MapTransformer

Release 1.0.0

Geoffrey Biggs

Jan 15, 2021

CONTENTS:

1 Using the library 3

2 YAML file format 5

3 Sample application 7

4 API 9

Index 15

Ths Map Transformer library provides transformations between two maps which may be close but not quite aligned, with the alignment difference varying across the map. A point provided in the coordinate space of one map is transformed into the coordinate space of the other, taking account of the non-linear transforms to ensure it is at the equivalent location in the map.

CHAPTER

ONE

USING THE LIBRARY

The main entry point to the Map Transformer library is the *map_transformer::Transformer* class. This class provides transformations in both directions between a reference map and one other map (termed the "robot map").

An instance of the *Transformer* object must be provided with a YAML document to load. This can be either passed to the constructor or passed to the *load()* method after construction. The YAML document contains information about the two maps, and a list of correspondence points. For a description of the YAML file format, see the next section.

The correspondence points must be provided. These points describe which points in one map are equivalent to which points in the other map. In other words, point A in the reference map is the same corner of two walls as point B in the robot map, even though those two points may have significantly different coordinates due to differences in map alignment, skew, etc. Currently, the correspondence points must be manually calculated and specified by hand.

In general, the more correspondence points you provide, the more accurate the transformation of points between the two maps will be.

Once *Transformer* object instance has been constructed and loaded with map information, you can call the following two member functions to transform points.

- *to_ref()* Transforms a point from the robot map to its equivalent point in the reference map.
- *to_robot()* Transforms a point from the reference map to its equivalent point in the robot map.

CHAPTER

TWO

YAML FILE FORMAT

The YAML file provided as input to the *Transformer* class must comply with the following format.

```
ref_map:
  name: [The human-readable name of the map, for visualisation purposes]
  image_file: [OPTIONAL; the path to the image file for the map]
  size: [The size, in pixels, of the map as a pair of numbers, e.g. [100, 80]]
  correspondence_points: [A YAML list of pairs of numbers providing the␣
↪correspondence points for the reference map.]
robot_map:
  name: [The human-readable name of the map, for visualisation purposes]
  image_file: [OPTIONAL; the path to the image file for the map]
  size: [694, 386]
  correspondence_points: [A YAML list of pairs of numbers providing the␣
↪correspondence points for the robot map.]
```

The correspondence point lists must be in the same order. That is, the first point in one list corresponds to the first point in the other list, and so on.

For example YAML files, see the samples directory.

CHAPTER

THREE

SAMPLE APPLICATION

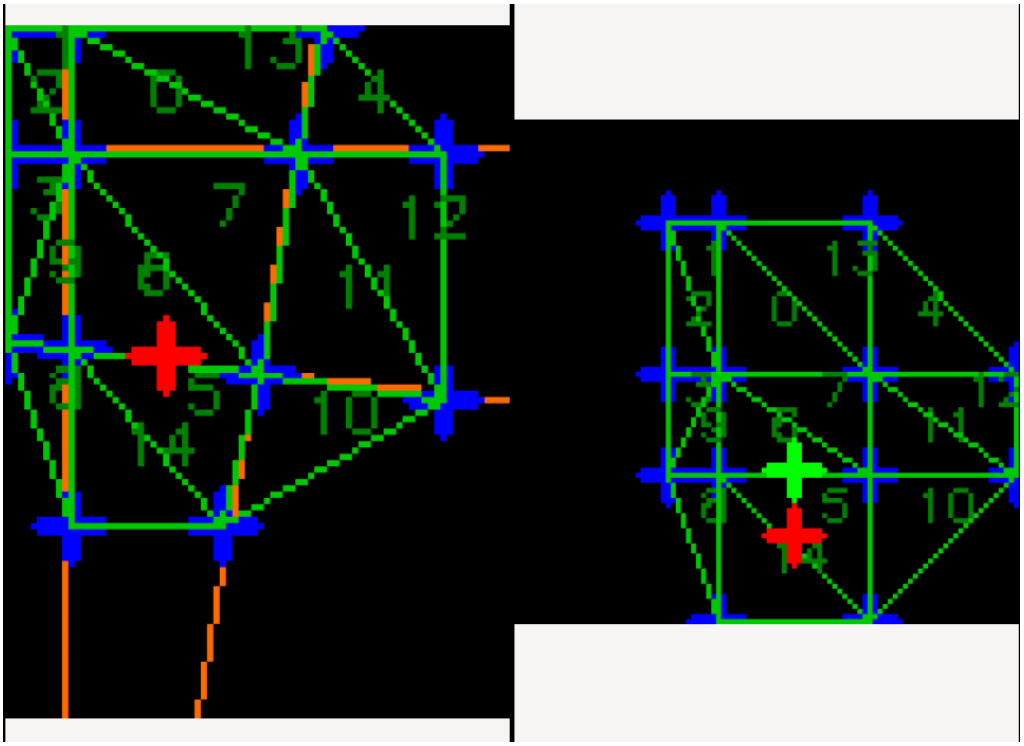
A sample application is provided in the source file *visualiser.cpp*. This is installed into the *sample* directory, usually located at *<install prefix>/share/map_transformer/sample*.

