1. 简介

rtabmap，全称Real-Time Appearance-Based Mapping，基于外观的实时地图构建，传感器可以是深度，双目相机或者激光雷达。
它的回环检测是采用词袋模型进行判断的，而词典是采用在线增量式创建的，而不是采用已经训练好的。
rtabmap的官网:http://introlab.github.io/rtabmap/

2. 使用步骤

不单独介绍rtabmap的使用，结合ros一起介绍它的使用。ros中有个包叫rtabmap_ros，这个包把rtabmap的接口使用转换成了ros消息的方式(如topic,service)，
我们只要把传感器的数据以topic的方式发给rtabmap_ros，rtabmap_ros就会把建图的数据也以topic的方式发出来。

rtabmap_ros的官方wiki:
http://wiki.ros.org/rtabmap_roshttp://wiki.ros.org/rtabmap_ros

rtabmap_ros安装方法很简单:
sudo apt-get install ros-kinetic-rtabmap
sudo apt-get install ros-kinetic-rtabmap-ros

demo运行，可查看官方wiki。

3. 框架介绍

rtabmap_ros主要有两个node:
计算里程计的node：rgbd_odometry，stereo_odometry，icp_odometry
建图的node：rtabmap

4. rgbd_odometry

rgbd_odometry节点的相关启动参数和topic：

Arguments

- `"--params"`: Show RTAB-Map's parameters related to this node and exit.

Parameters

`~frame_id` (`string`, default: `"base_link"`)

- The frame attached to the mobile base.

`~odom_frame_id` (`string`, default: `"odom"`)

- The odometry frame.

`~publish_tf` (`bool`, default: `"true"`)

- Publish TF from /odom to /base_link.

`~tf_prefix` (`bool`, default: `""`)

- Prefix to add to generated tf.

`~wait_for_transform` (`bool`, default: `"false"`)

- Wait (maximum 1 sec) for transform when a tf transform is not still available.

`~initial_pose` (`string`, default: `""`)

- The initial pose of the odometry. Format: `"x y z roll pitch yaw"`.

`~queue_size` (`int`, default: 10)

- Size of message queue for each synchronized topic.

`~depth_cameras` (`int`, default: 1)

- Number of RGB-D cameras to use. Well for now, only a maximum of 2 cameras can be synchronized at the same time. If > 1, the rgb, depth and camera_info topics should contain the camera index starting with 0. For example, with 2 cameras, you should set `rgb0/image`, `rgb0/camera_info`, `depth0/image`, `rgb1/image`, `rgb1/camera_info` and `depth1/image` topics.

Subscribed Topics

`rgb/image` (sensor_msgs/Image)

- RGB/Mono rectified image.

`rgb/camera_info` (sensor_msgs/CameraInfo)

- RGB camera metadata.

`depth/image` (sensor_msgs/Image)

- Registered depth image.

Services

`reset_odom` (std_srvs/Empty)

- Restart odometry to identity transformation.

`reset_odom_to_pose` (rtabmap_ros/ResetPose)

- Restart odometry to specified transformation. Format: `"x y z roll pitch yaw"`.

`pause_odom` (std_srvs/Empty)

- Pause odometry.

`resume_odom` (std_srvs/Empty)

- Resume odometry.

Published Topics

`odom` (nav_msgs/Odometry)

- Odometry stream. Send null odometry if cannot be computed. //发布里程计数据

`odom_info` (rtabmap_ros/OdomInfo)

- Odometry info stream. RTAB-Map's parameter `Odom/FillInfoData` should be true to fill features information (default is `true`).

`odom_last_frame` (sensor_msgs/PointCloud2)

- 3D features of the last frame used to estimate the transformation.

`odom_local_map` (sensor_msgs/PointCloud2)

- Local map of 3D features used as reference to estimate the transformation. Only published if RTAB-Map's parameter `Odom/Strategy=0`.

Required tf Transforms

`base_link` → `<the frame attached to sensors of incoming data>`

- usually a fixed value, broadcast periodically by a robot_state_publisher, or a `tf` static_transform_publisher.

Provided tf Transforms

`odom` → `base_link`

- the current estimate of the robot's pose within the odometry frame. //发布tf转换

5. rtabmap

rtabmap的启动参数和相关topic：

Arguments

- `"--delete_db_on_start"` or `"-d"`: Delete the database before starting, otherwise the previous mapping session is loaded.
- `"--udebug"`: Show RTAB-Map's debug/info/warning/error logs.

- "--uinfo": Show RTAB-Map's info/warning/error logs.
- "--params": Show RTAB-Map's parameters related to this node and exit.

Subscribed Topics

odom (nav_msgs/Odometry)

- Odometry stream. Required if parameters subscribe_depth or subscribe_stereo are true **and**odom_frame_id is not set.

rgb/image (sensor_msgs/Image)

- RGB/Mono image. Should be rectified when subscribe_depth is true. Not required if parameter subscribe_stereo is true (use left/image_rect instead).

rgb/camera_info (sensor_msgs/CameraInfo)

- RGB camera metadata. Not required if parameter subscribe_stereo is true (use left/camera_info instead).

depth/image (sensor_msgs/Image)

- Registered depth image. Required if parameter subscribe_depth is true.

scan (sensor_msgs/LaserScan)

- Laser scan stream. Required if parameter subscribe_scan is true.

scan_cloud (sensor_msgs/PointCloud2)

- Laser scan point cloud stream. Required if parameter subscribe_scan_cloud is true.

left/image_rect (sensor_msgs/Image)

- Left RGB/Mono rectified image. Required if parameter subscribe_stereo is true.

left/camera_info (sensor_msgs/CameraInfo)

- Left camera metadata. Required if parameter subscribe_stereo is true.

right/image_rect (sensor_msgs/Image)

- Right Mono rectified image. Required if parameter subscribe_stereo is true.

right/camera_info (sensor_msgs/CameraInfo)

- Right camera metadata. Required if parameter subscribe_stereo is

true.

`goal` (geometry_msgs/PoseStamped)

- ***Planning*** Plan a path to reach this goal using the current online map. The goal should be close enough to the map's graph to be accepted (see RTAB-Map's parameter `RGBD/LocalRadius`). Note that only nodes in Working Memory are used, for exploration it may be ok, but it would be better to use service `set_goal` if you are coming back in a previously mapped area. See ***Planning*** published topics for corresponding generated outputs.

Published Topics

`info` (rtabmap_ros/Info)

- RTAB-Map's info.

`mapData` (rtabmap_ros/MapData)

- RTAB-Map's graph and latest node data.

`mapGraph` (rtabmap_ros/MapGraph)

- RTAB-Map's graph only.

`grid_map` (nav_msgs/OccupancyGrid)

- ***Mapping*** Occupancy grid generated with laser scans. Use parameters with prefixes `map_` and `grid_` below.

`proj_map` (nav_msgs/OccupancyGrid)

- ***Mapping*** Occupancy grid generated from projection of the 3D point clouds on the ground. Use parameters with prefixes `map_`, `grid_` and `proj_` below.

`cloud_map` (sensor_msgs/PointCloud2)

- ***Mapping*** 3D point cloud generated with the 3D point clouds. Use parameters with prefixes `map_` and `cloud_` below.

`scan_map` (sensor_msgs/PointCloud2)

- ***Mapping*** 3D point cloud generated with the 2D scans or 3D scans. Use parameters with prefixes `map_` and `scan_`below.

`labels` (visualization_msgs/MarkerArray)

- Convenient way to show graph's labels in RVIZ.

`global_path` (nav_msgs/Path)

- ***Planning*** Poses of the planned global path. Published only once for each path planned.

`local_path` (nav_msgs/Path)

- ***Planning*** Upcoming local poses corresponding to those of the global path. Published on every map update.

`goal_reached` (std_msgs/Bool)

- ***Planning*** Status message if the goal is successfully reached or not.

`goal_out` (geometry_msgs/PoseStamped)

- ***Planning*** Current metric goal sent from the rtabmap's topological planner. For example, this can be connected to `move_base_simple/goal` topic of move_base.

Services

`get_map` (rtabmap_ros/GetMap)

- Call this service to get the map data

`get_grid_map` (nav_msgs/GetMap)

- Call this service to get the occupancy grid built from laser scans

`get_proj_map` (nav_msgs/GetMap)

- Call this service to get the occupancy grid built from the projection of the 3D point clouds

`publish_map` (rtabmap_ros/PublishMap)

- Call this service to publish the map data

`list_labels` (rtabmap_ros/ListLabels)

- Get current labels of the graph.

`update_parameters` (std_srvs/Empty)

- The node will update with current parameters of the rosparam server

`reset` (std_srvs/Empty)

- Delete the map

`pause` (std_srvs/Empty)

- Pause mapping

`resume` (std_srvs/Empty)

- Resume mapping

`trigger_new_map` (std_srvs/Empty)

- The node will begin a new map

`backup` (std_srvs/Empty)

- Backup the database to "`database_path`.back" (default

`~/.ros/rtabmap.db.back`)

`set_mode_localization` (std_srvs/Empty)

- Set localization mode

`set_mode_mapping` (std_srvs/Empty)

- Set mapping mode

`set_label` (rtabmap_ros/SetLabel)

- Set a label to latest node or a specified node.

`set_goal` (rtabmap_ros/SetGoal)

- ***Planning*** Set a topological goal (a node id or a node label in the graph). Plan a path to reach this goal using the whole map contained in memory (including nodes in Long-Term Memory). See ***Planning*** published topics for corresponding generated outputs.

`octomap_full` (octomap_msgs/GetOctomap)

- Get an octomap. *Available only if rtabmap_ros is built with octomap*.

