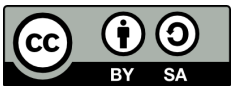
PbMap brief description and user guide.
C++ implementation in `libmrpt-pbmap`

Eduardo Fernández-Moral
efernandezmoral@gmail.com

`http://www.mrpt.org/`

MRPT version: 1.5.3
Document build: August 14, 2017

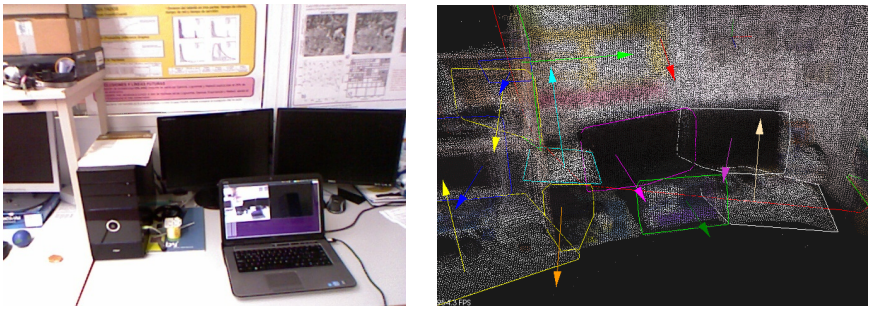


Contents

1 Introduction **3**

2 Setting the parameters **4**

3 Implementation in MRPT (lib `mrpt-pbmap`) **7**
3.1 How to create a simple program to build a PbMap 7



a) b)

Figure 1: Plane-based representation. a) RGB image of the scene. b) Point cloud representation with the segmented planar patches superimposed.

1 Introduction

A PbMap is a highly compact representation of the scene based on a planar model of it. This map representation is proposed to avoid the high memory requirements and processing cost of traditional point cloud representations, which use has raised considerably with the appearance of low cost RGB-D (Kinect like) sensors. A PbMap compresses the point cloud into a set of planar patches, neglecting the non planar data. In this way, it offers an enormous data compression at the cost of losing the non-planar details, but we argue that such details have little importance for some applications, as for building lifelong maps, since only the large planes belonging to the scene structure are persistent over time, while the non-planar, generally small objects are more likely to move or disapear from the scene.

We define a PbMap as a set of planar patches described by geometric features (shape, relative position, etc.) and/or radiometric features (dominant color). It is organized as an annotated, undirected graph, where nodes stand for planar patches and edges connect neighbor planes when the distance between their closest points is under a threshold. This graph structure permits to find efficiently the closest neighbors of a plane, or to select groups of nearby planes representing part of the scene.

The input data to construct a PbMap is given by organized point clouds (depth images) together with poses. The process to build a PbMap can run online from the streaming of RGB-D images. This is possible thanks to efficient algorithms to segmentat planes from organized clouds [3]. The poses of those RGB-D images can be obtained in a number of ways: e.g. visual odometry, robot localization, etc. Thus, the planes are efficiently segmented from the organized point clouds, and these planes are integrated into the PbMap according to the sensor pose, either by updating an already existing plane or by initializing a new one when it is first observed. Figure

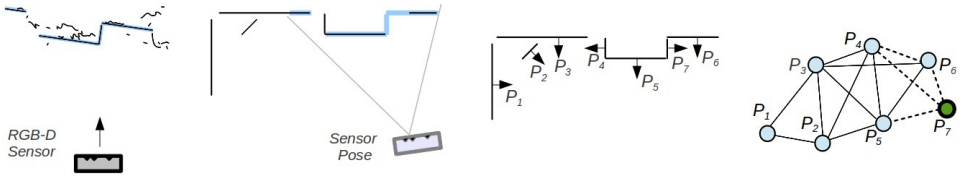


Figure 2: 2D representation of the map construction procedure. a) RGB-D capture with segmentated planes (blue). b) Current PbMap with segmented planes (blue) superimposed according to the sensor pose. c) PbMap updated: the planes updated are highlighted d) PbMap graph updated: the planes updated are highlighted in blue, the new plane P7 is marked in green and, the new edges are represented with dashed lines.

2 depicts a 2D scheme of the PbMap construction process.

An application for recognising previous places using PbMaps is presented in [1]. This method relies on an interpretation tree to efficiently match sets of neighboring planes. Such interpretation tree applies geometric and radiometric restrictions in the form of both unary and binary constraints [2]. The unary constraints check the individual correspondence of two planes by comparing directly their features (e.g. size, color), while the binary constraints validate if two pairs of connected planes present the same geometric relationship (e.g. perpendicularity). For further details refer to our paper *"Fast place recognition with plane-based maps"* [1]. Some results of this work are shown in this video

2 Setting the parameters

There are some heuristic parameters which govern the plane segmentation proces, the PbMap construction and the place recognition and relocalisation methods. These parameters are set in two configuration files: *configPbMap.ini* (for plane segmentation and PbMap construction) which is read by the class `PbMapMaker`; and *configLocaliser.ini* (for place recognition and localisation) which is read by the class `heuristicParamenters`. Each parameter is described below:

Plane segmentation (in *configPbMap.ini*)

- *float dist_threshold* → Set the maximum distance perpendicular to the plane between two 3D-points (default is set to 0.04 m).
- *float angle_threshold* → Set the maximum angle between the normal vectors of two neighbor 3D-points (default is set to 4 deg).
- *float minInliersRate* → Set the minimum number of inliers as a fraction of the image pixels to segment a plane (default is set to 0.005).