The sample application displays the reference map and robot map as specified in a YAML file. It can optionally display the correspondence points, triangulation, and triangle ID numbers.

Launch the sample using a command line similar to the following.

```
` ./transform_visualiser --map-info-file=/usr/local/share/map_transformer/
sample/offset_map.yaml -c -n -t `
```

It will produce an output similar to the following screenshot.



Clicking on one of the map images will display:

- the point you clicked on, in red;
- the equivalent point without transformation in the other map image, in red; and
- the equivalent point after transformation in othe other map image, in green.

You can also provide your own YAML files to the sample application and visualise them.

CHAPTER

FOUR

API

namespace map_transformer

Typedefs

using Point2D = std::pair<float, float>

using CorrespondencePoints = std::vector<*Point2D*>

using Vector2D = std::pair<float, float>

using Triangle = std::tuple<int, int, int>

using TriangleList = std::vector<*Triangle*>

class Transformer

class *map_transformer*::**Transformer**

The Transformer class provides transformation of points between two maps.

The maps are related by a non-linear transformation. In other words, the relation between two equivalent points in one part of the map is not necessarily the same as between two other equivalent points elsewhere in the map.

Public Functions

Transformer()

Create a new empty transformer object.

Transformer(std::string **const** &*yaml_doc*)

Create a new transformer object and load map information from the provided YAML document.

See *Transformer::load()*

Parameters

- `[in] yaml_doc`: The YAML document to load map information from. Must conform to the required format.

Exceptions

- `std::RuntimeError`: if there is an error translating the YAML document.

void **load**(std::string **const** &*yaml_doc*)

Load map information from the provided YAML document.

Pre The Transformer object must be empty (must not already contain map information) before calling *Transformer::load()*. Call *Transformer::reset()* to clear a Transformer instance prior to loading new map information. Transformer object instances are empty when first constructed.

Parameters

- `[in] yaml_doc`: The YAML document to load map information from. Must conform to the required format.

Exceptions

- `std::RuntimeError`: if there is an error translating the YAML document.
- `std::LogicError`: if the Transformer is not empty.

void **`reset`**`()`
Clear any loaded map information.

std::string **`ref_map_name`**`() const`
Get the name of the reference map that is loaded.

Return The name of the reference map, as loaded from the YAML document.

std::string **`ref_map_image_file`**`() const`
Get the path to the image file for the reference map, if there is one.

Return The path to the image file for the reference map, or an empty string if no image file is available.

Exceptions

- `std::LogicError`: if the Transformer has no loaded map information.

Vector2D **`ref_map_size`**`() const`
Get the dimensions of the reference map.

The dimensions are measured in arbitrary units, equivalent to the number of pixels (i.e. the image resolution) in the reference map image if there is one.

Return The dimensions of the reference map.

Exceptions

- `std::LogicError`: if the Transformer has no loaded map information.

std::string **`robot_map_name`**`() const`
Get the name of the robot map that is loaded.

Return A string containing the name of the robot map, as loaded from the YAML document.

Exceptions

- `std::LogicError`: if the Transformer has no loaded map information.

std::string **`robot_map_image_file`**`() const`
Get the path to the image file for the robot map, if there is one.

Return The path to the image file for the robot map, or an empty string if no image file is available.

Exceptions

- `std::LogicError`: if the Transformer has no loaded map information.

Vector2D **robot_map_size**() **const**
Get the dimensions of the robot map.

The dimensions are measured in arbitrary units, equivalent to the number of pixels (i.e. the image resolution) in the robot map image if there is one.

Return The dimensions of the robot map, as a Vector2D.

Exceptions

- `std::LogicError`: if the Transformer has no loaded map information.

Vector2D **robot_map_scale**() **const**
Get the relative scale of the robot map to the reference map.

The robot map may be at a different scale to the reference map. The scale is loaded from the YAML document, and is provided as a value with 1 meaning equal scale, less than 1 meaning the robot map scale is smaller than the reference map, and greater than one meaning the robot map scale is larger than the reference map. The X and Y values can be used as a scaling transformation between the reference map and the robot map.

Return The relative scale of the robot map to the reference map, as an X,Y pair.

Exceptions

- `std::LogicError`: if the Transformer has no loaded map information.

double **robot_map_rotation**() **const**
Get the relative rotation of the robot map around the reference map's origin.

The robot map may be rotated relative to the reference map. The rotation is loaded from the YAML document, and is provided via this member as an angle in radians. This value can be used to construct a rotational transform from the reference map to the robot map and vice versa.

Return The relative rotation of the robot map to the reference map, as an angle in radians.

Exceptions

- `std::LogicError`: if the Transformer has no loaded map information.

Point2D **robot_map_translation**() **const**
Get the relative translation of the robot map from the reference map's origin.

The robot map may be offset from the reference map. The offset is loaded from the YAML document, and is provided via this member as an offset in X and Y. This value can be used to construct a translational transform from the reference map to the robot map and vice versa.

Return The translation from the reference map's origin to the robot map's origin.

Exceptions

- `std::LogicError`: if the Transformer has no loaded map information.

const *CorrespondencePoints* &**ref_map_corr_points**() **const**
Get the list of correspondence points in the reference map.