`octomap_binary` (octomap_msgs/GetOctomap)

- Get an octomap. *Available only if rtabmap_ros is built with octomap*.

Parameters

`~subscribe_depth` (`bool`, default: `"true"`)

- Subscribe to depth image

`~subscribe_scan` (`bool`, default: `"false"`)

- Subscribe to laser scan

`~subscribe_scan_cloud` (`bool`, default: `"false"`)

- Subscribe to laser scan point cloud

`~subscribe_stereo` (`bool`, default: `"false"`)

- Subscribe to stereo images

`~frame_id` (`string`, default: `"base_link"`)

- The frame attached to the mobile base.

`~map_frame_id` (`string`, default: `"map"`)

- The frame attached to the map.

`~odom_frame_id` (`string`, default: `""`)

- The frame attached to odometry. If empty, rtabmap will subscribe to `odom` topic to get odometry. If set, odometry is got from tf (in this case, a covariance of 1 is used).

~queue_size (int, default: 10)

- Size of message queue for each synchronized topic.

~publish_tf (bool, default: "true")

- Publish TF from /map to /odom.

~tf_delay (double, default: 0.05)

- Rate at which the TF from /map to /odom is published (20 Hz).

~tf_prefix (string, default: "")

- Prefix to add to generated tf.

~wait_for_transform (bool, default: "true")

- Wait (maximum wait_for_transform_duration sec) for transform when a tf transform is not still available.

~wait_for_transform_duration (double, default: 0.1)

- Wait duration for wait_for_transform.

~config_path (string, default: "")

- Path of a config files containing RTAB-Map's parameters. Parameters set in the launch file overwrite those in the config file.

~database_path (string, default: "~/.ros/rtabmap.db")

- Path of the RTAB-Map's database.

~gen_scan (bool, default: "false")

- Generate laser scans from depth images (using the middle horizontal line of the depth image). Not generated if subscribe_scan or subscribe_scan_cloud are true.

~gen_scan_max_depth (double, default: 4.0)

- Maximum depth of the laser scans generated.

~approx_sync (bool, default: "false")

- Use approximate time synchronization of input messages. If false, note that the odometry input must have also exactly the same timestamps than the input images.

~depth_cameras (int, default: 1)

- Number of RGB-D cameras to use. Well for now, a maximum of 2 cameras can be synchronized at the same time. If > 1, the rgb, depth and camera_info topics should contain the camera index starting with 0. For example, if we have 2 cameras, you should set rgb0/image, rgb0/camera_info, depth0/image, rgb1/image,

`rgb1/camera_info` and `depth1/image` topics. `subscribe_depth` should be `true` as well.

`~use_action_for_goal` (`bool`, default: `"false"`)

- ***Planning*** Use actionlib to send the metric goals to move_base. The advantage over just connecting `goal_out` to `move_base_simple/goal` is that rtabmap can have a feedback if the goal is reached or if move_base has failed. See move_base Action API for more info.

`~map_filter_radius` (`double`, default: 0.5)

- ***Mapping*** Filter nodes before creating the maps. Only load data for one node in the filter radius (the latest data is used) up to filter angle (`map_filter_angle`). Set to 0.0 to disable node filtering. Used for all published maps.

`~map_filter_angle` (`double`, default: 30.0)

- ***Mapping*** Angle used when filtering nodes before creating the maps. See also `map_filter_radius`. Used for all published maps.

`~map_cleanup` (`bool`, default: `"true"`)

- ***Mapping*** If there is no subscription to any map `cloud_map`, `grid_map` or `proj_map`, clear the corresponding data.

`~latch` (`bool`, default: `"true"`)

- ***Mapping*** If true, the last message published on the map topics will be saved and sent to new subscribers when they connect.

`~cloud_decimation` (`int`, default: 4)

- ***Mapping*** Image decimation before creating clouds added to cloud map. Set 1 to disable decimation. Used for `cloud_map` published topic.

`~cloud_max_depth` (`double`, default: 4.0)

- ***Mapping*** Maximum depth of the clouds added to cloud map. Set 0.0 to maximum depth filtering. Used for `cloud_map` published topic.

`~cloud_voxel_size` (`double`, default: 0.05)

- ***Mapping*** Voxel size used to filter each cloud added to cloud map. Set 0.0 to disable voxel filtering. Used for `cloud_map` published topic.

`~ cloud_floor_culling_height` (`double`, default: 0.0)

- ***Mapping*** Filter the floor at the specified height (m). Used for `cloud_map` published topic.

`~cloud_output_voxelized` (`bool`, default: `"false"`)

- ***Mapping*** Do a final voxel filtering after all clouds are assembled. Used for `cloud_map` published topic.

`~cloud_frustum_culling` (`bool`, default: `"false"`)

- ***Mapping*** Filter the point cloud in the frustum defined by the last camera pose. Used for `cloud_map` published topic.

`~cloud_noise_filtering_radius` (`double`, default: 0)

- ***Mapping*** Filter points that have less neighbors than `cloud_noise_filtering_min_neighbors` in the radius `cloud_noise_filtering_radius` (0=disabled). Used for `cloud_map` published topic.

`~cloud_noise_filtering_min_neighbors` (`double`, default: 5)

- ***Mapping*** Filter points that have less neighbors than `cloud_noise_filtering_min_neighbors` in the radius `cloud_noise_filtering_radius` (0=disabled). Used for `cloud_map` published topic.

`~scan_voxel_size` (`double`, default: 0.05)

- ***Mapping*** Voxel size used to filter each scan added to scan map. Set 0.0 to disable voxel filtering. Used for `scan_map` published topic.

`~scan_output_voxelized` (`bool`, default: `"false"`)

- ***Mapping*** Do a final voxel filtering after all scans are assembled. Used for `scan_map` published topic.

`~grid_cell_size` (`double`, default: 0.05)

- ***Mapping*** Grid cell size (m). Used for `grid_map` and `proj_map` published topics.

`~grid_size` (`double`, default: 0)

- ***Mapping*** Initial size of the map (m). If 0, the map will grow as new data are added. If set, the map will still grow when the initial size is reached. Used for `grid_map` and `proj_map` published topics.

`~grid_eroded` (`bool`, default: `"false"`)

- ***Mapping*** Filter obstacle cells surrounded by empty space. Used for `grid_map` and `proj_map` published topics.

`~grid_unknown_space_filled` (`bool`, default: `"false"`)

- ***Mapping*** Fill empty space. Used for `grid_map` published topic.

`~proj_max_ground_angle` (`double`, default: 45)

- ***Mapping*** Maximum angle (degrees) between point's normal to ground's normal to label it as ground. Points with higher angle difference are considered as obstacles. Used for `proj_map` published topic.

`~proj_min_cluster_size` (`int`, default: 20)

- ***Mapping*** Minimum cluster size to project the points. The distance between clusters is defined by 2*`grid_cell_size`. Used for `proj_map` published topic.

`~proj_max_height` (`double`, default: 2.0)

- ***Mapping*** Maximum height of points used for projection. Used for `proj_map` published topic.

Required tf Transforms

`base_link` → `<the frame attached to sensors of incoming data>`

- usually a fixed value, broadcast periodically by a robot_state_publisher, or a `tf` static_transform_publisher.

`odom` → `base_link`

- usually provided by the odometry system (e.g., the driver for the mobile base).

Provided tf Transforms

`map` → `odom`

- the current odometry correction.

