plotted vertex

to_g2o()
Export the vertex to the .g2o format.

Returns The vertex in .g2o format

Return type str

1.1.3 Module contents

Graph SLAM solver in Python.

Documentation for this package can be found at https://python-graphslam.readthedocs.io/.

This package implements a Graph SLAM solver in Python.

CHAPTER

TWO

FEATURES

- Optimize $\mathbb{R}^2$, $\mathbb{R}^3$, $SE(2)$, and $SE(3)$ datasets
- Analytic Jacobians
- Supports odometry edges
- Import and export .g2o files for $SE(2)$ and $SE(3)$ datasets

CHAPTER

THREE

INSTALLATION

```
pip install graphslam
```

CHAPTER

FOUR

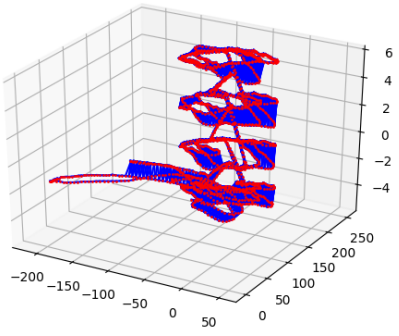
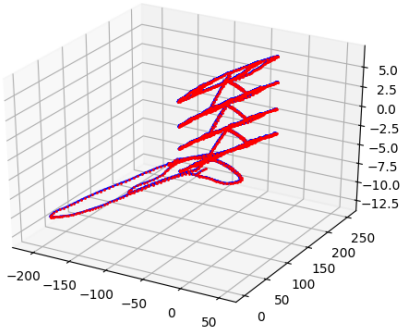
EXAMPLE USAGE

4.1 SE(3) Dataset

```
>>> from graphslam.load import load_g2o_se3

>>> g = load_g2o_se3("parking-garage.g2o")  # https://lucacarlone.mit.edu/datasets/

>>> g.plot(vertex_markersize=1)

>>> g.calc_chi2()

16720.020602489112

>>> g.optimize()

>>> g.plot(vertex_markersize=1)
```

Output:

```
Iteration                chi^2        rel. change
---------                -----        -----------
        0          16720.0206
        1             26.5495          -0.998412
        2              1.2712          -0.952119
        3              1.2402          -0.024439
        4              1.2396          -0.000456
        5              1.2395          -0.000091
```

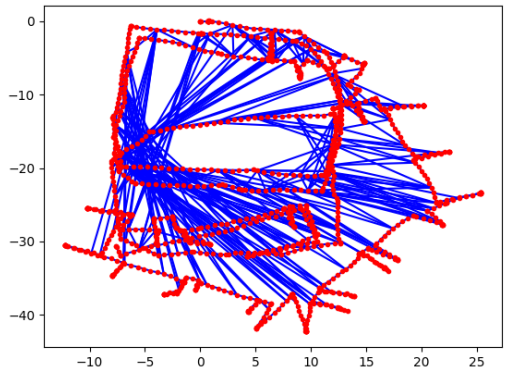
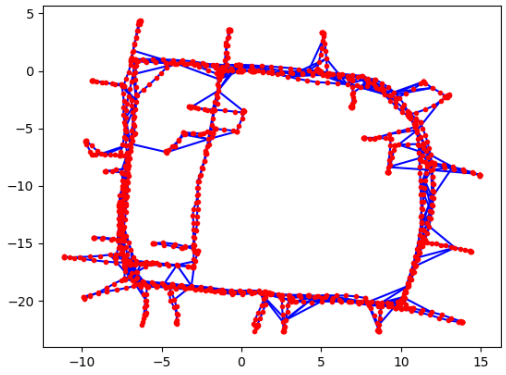
Original	Optimized
	

4.2 SE(2) Dataset

```
>>> from graphslam.load import load_g2o_se2

>>> g = load_g2o_se2("input_INTEL.g2o")  # https://lucacarlone.mit.edu/datasets/

>>> g.plot()

>>> g.calc_chi2()

7191686.382493544

>>> g.optimize()

>>> g.plot()
```

Output:

```
Iteration                chi^2        rel. change
---------                -----        -----------
        0         7191686.3825
        1       319915276.1284          43.484042
        2       124894535.1749          -0.609601
        3          338185.8171          -0.997292
        4             734.5142          -0.997828
        5             215.8405          -0.706145
        6             215.8405          -0.000000
```

Original	Optimized
	

CHAPTER

FIVE

REFERENCES AND LINKS

- A tutorial on graph-based SLAM
- A tutorial on SE(3) transformation parameterizations and on-manifold optimization
- Datasets from Luca Carlone

CHAPTER

SIX

LIVE CODING GRAPH SLAM IN PYTHON

If you're interested, you can watch as I coded this up.

1. Live coding Graph SLAM in Python (Part 1)
2. Live coding Graph SLAM in Python (Part 2)
3. Live coding Graph SLAM in Python (Part 3)
4. Live coding Graph SLAM in Python (Part 4)
5. Live coding Graph SLAM in Python (Part 5)

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

g

INDEX

Symbols

B

C

D

E

F

G

H

I

J