Map construction (in *configPbMap.ini*)

Global settings:

- *bool use_color* → Choose wether to add color information or not to the planes (default set to true);
- *int graph_mode* → Choose between establishing edges in the graph according to distance (0) or to visibility (1) (default is set to 0);
- *float proximity_neighbor_planes* → Set the maximum distance between two planar patches to consider them as neighbors (default is set to 1 m);

Parameters to merge two planes representing the same planar surface:

- *float max_cos_normal* → set the maximum angle (actually the minimum angle cosine) between two planes to merge them (default is set to 0.9848 = 10deg);
- *float max_dist_center_plane* → Set the maximum distance between a plane and another plane's center to merge the planes (default is set to 0.1 m);
- *float proximity_threshold* → Set the maximum distance between two planes to merge them (default is set to 0.15 m);

Parameters to infer some simple semantic knowledge to the planar patches:

- *bool inferStructure* → Choose wether to infer if the planes correspond to entities as e.g. floor, ceiling, walls, etc. (default is set to true);
- *bool makeCovisibilityClusters* → Should the PbMapMaker cluster groups of planes according to their co-visibility (default is set to true);

Loop closure:

- *bool detect_loopClosure* → If set to true it runs the PbMapLocaliser functionality in a different thread to detect loop closures or to recognise previous PbMaps (default is set to true)
- *string config_localiser* → Path to the configuration file containing the heuristic parameters which control the place recognition functionality;

Place recognition (in *configLocaliser.ini*) *These parameters are required only if detect_loopClosure=true

Global settings:

- *string path_prev_pbmaps* → Path to previous PbMaps which are to be detected while cunstructing the current PbMap;
- *int min_planes_recognition* → Minimum number of planes to accept a match between two places (defaultis is set to 4);
- *bool use_structure* → Choose whether to employ or not the semantic knowledge inferred to the planes (default is set to true);
- *use_completeness* → Choose whether to differentiate between fully detected planes and partial observations to set different constraints for matching planes (default set to true);

Unary constraints:

- *color_threshold* → Maximum color difference to match two planes (default set to 0.1);
- *elongation_threshold* → Maximum elongation ratio to match two planes ratio (default set to 2.8)
- *area_threshold* → Maximum areas ratio to match two planes (default set to 3.0)
- *area_full_threshold* → Used only if use_completeness is true. Maximum areas ratio to match two planes (default set to 3.0);
- *area_half_threshold* → Used only if use_completeness is true. Maximum areas ratio to match two planes (default set to 2.5);

Binary constraints:

- *angle_threshold* → Maximum difference between the angles formed by of two pairs of planes to match such pairs (default set to 7.0)
- *dist_threshold* → Maximum ratio between the distances of two pairs of planes to match such pairs (default set to 2.0)
- *height_threshold* → Maximum difference between the height of two pairs of planes (almost parallel) to match such pairs (default set to 0.2 m)
- *cos_angle_parallel* → Maximum angle difference to consider two planes almost parallel (default set to 0.985)

3 Implementation in MRPT (lib `mrpt-pbmap`)

This library implements the functionality to build Plane-based Maps (PbMaps) from a set of point clouds plus their corresponding poses, which might be given by e.g. the odometry of a robot. Application examples

Two application examples have been created for creating PbMaps (`pbmap_example`) and for visualising them (`pbmap_visualizer`). To build the example within MRPT, the Cmake option BUILD_EXAMPLES must be set to ON (default is OFF).

3.1 How to create a simple program to build a PbMap

The central class in mrpt-pbmap is PbMapMaker. This class creates its own thread in which it runs. The input data is passed through its public member frameQueue which stores a vector of pairs of point cloud plus pose. Thus, in order to create a PbMap, the user just need to create an object PbMapMaker and fill the vector frameQueue:

```
#include <mrpt/pbmap.h>

using namespace mrpt::pbmap;
using namespace std;

int main(int argc, char**argv)
{
  //  Create a PbMapMaker object specifying the configPbMap.ini
  PbMapMaker pbmap_maker("path_to/config_files/pbmap/configPbMap.ini")
      ;

  // While certain condition is fulfilled (e.g. while exploration)
  while (...) {
    ...

    // Get dupla point_cloud + pose
    frameRGBDandPose cloudAndPose;
    pcl::copyPointCloud(point_cloud, *cloudAndPose.cloudPtr);
    cloudAndPose.pose << pose;

    // Detect planes and build PbMap
    pbmap_maker.frameQueue.push_back(cloudAndPose);

    ...
  }
}
```

Also, loop closure is run if it was activated in the configuration file configPbMap.ini. Note that we treat loop closure and place recognition in an equivalent manner, with the only difference that place recognition implies searching previous PbMaps with yet no relation with the current one being built.

References

[1] E. Fernández-Moral, W. Mayol-Cuevas, V. Arévalo, and J. González-Jiménez. Fast place recognition with plane-based maps. In *Robotics and Automation (ICRA), to appear in 2013 IEEE International Conference on*, pages 5210–5215. IEEE, 2013.

[2] W. E. L Grimson. *Object Recognition by Computer - The role of Geometric Constraints*. MIT Press, Cambridge, MA, 2012.

[3] D. Holz and S. Behnke. Fast range image segmentation and smoothing using approximate surface reconstruction and region growing. *Proceedings of the International Conference on Intelligent Autonomous Systems (IAS), Jeju Island, Korea*, 2012.