6. rtabmap的参数介绍

```
    // Rtabmap parameters
    RTABMAP_PARAM(Rtabmap, PublishStats,                bool, true, "Publishing statistics.");
    RTABMAP_PARAM(Rtabmap, PublishLastSignature,        bool, true, "Publishing last signature.");
    RTABMAP_PARAM(Rtabmap, PublishPdf,                  bool, true, "Publishing pdf.");
```

```
    RTABMAP_PARAM(Rtabmap, PublishLikelihood,            bool, true,
"Publishing likelihood.");
    RTABMAP_PARAM(Rtabmap, PublishRAMUsage,              bool, false,
"Publishing RAM usage in statistics (may add a small overhead to get info
from the system).");
    RTABMAP_PARAM(Rtabmap, ComputeRMSE,                  bool, true,
"Compute root mean square error (RMSE) and publish it in statistics, if
ground truth is provided.");
    RTABMAP_PARAM(Rtabmap, SaveWMState,                  bool, false, "Save
working memory state after each update in statistics.");
    RTABMAP_PARAM(Rtabmap, TimeThr,                      float, 0,
"Maximum time allowed for the detector (ms) (0 means infinity).");
    RTABMAP_PARAM(Rtabmap, MemoryThr,                    int, 0,
"Maximum signatures in the Working Memory (ms) (0 means infinity).");
    RTABMAP_PARAM(Rtabmap, DetectionRate,                float, 1,
"Detection rate (Hz). RTAB-Map will filter input images to satisfy this
rate.");
    RTABMAP_PARAM(Rtabmap, ImageBufferSize,          unsigned int, 1, "Data
buffer size (0 min inf).");
    RTABMAP_PARAM(Rtabmap, CreateIntermediateNodes,      bool, false,
uFormat("Create intermediate nodes between loop closure detection. Only used
when %s>0.", kRtabmapDetectionRate().c_str()));
    RTABMAP_PARAM_STR(Rtabmap, WorkingDirectory,         "",
"Working directory.");
    RTABMAP_PARAM(Rtabmap, MaxRetrieved,             unsigned int, 2,
"Maximum locations retrieved at the same time from LTM.");
    RTABMAP_PARAM(Rtabmap, StatisticLogsBufferedInRAM,   bool, true,
"Statistic logs buffered in RAM instead of written to hard drive after each
iteration.");
    RTABMAP_PARAM(Rtabmap, StatisticLogged,              bool, false,
"Logging enabled.");
    RTABMAP_PARAM(Rtabmap, StatisticLoggedHeaders,       bool, true,  "Add
column header description to log files.");
    RTABMAP_PARAM(Rtabmap, StartNewMapOnLoopClosure,     bool, false, "Start
a new map only if there is a global loop closure with a previous map.");
    RTABMAP_PARAM(Rtabmap, ImagesAlreadyRectified,       bool, true,
"Images are already rectified. By default RTAB-Map assumes that received
images are rectified. If they are not, they can be rectified by RTAB-Map if
this parameter is false.");

    // Hypotheses selection
    RTABMAP_PARAM(Rtabmap, LoopThr,           float, 0.11,      "Loop
closing threshold.");
```

```
    RTABMAP_PARAM(Rtabmap, LoopRatio,         float, 0,         "The loop closure hypothesis must be over LoopRatio x lastHypothesisValue.");

    // Memory
    RTABMAP_PARAM(Mem, RehearsalSimilarity,         float, 0.6,  "Rehearsal similarity.");
    RTABMAP_PARAM(Mem, ImageKept,                   bool, false,    "Keep raw images in RAM.");
    RTABMAP_PARAM(Mem, BinDataKept,                 bool, true,     "Keep binary data in db.");
    RTABMAP_PARAM(Mem, RawDescriptorsKept,          bool, true,     "Raw descriptors kept in memory.");
    RTABMAP_PARAM(Mem, MapLabelsAdded,              bool, true,     "Create map labels. The first node of a map will be labelled as \"map#\" where # is the map ID.");
    RTABMAP_PARAM(Mem, SaveDepth16Format,           bool, false,    "Save depth image into 16 bits format to reduce memory used. Warning: values over ~65 meters are ignored (maximum 65535 millimeters).");
    RTABMAP_PARAM(Mem, NotLinkedNodesKept,          bool, true,     "Keep not linked nodes in db (rehearsed nodes and deleted nodes).");
    RTABMAP_PARAM(Mem, IntermediateNodeDataKept,    bool, false,    "Keep intermediate node data in db.");
    RTABMAP_PARAM(Mem, STMSize,                     unsigned int, 10, "Short-term memory size.");
    RTABMAP_PARAM(Mem, IncrementalMemory,           bool, true,     "SLAM mode, otherwise it is Localization mode.");
    RTABMAP_PARAM(Mem, ReduceGraph,                 bool, false,    "Reduce graph. Merge nodes when loop closures are added (ignoring those with user data set).");
    RTABMAP_PARAM(Mem, RecentWmRatio,               float, 0.2,     "Ratio of locations after the last loop closure in WM that cannot be transferred.");
    RTABMAP_PARAM(Mem, TransferSortingByWeightId,   bool, false,    "On transfer, signatures are sorted by weight->ID only (i.e. the oldest of the lowest weighted signatures are transferred first). If false, the signatures are sorted by weight->Age->ID (i.e. the oldest inserted in WM of the lowest weighted signatures are transferred first). Note that retrieval updates the age, not the ID.");
    RTABMAP_PARAM(Mem, RehearsalIdUpdatedToNewOne,  bool, false,    "On merge, update to new id. When false, no copy.");
    RTABMAP_PARAM(Mem, RehearsalWeightIgnoredWhileMoving, bool, false, "When the robot is moving, weights are not updated on rehearsal.");
    RTABMAP_PARAM(Mem, GenerateIds,                 bool, true,
```

```
"True=Generate location IDs, False=use input image IDs.");
    RTABMAP_PARAM(Mem, BadSignaturesIgnored,        bool, false,    "Bad
signatures are ignored.");
    RTABMAP_PARAM(Mem, InitWMWithAllNodes,          bool, false,
"Initialize the Working Memory with all nodes in Long-Term Memory. When
false, it is initialized with nodes of the previous session.");
    RTABMAP_PARAM(Mem, DepthAsMask,                 bool, true,     "Use
depth image as mask when extracting features for vocabulary.");
    RTABMAP_PARAM(Mem, ImagePreDecimation,          int, 1,         "Image
decimation (>=1) before features extraction. Negative decimation is done
from RGB size instead of depth size (if depth is smaller than RGB, it may be
interpolated depending of the decimation value).");
    RTABMAP_PARAM(Mem, ImagePostDecimation,         int, 1,         "Image
decimation (>=1) of saved data in created signatures (after features
extraction). Decimation is done from the original image. Negative decimation
is done from RGB size instead of depth size (if depth is smaller than RGB,
it may be interpolated depending of the decimation value).");
    RTABMAP_PARAM(Mem, CompressionParallelized,     bool, true,
"Compression of sensor data is multi-threaded.");
    RTABMAP_PARAM(Mem, LaserScanDownsampleStepSize, int, 1,         "If > 1,
downsample the laser scans when creating a signature.");
    RTABMAP_PARAM(Mem, LaserScanVoxelSize,          float, 0.0,
uFormat("If > 0 m, voxel filtering is done on laser scans when creating a
signature. If the laser scan had normals, they will be removed. To recompute
the normals, make sure to use \"%s\" or \"%s\" parameters.",
kMemLaserScanNormalK().c_str(),
kMemLaserScanNormalRadius().c_str()).c_str());
    RTABMAP_PARAM(Mem, LaserScanNormalK,            int, 0,         "If > 0
and laser scans don't have normals, normals will be computed with K search
neighbors when creating a signature.");
    RTABMAP_PARAM(Mem, LaserScanNormalRadius,       int, 0,         "If > 0
m and laser scans don't have normals, normals will be computed with radius
search neighbors when creating a signature.");
    RTABMAP_PARAM(Mem, UseOdomFeatures,             bool, true,     "Use
odometry features.");

    // KeypointMemory (Keypoint-based)
    RTABMAP_PARAM(Kp, NNStrategy,                int, 1,
"kNNFlannNaive=0, kNNFlannKdTree=1, kNNFlannLSH=2, kNNBruteForce=3,
kNNBruteForceGPU=4");
    RTABMAP_PARAM(Kp, IncrementalDictionary,    bool, true,    "");
    RTABMAP_PARAM(Kp, IncrementalFlann,         bool, true,    uFormat("When
using FLANN based strategy, add/remove points to its index without always
```

```
rebuilding the index (the index is built only when the dictionary increases
of the factor \"%s\" in size).", kKpFlannRebalancingFactor().c_str()));
    RTABMAP_PARAM(Kp, FlannRebalancingFactor,   float, 2.0,
uFormat("Factor used when rebuilding the incremental FLANN index (see
\"%s\"). Set <=1 to disable.", kKpIncrementalFlann().c_str()));
    RTABMAP_PARAM(Kp, MaxDepth,                 float, 0,     "Filter
extracted keypoints by depth (0=inf).");
    RTABMAP_PARAM(Kp, MinDepth,                 float, 0,     "Filter
extracted keypoints by depth.");
    RTABMAP_PARAM(Kp, MaxFeatures,              int, 500,     "Maximum
features extracted from the images (0 means not bounded, <0 means no
extraction).");
    RTABMAP_PARAM(Kp, BadSignRatio,             float, 0.5,   "Bad signature
ratio (less than Ratio x AverageWordsPerImage = bad).");
    RTABMAP_PARAM(Kp, NndrRatio,                float, 0.8,   "NNDR ratio (A