The correspondence points in the reference map are one-to-one matched to the correspondence points in the robot map. This means that each entry in this list is matched to its same-indexed entry in the list provided by *robot_map_corr_points()*. For example, the point in this list at index 5 is matched to the point in the robot map correspondence points list at index 5.

This list is provided for visualisation and debugging purposes.

Return The list of correspondence points in the reference map.

Exceptions

- `std::LogicError`: if the Transformer has no loaded map information.

const *CorrespondencePoints* &**robot_map_corr_points**() **const**

Get the list of correspondence points in the robot map.

The correspondence points in the robot map are one-to-one matched to the correspondence points in the reference map. This means that each entry in this list is matched to its same-indexed entry in the list provided by *ref_map_corr_points()*. For example, the point in this list at index 5 is matched to the point in the reference map correspondence points list at index 5.

This list is provided for visualisation and debugging purposes.

Return The list of correspondence points in the robot map.

Exceptions

- `std::LogicError`: if the Transformer has no loaded map information.

const *TriangleList* &**triangle_indices**() **const**

Get the list of triangles calculated by the Delaunay triangulation.

This triangle list is provided for visualisation and debugging purposes.

Return The calculated triangles, as indices into the correspondence point lists.

Exceptions

- `std::LogicError`: if the Transformer has no loaded map information.

std::pair<*Point2D*, *Point2D*> **bounding_box**() **const**

Get the bounding box of the two maps.

Returns the bounding box (with one corner at 0, 0) of the two maps. This is the total size of the two maps. If the robot map is aligned with the reference map, it will be the size of the reference/robot map. However, if the robot map is offset, it will be larger, containing the size of an image that is needed to hold both the reference map and the robot map.

Return The bounding box of the two maps.

Exceptions

- `std::LogicError`: if the Transformer has no loaded map information.

Point2D **to_ref**(*Point2D* **const** &*point*) **const**

Transform a point in the robot map to its equivalent point in the reference map.

The transform is performed according to the affine transforms of the Delaunay triangles that were calculated when the map information was loaded, along with the transformation from the robot map to the reference map, if any.

Note If the point lies outside of all Delaunay triangles, it will be transformed only by the relative map transformation. This may or may not be accurate depending on your maps. In the general case, you should assume that any points that lie outside the Delaunay triangulation (i.e. are not enclosed by correspondence points) cannot be transformed accurately.

Return The transformed point in the reference map.

Parameters

- `point`: The point in the robot map to transform.

Exceptions

- `std::RuntimeError`: if an error occurs in the calculations.
- `std::LogicError`: if the Transformer has no loaded map information.

Point2D **to_robot**(*Point2D* **const** &*point*) **const**

Transform a point in the reference map to its equivalent point in the robot map.

The transform is performed according to the affine transforms of the Delaunay triangles that were calculated when the map information was loaded, along with the transformation from the reference map to the robot map, if any.

Note If the point lies outside of all Delaunay triangles, it will be transformed only by the relative map transformation. This may or may not be accurate depending on your maps. In the general case, you should assume that any points that lie outside the Delaunay triangulation (i.e. are not enclosed by correspondence points) cannot be transformed accurately.

Return The transformed point in the robot map.

Parameters

- `point`: The point in the reference map to transform.

Exceptions

- `std::RuntimeError`: if an error occurs in the calculations.
- `std::LogicError`: if the Transformer has no loaded map information.

- genindex

INDEX

M

map_transformer (*C++ type*), 9
map_transformer::CorrespondencePoints (*C++ type*), 9
map_transformer::Point2D (*C++ type*), 9
map_transformer::Transformer (*C++ class*), 9
map_transformer::Transformer::bounding_box (*C++ function*), 12
map_transformer::Transformer::load (*C++ function*), 9
map_transformer::Transformer::ref_map_corr_points (*C++ function*), 11
map_transformer::Transformer::ref_map_image_file (*C++ function*), 10
map_transformer::Transformer::ref_map_name (*C++ function*), 10
map_transformer::Transformer::ref_map_size (*C++ function*), 10
map_transformer::Transformer::reset (*C++ function*), 10
map_transformer::Transformer::robot_map_corr_points (*C++ function*), 12
map_transformer::Transformer::robot_map_image_file (*C++ function*), 10
map_transformer::Transformer::robot_map_name (*C++ function*), 10
map_transformer::Transformer::robot_map_rotation (*C++ function*), 11
map_transformer::Transformer::robot_map_scale (*C++ function*), 11
map_transformer::Transformer::robot_map_size (*C++ function*), 10
map_transformer::Transformer::robot_map_translation (*C++ function*), 11
map_transformer::Transformer::to_ref (*C++ function*), 12
map_transformer::Transformer::to_robot (*C++ function*), 13
map_transformer::Transformer::Transformer (*C++ function*), 9
map_transformer::Transformer::triangle_indices (*C++ function*), 12
map_transformer::Triangle (*C++ type*), 9
map_transformer::TriangleList (*C++ type*), 9
map_transformer::Vector2D (*C++ type*), 9