matching pair is detected, if its distance is closer than X times the
distance of the second nearest neighbor.)");
#ifndef RTABMAP_NONFREE
#ifdef RTABMAP_OPENCV3
    // OpenCV 3 without xFeatures2D module doesn't have BRIEF
    RTABMAP_PARAM(Kp, DetectorStrategy,         int, 8,       "0=SURF 1=SIFT
2=ORB 3=FAST/FREAK 4=FAST/BRIEF 5=GFTT/FREAK 6=GFTT/BRIEF 7=BRISK 8=GFTT/ORB
9=KAZE.");
#else
    RTABMAP_PARAM(Kp, DetectorStrategy,         int, 6,       "0=SURF 1=SIFT
2=ORB 3=FAST/FREAK 4=FAST/BRIEF 5=GFTT/FREAK 6=GFTT/BRIEF 7=BRISK 8=GFTT/ORB
9=KAZE.");
#endif
#else
    RTABMAP_PARAM(Kp, DetectorStrategy,         int, 6,       "0=SURF 1=SIFT
2=ORB 3=FAST/FREAK 4=FAST/BRIEF 5=GFTT/FREAK 6=GFTT/BRIEF 7=BRISK 8=GFTT/ORB
9=KAZE.");
#endif
    RTABMAP_PARAM(Kp, TfIdfLikelihoodUsed,      bool, true,   "Use of the
td-idf strategy to compute the likelihood.");
    RTABMAP_PARAM(Kp, Parallelized,             bool, true,   "If the
dictionary update and signature creation were parallelized.");
    RTABMAP_PARAM_STR(Kp, RoiRatios,       "0.0 0.0 0.0 0.0", "Region of
interest ratios [left, right, top, bottom].");
    RTABMAP_PARAM_STR(Kp, DictionaryPath,       "",           "Path of the
pre-computed dictionary");
    RTABMAP_PARAM(Kp, NewWordsComparedTogether, bool, true,   "When adding
new words to dictionary, they are compared also with each other (to detect
```

```
same words in the same signature).");
    RTABMAP_PARAM(Kp, SubPixWinSize,            int, 3,       "See
cv::cornerSubPix().");
    RTABMAP_PARAM(Kp, SubPixIterations,         int, 0,       "See
cv::cornerSubPix(). 0 disables sub pixel refining.");
    RTABMAP_PARAM(Kp, SubPixEps,                double, 0.02, "See
cv::cornerSubPix().");
    RTABMAP_PARAM(Kp, GridRows,                 int, 1,
uFormat("Number of rows of the grid used to extract uniformly \"%s / grid
cells\" features from each cell.", kKpMaxFeatures().c_str()));
    RTABMAP_PARAM(Kp, GridCols,                 int, 1,
uFormat("Number of columns of the grid used to extract uniformly \"%s / grid
cells\" features from each cell.", kKpMaxFeatures().c_str()));

    //Database
    RTABMAP_PARAM(DbSqlite3, InMemory,     bool, false,      "Using database
in the memory instead of a file on the hard disk.");
    RTABMAP_PARAM(DbSqlite3, CacheSize, unsigned int, 10000, "Sqlite cache
size (default is 2000).");
    RTABMAP_PARAM(DbSqlite3, JournalMode,  int, 3,           "0=DELETE,
1=TRUNCATE, 2=PERSIST, 3=MEMORY, 4=OFF (see sqlite3 doc : \"PRAGMA
journal_mode\")");
    RTABMAP_PARAM(DbSqlite3, Synchronous,  int, 0,           "0=OFF,
1=NORMAL, 2=FULL (see sqlite3 doc : \"PRAGMA synchronous\")");
    RTABMAP_PARAM(DbSqlite3, TempStore,    int, 2,           "0=DEFAULT,
1=FILE, 2=MEMORY (see sqlite3 doc : \"PRAGMA temp_store\")");

    // Keypoints descriptors/detectors
    RTABMAP_PARAM(SURF, Extended,          bool, false,  "Extended
descriptor flag (true - use extended 128-element descriptors; false - use
64-element descriptors).");
    RTABMAP_PARAM(SURF, HessianThreshold,  float, 500,   "Threshold for
hessian keypoint detector used in SURF.");
    RTABMAP_PARAM(SURF, Octaves,           int, 4,       "Number of pyramid
octaves the keypoint detector will use.");
    RTABMAP_PARAM(SURF, OctaveLayers,      int, 2,       "Number of octave
layers within each octave.");
    RTABMAP_PARAM(SURF, Upright,           bool, false,  "Up-right or
rotated features flag (true - do not compute orientation of features; false
- compute orientation).");
    RTABMAP_PARAM(SURF, GpuVersion,        bool, false,  "GPU-SURF: Use GPU
version of SURF. This option is enabled only if OpenCV is built with CUDA
and GPUs are detected.");
```

```
    RTABMAP_PARAM(SURF, GpuKeypointsRatio,  float, 0.01, "Used with SURF GPU.");

    RTABMAP_PARAM(SIFT, NFeatures,          int, 0,       "The number of best features to retain. The features are ranked by their scores (measured in SIFT algorithm as the local contrast).");
    RTABMAP_PARAM(SIFT, NOctaveLayers,      int, 3,       "The number of layers in each octave. 3 is the value used in D. Lowe paper. The number of octaves is computed automatically from the image resolution.");
    RTABMAP_PARAM(SIFT, ContrastThreshold, double, 0.04, "The contrast threshold used to filter out weak features in semi-uniform (low-contrast) regions. The larger the threshold, the less features are produced by the detector.");
    RTABMAP_PARAM(SIFT, EdgeThreshold,      double, 10,   "The threshold used to filter out edge-like features. Note that the its meaning is different from the contrastThreshold, i.e. the larger the edgeThreshold, the less features are filtered out (more features are retained).");
    RTABMAP_PARAM(SIFT, Sigma,              double, 1.6,  "The sigma of the Gaussian applied to the input image at the octave #0. If your image is captured with a weak camera with soft lenses, you might want to reduce the number.");

    RTABMAP_PARAM(BRIEF, Bytes,             int, 32,      "Bytes is a length of descriptor in bytes. It can be equal 16, 32 or 64 bytes.");

    RTABMAP_PARAM(FAST, Threshold,           int, 20,      "Threshold on difference between intensity of the central pixel and pixels of a circle around this pixel.");
    RTABMAP_PARAM(FAST, NonmaxSuppression,  bool, true,   "If true, non-maximum suppression is applied to detected corners (keypoints).");
    RTABMAP_PARAM(FAST, Gpu,                 bool, false,  "GPU-FAST: Use GPU version of FAST. This option is enabled only if OpenCV is built with CUDA and GPUs are detected.");
    RTABMAP_PARAM(FAST, GpuKeypointsRatio,  double, 0.05, "Used with FAST GPU.");
    RTABMAP_PARAM(FAST, MinThreshold,        int, 7,       "Minimum threshold. Used only when FAST/GridRows and FAST/GridCols are set.");
    RTABMAP_PARAM(FAST, MaxThreshold,        int, 200,     "Maximum threshold. Used only when FAST/GridRows and FAST/GridCols are set.");
    RTABMAP_PARAM(FAST, GridRows,            int, 4,       "Grid rows (0 to disable). Adapts the detector to partition the source image into a grid and detect points in each cell.");
    RTABMAP_PARAM(FAST, GridCols,            int, 4,       "Grid cols (0 to
```

```
disable). Adapts the detector to partition the source image into a grid and
detect points in each cell.");

    RTABMAP_PARAM(GFTT, QualityLevel,      double, 0.001, "");
    RTABMAP_PARAM(GFTT, MinDistance,       double, 3,     "");
    RTABMAP_PARAM(GFTT, BlockSize,         int, 3,        "");
    RTABMAP_PARAM(GFTT, UseHarrisDetector, bool, false,   "");
    RTABMAP_PARAM(GFTT, K,                 double, 0.04, "");

    RTABMAP_PARAM(ORB, ScaleFactor,   float, 1.2,  "Pyramid decimation
ratio, greater than 1. scaleFactor==2 means the classical pyramid, where
each next level has 4x less pixels than the previous, but such a big scale
factor will degrade feature matching scores dramatically. On the other hand,
too close to 1 scale factor will mean that to cover certain scale range you
will need more pyramid levels and so the speed will suffer.");
    RTABMAP_PARAM(ORB, NLevels,       int, 8,      "The number of pyramid
levels. The smallest level will have linear size equal to
input_image_linear_size/pow(scaleFactor, nlevels).");
    RTABMAP_PARAM(ORB, EdgeThreshold, int, 31,     "This is size of the
border where the features are not detected. It should roughly match the
patchSize parameter.");
    RTABMAP_PARAM(ORB, FirstLevel,    int, 0,      "It should be 0 in the
current implementation.");
    RTABMAP_PARAM(ORB, WTA_K,         int, 2,      "The number of points
that produce each element of the oriented BRIEF descriptor. The default
value 2 means the BRIEF where we take a random point pair and compare their
brightnesses, so we get 0/1 response. Other possible values are 3 and 4. For
example, 3 means that we take 3 random points (of course, those point
coordinates are random, but they are generated from the pre-defined seed, so
each element of BRIEF descriptor is computed deterministically from the
pixel rectangle), find point of maximum brightness and output index of the
winner (0, 1 or 2). Such output will occupy 2 bits, and therefore it will
need a special variant of Hamming distance, denoted as NORM_HAMMING2 (2 bits
per bin). When WTA_K=4, we take 4 random points to compute each bin (that
will also occupy 2 bits with possible values 0, 1, 2 or 3).");
    RTABMAP_PARAM(ORB, ScoreType,     int, 0,      "The default
HARRIS_SCORE=0 means that Harris algorithm is used to rank features (the
score is written to KeyPoint::score and is used to retain best nfeatures
features); FAST_SCORE=1 is alternative value of the parameter that produces
slightly less stable keypoints, but it is a little faster to compute.");
    RTABMAP_PARAM(ORB, PatchSize,     int, 31,     "size of the patch used
by the oriented BRIEF descriptor. Of course, on smaller pyramid layers the
perceived image area covered by a feature will be larger.");
```

```
    RTABMAP_PARAM(ORB, Gpu,           bool, false, "GPU-ORB: Use GPU version
of ORB. This option is enabled only if OpenCV is built with CUDA and GPUs
are detected.");

    RTABMAP_PARAM(FREAK, OrientationNormalized, bool, true,   "Enable
orientation normalization.");
    RTABMAP_PARAM(FREAK, ScaleNormalized,       bool, true,   "Enable scale
normalization.");
    RTABMAP_PARAM(FREAK, PatternScale,          float, 22,    "Scaling of
the description pattern.");
    RTABMAP_PARAM(FREAK, NOctaves,              int, 4,       "Number of
octaves covered by the detected keypoints.");

    RTABMAP_PARAM(BRISK, Thresh,       int, 30, "FAST/AGAST detection
threshold score.");
    RTABMAP_PARAM(BRISK, Octaves,      int, 3,  "Detection octaves. Use 0 to
do single scale.");
    RTABMAP_PARAM(BRISK, PatternScale, float, 1,"Apply this scale to the
pattern used for sampling the neighbourhood of a keypoint.");

    RTABMAP_PARAM(KAZE, Extended,       bool, false,   "Set to enable
extraction of extended (128-byte) descriptor.");
    RTABMAP_PARAM(KAZE, Upright,        bool, false,   "Set to enable use of
upright descriptors (non rotation-invariant).");
    RTABMAP_PARAM(KAZE, Threshold,      float, 0.001,  "Detector response
threshold to accept point.");
    RTABMAP_PARAM(KAZE, NOctaves,       int, 4,        "Maximum octave
evolution of the image.");
    RTABMAP_PARAM(KAZE, NOctaveLayers,  int, 4,        "Default number of
sublevels per scale level.");
    RTABMAP_PARAM(KAZE, Diffusivity,    int, 1,        "Diffusivity type:
0=DIFF_PM_G1, 1=DIFF_PM_G2, 2=DIFF_WEICKERT or 3=DIFF_CHARBONNIER.");

    // BayesFilter
    RTABMAP_PARAM(Bayes, VirtualPlacePriorThr, float, 0.9,  "Virtual place
prior");
    RTABMAP_PARAM_STR(Bayes, PredictionLC, "0.1 0.36 0.30 0.16 0.062 0.0151
0.00255 0.000324 2.5e-05 1.3e-06 4.8e-08 1.2e-09 1.9e-11 2.2e-13 1.7e-15
8.5e-18 2.9e-20 6.9e-23", "Prediction of loop closures (Gaussian-like, here
with sigma=1.6) - Format: {VirtualPlaceProb, LoopClosureProb, NeighborLvl1,
NeighborLvl2, ...}.");
    RTABMAP_PARAM(Bayes, FullPredictionUpdate, bool, false, "Regenerate all
the prediction matrix on each iteration (otherwise only removed/added ids
```

```
are updated).");

    // Verify hypotheses
    RTABMAP_PARAM(VhEp, Enabled, bool, false,       uFormat("Verify visual
loop closure hypothesis by computing a fundamental matrix. This is done
prior to transformation computation when %s is enabled.",
kRGBDEnabled().c_str()));
    RTABMAP_PARAM(VhEp, MatchCountMin, int, 8,      "Minimum of matching
visual words pairs to accept the loop hypothesis.");
    RTABMAP_PARAM(VhEp, RansacParam1,  float, 3,    "Fundamental matrix (see
cvFindFundamentalMat()): Max distance (in pixels) from the epipolar line for
a point to be inlier.");
    RTABMAP_PARAM(VhEp, RansacParam2,  float, 0.99, "Fundamental matrix (see
cvFindFundamentalMat()): Performance of RANSAC.");

    // RGB-D SLAM
    RTABMAP_PARAM(RGBD, Enabled,                    bool, true,  "");
    RTABMAP_PARAM(RGBD, LinearUpdate,               float, 0.1,  "Minimum
linear displacement (m) to update the map. Rehearsal is done prior to this,
so weights are still updated.");
    RTABMAP_PARAM(RGBD, AngularUpdate,              float, 0.1,  "Minimum
angular displacement (rad) to update the map. Rehearsal is done prior to
this, so weights are still updated.");
    RTABMAP_PARAM(RGBD, LinearSpeedUpdate,          float, 0.0,  "Maximum
linear speed (m/s) to update the map (0 means not limit).");
    RTABMAP_PARAM(RGBD, AngularSpeedUpdate,         float, 0.0,  "Maximum
angular speed (rad/s) to update the map (0 means not limit).");
    RTABMAP_PARAM(RGBD, NewMapOdomChangeDistance, float, 0,    "A new map is
created if a change of odometry translation greater than X m is detected (0
m = disabled).");
    RTABMAP_PARAM(RGBD, OptimizeFromGraphEnd,     bool, false, "Optimize
graph from the newest node. If false, the graph is optimized from the oldest
node of the current graph (this adds an overhead computation to detect to
oldest node of the current graph, but it can be useful to preserve the map
referential from the oldest node). Warning when set to false: when some
nodes are transferred, the first referential of the local map may change,
resulting in momentary changes in robot/map position (which are annoying in
teleoperation).");
    RTABMAP_PARAM(RGBD, OptimizeMaxError,         float, 1.0,
uFormat("Reject loop closures if optimization error ratio is greater than
this value (0=disabled). Ratio is computed as absolute error over standard
deviation of each link. This will help to detect when a wrong loop closure
is added to the graph. Not compatible with \"%s\" if enabled.",
```

```
kOptimizerRobust().c_str()));
    RTABMAP_PARAM(RGBD, SavedLocalizationIgnored, bool, false, "Ignore last saved localization pose from previous session. If true, RTAB-Map won't assume it is restarting from the same place than where it shut down previously.");
    RTABMAP_PARAM(RGBD, GoalReachedRadius,        float, 0.5,  "Goal reached radius (m).");
    RTABMAP_PARAM(RGBD, PlanStuckIterations,      int, 0,      "Mark the current goal node on the path as unreachable if it is not updated after X iterations (0=disabled). If all upcoming nodes on the path are unreachabled, the plan fails.");
    RTABMAP_PARAM(RGBD, PlanLinearVelocity,       float, 0,    "Linear velocity (m/sec) used to compute path weights.");
    RTABMAP_PARAM(RGBD, PlanAngularVelocity,      float, 0,    "Angular velocity (rad/sec) used to compute path weights.");
    RTABMAP_PARAM(RGBD, GoalsSavedInUserData,     bool, false, "When a goal is received and processed with success, it is saved in user data of the location with this format: \"GOAL:#\".");
    RTABMAP_PARAM(RGBD, MaxLocalRetrieved,        unsigned int, 2, "Maximum local locations retrieved (0=disabled) near the current pose in the local map or on the current planned path (those on the planned path have priority).");
    RTABMAP_PARAM(RGBD, LocalRadius,              float, 10,   "Local radius (m) for nodes selection in the local map. This parameter is used in some approaches about the local map management.");
    RTABMAP_PARAM(RGBD, LocalImmunizationRatio,   float, 0.25, "Ratio of working memory for which local nodes are immunized from transfer.");
    RTABMAP_PARAM(RGBD, ScanMatchingIdsSavedInLinks, bool, true,    "Save scan matching IDs in link's user data.");
    RTABMAP_PARAM(RGBD, NeighborLinkRefining,         bool, false,
uFormat("When a new node is added to the graph, the transformation of its neighbor link to the previous node is refined using registration approach selected (%s).", kRegStrategy().c_str()));
    RTABMAP_PARAM(RGBD, LoopClosureReextractFeatures, bool, false,  "Extract features even if there are some already in the nodes.");
    RTABMAP_PARAM(RGBD, CreateOccupancyGrid,          bool, false,  "Create local occupancy grid maps. See \"Grid\" group for parameters.");

    // Local/Proximity loop closure detection
    RTABMAP_PARAM(RGBD, ProximityByTime,              bool, false,
"Detection over all locations in STM.");
    RTABMAP_PARAM(RGBD, ProximityBySpace,             bool, true,
"Detection over locations (in Working Memory) near in space.");
```

```
    RTABMAP_PARAM(RGBD, ProximityMaxGraphDepth,       int, 50,     "Maximum depth from the current/last loop closure location and the local loop closure hypotheses. Set 0 to ignore.");
    RTABMAP_PARAM(RGBD, ProximityMaxPaths,            int, 3,      "Maximum paths compared (from the most recent) for proximity detection by space. 0 means no limit.");
    RTABMAP_PARAM(RGBD, ProximityPathFilteringRadius, float, 1,    "Path filtering radius to reduce the number of nodes to compare in a path. A path should also be inside that radius to be considered for proximity detection.");
    RTABMAP_PARAM(RGBD, ProximityPathMaxNeighbors,    int, 0,      "Maximum neighbor nodes compared on each path. Set to 0 to disable merging the laser scans.");
    RTABMAP_PARAM(RGBD, ProximityPathRawPosesUsed,    bool, true,  "When comparing to a local path, merge the scan using the odometry poses (with neighbor link optimizations) instead of the ones in the optimized local graph.");
    RTABMAP_PARAM(RGBD, ProximityAngle,               float, 45,   "Maximum angle (degrees) for visual proximity detection.");

    // Graph optimization
#ifdef RTABMAP_GTSAM
    RTABMAP_PARAM(Optimizer, Strategy,        int, 2,          "Graph optimization strategy: 0=TORO, 1=g2o and 2=GTSAM.");
    RTABMAP_PARAM(Optimizer, Iterations,      int, 20,         "Optimization iterations.");
    RTABMAP_PARAM(Optimizer, Epsilon,         double, 0.00001, "Stop optimizing when the error improvement is less than this value.");
#else
#ifdef RTABMAP_G2O
    RTABMAP_PARAM(Optimizer, Strategy,        int, 1,          "Graph optimization strategy: 0=TORO, 1=g2o and 2=GTSAM.");
    RTABMAP_PARAM(Optimizer, Iterations,      int, 20,         "Optimization iterations.");
    RTABMAP_PARAM(Optimizer, Epsilon,         double, 0.0,     "Stop optimizing when the error improvement is less than this value.");
#else
    RTABMAP_PARAM(Optimizer, Strategy,        int, 0,          "Graph optimization strategy: 0=TORO, 1=g2o and 2=GTSAM.");
    RTABMAP_PARAM(Optimizer, Iterations,      int, 100,        "Optimization iterations.");
    RTABMAP_PARAM(Optimizer, Epsilon,         double, 0.00001, "Stop optimizing when the error improvement is less than this value.");
```

```
#endif
#endif
    RTABMAP_PARAM(Optimizer, VarianceIgnored, bool, false,     "Ignore constraints' variance. If checked, identity information matrix is used for each constraint. Otherwise, an information matrix is generated from the variance saved in the links.");
    RTABMAP_PARAM(Optimizer, Robust,          bool, false,     uFormat("Robust graph optimization using Vertigo (only work for g2o and GTSAM optimization strategies). Not compatible with \"%s\" if enabled.", kRGBDOptimizeMaxError().c_str()));
    RTABMAP_PARAM(Optimizer, PriorsIgnored,   bool, true,      "Ignore prior constraints (global pose or GPS) while optimizing. Currently only g2o and gtsam optimization supports this.");

#ifdef RTABMAP_ORB_SLAM2
    RTABMAP_PARAM(g2o, Solver,            int, 3,          "0=csparse 1=pcg 2=cholmod 3=Eigen");
#else
    RTABMAP_PARAM(g2o, Solver,            int, 0,          "0=csparse 1=pcg 2=cholmod 3=Eigen");
#endif
    RTABMAP_PARAM(g2o, Optimizer,         int, 0,          "0=Levenberg 1=GaussNewton");
    RTABMAP_PARAM(g2o, PixelVariance,     double, 1.0,     "Pixel variance used for bundle adjustment.");
    RTABMAP_PARAM(g2o, RobustKernelDelta, double, 8,       "Robust kernel delta used for bundle adjustment (0 means don't use robust kernel). Observations with chi2 over this threshold will be ignored in the second optimization pass.");
    RTABMAP_PARAM(g2o, Baseline,          double, 0.075,   "When doing bundle adjustment with RGB-D data, we can set a fake baseline (m) to do stereo bundle adjustment (if 0, mono bundle adjustment is done). For stereo data, the baseline in the calibration is used directly.");

    RTABMAP_PARAM(GTSAM, Optimizer,       int, 1,          "0=Levenberg 1=GaussNewton 2=Dogleg");

    // Odometry
    //所采用slam的框架
    RTABMAP_PARAM(Odom, Strategy,               int, 0,       "0=Frame-to-Map (F2M) 1=Frame-to-Frame (F2F) 2=Fovis 3=viso2 4=DVO-SLAM 5=ORB_SLAM2");

    //经过多少次odom计算失败的次数后，重新设置建图起点
```

```
    RTABMAP_PARAM(Odom, ResetCountdown,         int, 0,
"Automatically reset odometry after X consecutive images on which odometry
cannot be computed (value=0 disables auto-reset).");
    RTABMAP_PARAM(Odom, Holonomic,              bool, true,   "If the robot
is holonomic (strafing commands can be issued). If not, y value will be
estimated from x and yaw values (y=x*tan(yaw)).");
    RTABMAP_PARAM(Odom, FillInfoData,           bool, true,   "Fill info
with data (inliers/outliers features).");
    RTABMAP_PARAM(Odom, ImageBufferSize,     unsigned int, 1, "Data buffer
size (0 min inf).");

    //设置滤波器的方式
    RTABMAP_PARAM(Odom, FilteringStrategy,      int, 0,       "0=No
filtering 1=Kalman filtering 2=Particle filtering");

    RTABMAP_PARAM(Odom, ParticleSize,      unsigned int, 400, "Number of
particles of the filter.");
    RTABMAP_PARAM(Odom, ParticleNoiseT,         float, 0.002, "Noise (m) of
translation components (x,y,z).");
    RTABMAP_PARAM(Odom, ParticleLambdaT,        float, 100,   "Lambda of
translation components (x,y,z).");
    RTABMAP_PARAM(Odom, ParticleNoiseR,         float, 0.002, "Noise (rad)
of rotational components (roll,pitch,yaw).");
    RTABMAP_PARAM(Odom, ParticleLambdaR,        float, 100,   "Lambda of
rotational components (roll,pitch,yaw).");
    RTABMAP_PARAM(Odom, KalmanProcessNoise,     float, 0.001, "Process noise
covariance value.");
    RTABMAP_PARAM(Odom, KalmanMeasurementNoise, float, 0.01,  "Process
measurement covariance value.");
    RTABMAP_PARAM(Odom, GuessMotion,            bool, true,   "Guess next
transformation from the last motion computed.");
    RTABMAP_PARAM(Odom, KeyFrameThr,            float, 0.3,   "[Visual]
Create a new keyframe when the number of inliers drops under this ratio of
features in last frame. Setting the value to 0 means that a keyframe is
created for each processed frame.");
    RTABMAP_PARAM(Odom, VisKeyFrameThr,         int, 150,     "[Visual]
Create a new keyframe when the number of inliers drops under this threshold.
Setting the value to 0 means that a keyframe is created for each processed
frame.");
    RTABMAP_PARAM(Odom, ScanKeyFrameThr,        float, 0.9,   "[Geometry]
Create a new keyframe when the number of ICP inliers drops under this ratio
of points in last frame's scan. Setting the value to 0 means that a keyframe
is created for each processed frame.");
```

```
    RTABMAP_PARAM(Odom, ImageDecimation,        int, 1,       "Decimation of the images before registration. Negative decimation is done from RGB size instead of depth size (if depth is smaller than RGB, it may be interpolated depending of the decimation value).");
    RTABMAP_PARAM(Odom, AlignWithGround,        bool, false,  "Align odometry with the ground on initialization.");

    // Odometry Frame-to-Map
    RTABMAP_PARAM(OdomF2M, MaxSize,             int, 2000,    "[Visual] Local map size: If > 0 (example 5000), the odometry will maintain a local map of X maximum words.");
    RTABMAP_PARAM(OdomF2M, MaxNewFeatures,      int, 0,       "[Visual] Maximum features (sorted by keypoint response) added to local map from a new key-frame. 0 means no limit.");
    RTABMAP_PARAM(OdomF2M, ScanMaxSize,         int, 2000,    "[Geometry] Maximum local scan map size.");
    RTABMAP_PARAM(OdomF2M, ScanSubtractRadius,  float, 0.05,  "[Geometry] Radius used to filter points of a new added scan to local map. This could match the voxel size of the scans.");
    RTABMAP_PARAM(OdomF2M, ScanSubtractAngle,   float, 45,    uFormat("[Geometry] Max angle (degrees) used to filter points of a new added scan to local map (when \"%s\">0). 0 means any angle.", kOdomF2MScanSubtractRadius().c_str()).c_str());

     //设置是否ba优化，以及ba优化的方式
#if defined(RTABMAP_G2O) || defined(RTABMAP_ORB_SLAM2)
    RTABMAP_PARAM(OdomF2M, BundleAdjustment,          int, 1, "Local bundle adjustment: 0=disabled, 1=g2o, 2=cvsba.");
#else
    RTABMAP_PARAM(OdomF2M, BundleAdjustment,          int, 0, "Local bundle adjustment: 0=disabled, 1=g2o, 2=cvsba.");
#endif
    RTABMAP_PARAM(OdomF2M, BundleAdjustmentMaxFrames, int, 10, "Maximum frames used for bundle adjustment (0=inf or all current frames in the local map).");

    // Odometry Mono
    RTABMAP_PARAM(OdomMono, InitMinFlow,        float, 100,  "Minimum optical flow required for the initialization step.");
    RTABMAP_PARAM(OdomMono, InitMinTranslation, float, 0.1,  "Minimum translation required for the initialization step.");
    RTABMAP_PARAM(OdomMono, MinTranslation,     float, 0.02, "Minimum translation to add new points to local map. On initialization, translation x
```

```
5 is used as the minimum.");
    RTABMAP_PARAM(OdomMono, MaxVariance,        float, 0.01, "Maximum
variance to add new points to local map.");

    // Odometry Fovis
    RTABMAP_PARAM(OdomFovis, FeatureWindowSize,               int, 9,
"The size of the n x n image patch surrounding each feature, used for
keypoint matching.");
    RTABMAP_PARAM(OdomFovis, MaxPyramidLevel,                 int, 3,
"The maximum Gaussian pyramid level to process the image at. Pyramid level 1
corresponds to the original image.");
    RTABMAP_PARAM(OdomFovis, MinPyramidLevel,                 int, 0,
"The minimum pyramid level.");
    RTABMAP_PARAM(OdomFovis, TargetPixelsPerFeature,          int, 250,
"Specifies the desired feature density as a ratio of input image pixels per
feature detected.  This number is used to control the adaptive feature
thresholding.");
    RTABMAP_PARAM(OdomFovis, FastThreshold,                   int, 20,
"FAST threshold.");
    RTABMAP_PARAM(OdomFovis, UseAdaptiveThreshold,            bool, true,
"Use FAST adaptive threshold.");
    RTABMAP_PARAM(OdomFovis, FastThresholdAdaptiveGain,       double, 0.005,
"FAST threshold adaptive gain.");
    RTABMAP_PARAM(OdomFovis, UseHomographyInitialization,     bool, true,
"Use homography initialization.");

    RTABMAP_PARAM(OdomFovis, UseBucketing,                    bool, true,
"");
    RTABMAP_PARAM(OdomFovis, BucketWidth,                     int, 80,
"");
    RTABMAP_PARAM(OdomFovis, BucketHeight,                    int, 80,
"");
    RTABMAP_PARAM(OdomFovis, MaxKeypointsPerBucket,           int, 25,
"");
    RTABMAP_PARAM(OdomFovis, UseImageNormalization,           bool, false,
"");

    RTABMAP_PARAM(OdomFovis, InlierMaxReprojectionError,      double, 1.5,
"The maximum image-space reprojection error (in pixels) a feature match is
allowed to have and still be considered an inlier in the set of features
used for motion estimation.");
    RTABMAP_PARAM(OdomFovis, CliqueInlierThreshold,           double, 0.1,
"See Howard's greedy max-clique algorithm for determining the maximum set of
```

```
mutually consisten feature matches. This specifies the compatibility
threshold, in meters.");
    RTABMAP_PARAM(OdomFovis, MinFeaturesForEstimate,          int, 20,
"Minimum number of features in the inlier set for the motion estimate to be
considered valid.");
    RTABMAP_PARAM(OdomFovis, MaxMeanReprojectionError,        double, 10.0,
"Maximum mean reprojection error over the inlier feature matches for the
motion estimate to be considered valid.");
    RTABMAP_PARAM(OdomFovis, UseSubpixelRefinement,           bool, true,
"Specifies whether or not to refine feature matches to subpixel
resolution.");
    RTABMAP_PARAM(OdomFovis, FeatureSearchWindow,             int, 25,
"Specifies the size of the search window to apply when searching for feature
matches across time frames.  The search is conducted around the feature
location predicted by the initial rotation estimate.");
    RTABMAP_PARAM(OdomFovis, UpdateTargetFeaturesWithRefined, bool, false,
"When subpixel refinement is enabled, the refined feature locations can be
saved over the original feature locations.  This has a slightly negative
impact on frame-to-frame visual odometry, but is likely better when using
this library as part of a visual SLAM algorithm.");

    RTABMAP_PARAM(OdomFovis, StereoRequireMutualMatch,        bool, true,
"");
    RTABMAP_PARAM(OdomFovis, StereoMaxDistEpipolarLine,       double, 1.5,
"");
    RTABMAP_PARAM(OdomFovis, StereoMaxRefinementDisplacement, double, 1.0,
"");
    RTABMAP_PARAM(OdomFovis, StereoMaxDisparity,              int, 128,
"");

    // Odometry viso2
    RTABMAP_PARAM(OdomViso2, RansacIters,               int, 200,    "Number
of RANSAC iterations.");
    RTABMAP_PARAM(OdomViso2, InlierThreshold,           double, 2.0,
"Fundamental matrix inlier threshold.");
    RTABMAP_PARAM(OdomViso2, Reweighting,               bool, true,  "Lower
border weights (more robust to calibration errors).");
    RTABMAP_PARAM(OdomViso2, MatchNmsN,                 int, 3,      "Non-
max-suppression: min. distance between maxima (in pixels).");
    RTABMAP_PARAM(OdomViso2, MatchNmsTau,               int, 50,     "Non-
max-suppression: interest point peakiness threshold.");
    RTABMAP_PARAM(OdomViso2, MatchBinsize,              int, 50,
"Matching bin width/height (affects efficiency only).");
```

```
    RTABMAP_PARAM(OdomViso2, MatchRadius,               int, 200,
"Matching radius (du/dv in pixels).");
    RTABMAP_PARAM(OdomViso2, MatchDispTolerance,        int, 2,
"Disparity tolerance for stereo matches (in pixels).");
    RTABMAP_PARAM(OdomViso2, MatchOutlierDispTolerance, int, 5,
"Outlier removal: disparity tolerance (in pixels).");
    RTABMAP_PARAM(OdomViso2, MatchOutlierFlowTolerance, int, 5,
"Outlier removal: flow tolerance (in pixels).");
    RTABMAP_PARAM(OdomViso2, MatchMultiStage,           bool, true,
"Multistage matching (denser and faster).");
    RTABMAP_PARAM(OdomViso2, MatchHalfResolution,       bool, true,  "Match
at half resolution, refine at full resolution.");
    RTABMAP_PARAM(OdomViso2, MatchRefinement,           int, 1,
"Refinement (0=none,1=pixel,2=subpixel).");
    RTABMAP_PARAM(OdomViso2, BucketMaxFeatures,         int, 2,
"Maximal number of features per bucket.");
    RTABMAP_PARAM(OdomViso2, BucketWidth,               double, 50,  "Width
of bucket.");
    RTABMAP_PARAM(OdomViso2, BucketHeight,              double, 50,  "Height
of bucket.");

    // Odometry ORB_SLAM2
    RTABMAP_PARAM_STR(OdomORBSLAM2, VocPath,             "",    "Path to ORB
vocabulary (*.txt).");
    RTABMAP_PARAM(OdomORBSLAM2, Bf,              double, 0.076, "Fake IR
projector baseline (m) used only when stereo is not used.");
    RTABMAP_PARAM(OdomORBSLAM2, ThDepth,         double, 40.0,  "Close/Far
threshold. Baseline times.");
    RTABMAP_PARAM(OdomORBSLAM2, Fps,             float,  0.0,   "Camera
FPS.");
    RTABMAP_PARAM(OdomORBSLAM2, MaxFeatures,     int,    1000,  "Maximum ORB
features extracted per frame.");
    RTABMAP_PARAM(OdomORBSLAM2, MapSize,         int,    3000,  "Maximum
size of the feature map (0 means infinite).");

    // Odometry OKVIS
    RTABMAP_PARAM_STR(OdomOKVIS, ConfigPath,     "",  "Path of OKVIS config
file.");

    // Common registration parameters
    RTABMAP_PARAM(Reg, RepeatOnce,               bool, true,    "Do a second
registration with the output of the first registration as guess. Only done
if no guess was provided for the first registration (like on loop closure).
```

```
It can be useful if the registration approach used can use a guess to get
better matches.");
    RTABMAP_PARAM(Reg, Strategy,                int, 0,        "0=Vis,
1=Icp, 2=VisIcp");
    RTABMAP_PARAM(Reg, Force3DoF,               bool, false,   "Force 3
degrees-of-freedom transform (3Dof: x,y and yaw). Parameters z, roll and
pitch will be set to 0.");

    // Visual registration parameters
    RTABMAP_PARAM(Vis, EstimationType,          int, 1,        "Motion
estimation approach: 0:3D->3D, 1:3D->2D (PnP), 2:2D->2D (Epipolar
Geometry)");
    RTABMAP_PARAM(Vis, ForwardEstOnly,          bool, true,    "Forward
estimation only (A->B). If false, a transformation is also computed in
backward direction (B->A), then the two resulting transforms are merged
(middle interpolation between the transforms).");
    RTABMAP_PARAM(Vis, InlierDistance,          float, 0.1,    uFormat("[%s
= 0] Maximum distance for feature correspondences. Used by 3D->3D estimation
approach.", kVisEstimationType().c_str()));
    RTABMAP_PARAM(Vis, RefineIterations,        int, 5,        uFormat("[%s
= 0] Number of iterations used to refine the transformation found by RANSAC.
0 means that the transformation is not refined.",
kVisEstimationType().c_str()));
    RTABMAP_PARAM(Vis, PnPReprojError,          float, 2,      uFormat("[%s
= 1] PnP reprojection error.", kVisEstimationType().c_str()));
    RTABMAP_PARAM(Vis, PnPFlags,                int, 0,        uFormat("[%s
= 1] PnP flags: 0=Iterative, 1=EPNP, 2=P3P", kVisEstimationType().c_str()));
#if defined(RTABMAP_G2O) || defined(RTABMAP_ORB_SLAM2)
    RTABMAP_PARAM(Vis, PnPRefineIterations,     int, 0,        uFormat("[%s
= 1] Refine iterations. Set to 0 if \"%s\" is also used.",
kVisEstimationType().c_str(), kVisBundleAdjustment().c_str()));
#else
    RTABMAP_PARAM(Vis, PnPRefineIterations,     int, 1,        uFormat("[%s
= 1] Refine iterations. Set to 0 if \"%s\" is also used.",
kVisEstimationType().c_str(), kVisBundleAdjustment().c_str()));
#endif

    RTABMAP_PARAM(Vis, EpipolarGeometryVar,     float, 0.02,   uFormat("[%s
= 2] Epipolar geometry maximum variance to accept the transformation.",
kVisEstimationType().c_str()));
    RTABMAP_PARAM(Vis, MinInliers,              int, 20,       "Minimum
feature correspondences to compute/accept the transformation.");
    RTABMAP_PARAM(Vis, Iterations,              int, 300,      "Maximum
```

```
iterations to compute the transform.");
#ifndef RTABMAP_NONFREE
#ifdef RTABMAP_OPENCV3
    // OpenCV 3 without xFeatures2D module doesn't have BRIEF
    RTABMAP_PARAM(Vis, FeatureType, int, 8, "0=SURF 1=SIFT 2=ORB
3=FAST/FREAK 4=FAST/BRIEF 5=GFTT/FREAK 6=GFTT/BRIEF 7=BRISK 8=GFTT/ORB
9=KAZE.");
#else
    RTABMAP_PARAM(Vis, FeatureType, int, 6, "0=SURF 1=SIFT 2=ORB
3=FAST/FREAK 4=FAST/BRIEF 5=GFTT/FREAK 6=GFTT/BRIEF 7=BRISK 8=GFTT/ORB
9=KAZE.");
#endif
#else
    RTABMAP_PARAM(Vis, FeatureType, int, 6, "0=SURF 1=SIFT 2=ORB
3=FAST/FREAK 4=FAST/BRIEF 5=GFTT/FREAK 6=GFTT/BRIEF 7=BRISK 8=GFTT/ORB
9=KAZE.");
#endif
    RTABMAP_PARAM(Vis, MaxFeatures,              int, 1000,   "0 no
limits.");
    RTABMAP_PARAM(Vis, MaxDepth,                  float, 0,   "Max depth of
the features (0 means no limit).");
    RTABMAP_PARAM(Vis, MinDepth,                  float, 0,   "Min depth of
the features (0 means no limit).");
    RTABMAP_PARAM(Vis, DepthAsMask,               bool, true, "Use depth
image as mask when extracting features.");
    RTABMAP_PARAM_STR(Vis, RoiRatios,      "0.0 0.0 0.0 0.0", "Region of
interest ratios [left, right, top, bottom].");
    RTABMAP_PARAM(Vis, SubPixWinSize,            int, 3,      "See
cv::cornerSubPix().");
    RTABMAP_PARAM(Vis, SubPixIterations,         int, 0,      "See
cv::cornerSubPix(). 0 disables sub pixel refining.");
    RTABMAP_PARAM(Vis, SubPixEps,                float, 0.02, "See
cv::cornerSubPix().");
    RTABMAP_PARAM(Vis, GridRows,                 int, 1,
uFormat("Number of rows of the grid used to extract uniformly \"%s / grid
cells\" features from each cell.", kVisMaxFeatures().c_str()));
    RTABMAP_PARAM(Vis, GridCols,                 int, 1,
uFormat("Number of columns of the grid used to extract uniformly \"%s / grid
cells\" features from each cell.", kVisMaxFeatures().c_str()));
    RTABMAP_PARAM(Vis, CorType,                  int, 0,
"Correspondences computation approach: 0=Features Matching, 1=Optical
Flow");
    RTABMAP_PARAM(Vis, CorNNType,                int, 1,    uFormat("[%s=0]
```

```
kNNFlannNaive=0, kNNFlannKdTree=1, kNNFlannLSH=2, kNNBruteForce=3,
kNNBruteForceGPU=4. Used for features matching approach.",
kVisCorType().c_str()));
    RTABMAP_PARAM(Vis, CorNNDR,                  float, 0.6,  uFormat("
[%s=0] NNDR: nearest neighbor distance ratio. Used for features matching
approach.", kVisCorType().c_str()));
    RTABMAP_PARAM(Vis, CorGuessWinSize,          int, 20,     uFormat("
[%s=0] Matching window size (pixels) around projected points when a guess
transform is provided to find correspondences. 0 means disabled.",
kVisCorType().c_str()));
    RTABMAP_PARAM(Vis, CorGuessMatchToProjection, bool, false, uFormat("
[%s=0] Match frame's corners to source's projected points (when guess
transform is provided) instead of projected points to frame's corners.",
kVisCorType().c_str()));
    RTABMAP_PARAM(Vis, CorFlowWinSize,           int, 16,     uFormat("
[%s=1] See cv::calcOpticalFlowPyrLK(). Used for optical flow approach.",
kVisCorType().c_str()));
    RTABMAP_PARAM(Vis, CorFlowIterations,        int, 30,     uFormat("
[%s=1] See cv::calcOpticalFlowPyrLK(). Used for optical flow approach.",
kVisCorType().c_str()));
    RTABMAP_PARAM(Vis, CorFlowEps,               float, 0.01, uFormat("
[%s=1] See cv::calcOpticalFlowPyrLK(). Used for optical flow approach.",
kVisCorType().c_str()));
    RTABMAP_PARAM(Vis, CorFlowMaxLevel,          int, 3,      uFormat("
[%s=1] See cv::calcOpticalFlowPyrLK(). Used for optical flow approach.",
kVisCorType().c_str()));
#if defined(RTABMAP_G2O) || defined(RTABMAP_ORB_SLAM2)
    RTABMAP_PARAM(Vis, BundleAdjustment,         int, 1,      "Optimization
with bundle adjustment: 0=disabled, 1=g2o, 2=cvsba.");
#else
    RTABMAP_PARAM(Vis, BundleAdjustment,         int, 0,      "Optimization
with bundle adjustment: 0=disabled, 1=g2o, 2=cvsba.");
#endif

    // ICP registration parameters
    RTABMAP_PARAM(Icp, MaxTranslation,           float, 0.2,   "Maximum ICP
translation correction accepted (m).");
    RTABMAP_PARAM(Icp, MaxRotation,              float, 0.78,  "Maximum ICP
rotation correction accepted (rad).");
    RTABMAP_PARAM(Icp, VoxelSize,                float, 0.0,   "Uniform
sampling voxel size (0=disabled).");
    RTABMAP_PARAM(Icp, DownsamplingStep,         int, 1,
"Downsampling step size (1=no sampling). This is done before uniform
```

```
sampling.");
#ifdef RTABMAP_POINTMATCHER
    RTABMAP_PARAM(Icp, MaxCorrespondenceDistance, float, 0.1,   "Max
distance for point correspondences.");
#else
    RTABMAP_PARAM(Icp, MaxCorrespondenceDistance, float, 0.05,  "Max
distance for point correspondences.");
#endif
    RTABMAP_PARAM(Icp, Iterations,                int, 30,      "Max
iterations.");
    RTABMAP_PARAM(Icp, Epsilon,                   float, 0,     "Set the
transformation epsilon (maximum allowable difference between two consecutive
transformations) in order for an optimization to be considered as having
converged to the final solution.");
    RTABMAP_PARAM(Icp, CorrespondenceRatio,       float, 0.1,   "Ratio of
matching correspondences to accept the transform.");
#ifdef RTABMAP_POINTMATCHER
    RTABMAP_PARAM(Icp, PointToPlane,              bool, true,   "Use point
to plane ICP.");
#else
    RTABMAP_PARAM(Icp, PointToPlane,              bool, false,  "Use point
to plane ICP.");
#endif
    RTABMAP_PARAM(Icp, PointToPlaneK,             int, 5,       "Number of
neighbors to compute normals for point to plane if the cloud doesn't have
already normals.");
    RTABMAP_PARAM(Icp, PointToPlaneRadius,        float, 1.0,   "Search
radius to compute normals for point to plane if the cloud doesn't have
already normals.");
    RTABMAP_PARAM(Icp, PointToPlaneMinComplexity, float, 0.02,  "Minimum
structural complexity (0.0=low, 1.0=high) of the scan to do point to plane
registration, otherwise point to point registration is done instead.");

    // libpointmatcher
#ifdef RTABMAP_POINTMATCHER
    RTABMAP_PARAM(Icp, PM,                        bool, true,    "Use
libpointmatcher for ICP registration instead of PCL's implementation.");
#else
    RTABMAP_PARAM(Icp, PM,                        bool, false,   "Use
libpointmatcher for ICP registration instead of PCL's implementation.");
#endif
    RTABMAP_PARAM_STR(Icp, PMConfig,              "",
uFormat("Configuration file (*.yaml) used by libpointmatcher. Note that data
```

```
filters set for libpointmatcher are done after filtering done by rtabmap
(i.e., %s, %s), so make sure to disable those in rtabmap if you want to use
only those from libpointmatcher. Parameters %s, %s and %s are also ignored
if configuration file is set.", kIcpVoxelSize().c_str(),
kIcpDownsamplingStep().c_str(), kIcpIterations().c_str(),
kIcpEpsilon().c_str(), kIcpMaxCorrespondenceDistance().c_str()).c_str());
    RTABMAP_PARAM(Icp, PMMatcherKnn,             int, 1,
"KDTreeMatcher/knn: number of nearest neighbors to consider it the
reference. For convenience when configuration file is not set.");
    RTABMAP_PARAM(Icp, PMMatcherEpsilon,         float, 0.0,
"KDTreeMatcher/epsilon: approximation to use for the nearest-neighbor
search. For convenience when configuration file is not set.");
    RTABMAP_PARAM(Icp, PMOutlierRatio,           float, 0.95,
"TrimmedDistOutlierFilter/ratio: For convenience when configuration file is
not set. For kinect-like point cloud, use 0.65.");

    // Stereo disparity
    RTABMAP_PARAM(Stereo, WinWidth,              int, 15,       "Window
width.");
    RTABMAP_PARAM(Stereo, WinHeight,             int, 3,        "Window
height.");
    RTABMAP_PARAM(Stereo, Iterations,            int, 30,       "Maximum
iterations.");
    RTABMAP_PARAM(Stereo, MaxLevel,              int, 5,        "Maximum
pyramid level.");
    RTABMAP_PARAM(Stereo, MinDisparity,          float, 0.5,    "Minimum
disparity.");
    RTABMAP_PARAM(Stereo, MaxDisparity,          float, 128.0,  "Maximum
disparity.");
    RTABMAP_PARAM(Stereo, OpticalFlow,           bool, true,    "Use optical
flow to find stereo correspondences, otherwise a simple block matching
approach is used.");
    RTABMAP_PARAM(Stereo, SSD,                   bool, true,    uFormat("
[%s=false] Use Sum of Squared Differences (SSD) window, otherwise Sum of
Absolute Differences (SAD) window is used.", kStereoOpticalFlow().c_str()));
    RTABMAP_PARAM(Stereo, Eps,                   double, 0.01,  uFormat("
[%s=true] Epsilon stop criterion.", kStereoOpticalFlow().c_str()));

    RTABMAP_PARAM(StereoBM, BlockSize,           int, 15,       "See
cv::StereoBM");
    RTABMAP_PARAM(StereoBM, MinDisparity,        int, 0,        "See
cv::StereoBM");
    RTABMAP_PARAM(StereoBM, NumDisparities,      int, 128,      "See
```

```
cv::StereoBM");
    RTABMAP_PARAM(StereoBM, PreFilterSize,       int, 9,        "See
cv::StereoBM");
    RTABMAP_PARAM(StereoBM, PreFilterCap,        int, 31,       "See
cv::StereoBM");
    RTABMAP_PARAM(StereoBM, UniquenessRatio,     int, 15,       "See
cv::StereoBM");
    RTABMAP_PARAM(StereoBM, TextureThreshold,    int, 10,       "See
cv::StereoBM");
    RTABMAP_PARAM(StereoBM, SpeckleWindowSize,   int, 100,      "See
cv::StereoBM");
    RTABMAP_PARAM(StereoBM, SpeckleRange,        int, 4,        "See
cv::StereoBM");

    // Occupancy Grid
    RTABMAP_PARAM(Grid, FromDepth,               bool,   true,    "Create
occupancy grid from depth image(s), otherwise it is created from laser
scan.");
    RTABMAP_PARAM(Grid, DepthDecimation,         int,    4,       uFormat("
[%s=true] Decimation of the depth image before creating cloud. Negative
decimation is done from RGB size instead of depth size (if depth is smaller
than RGB, it may be interpolated depending of the decimation value).",
kGridDepthDecimation().c_str()));
    RTABMAP_PARAM(Grid, RangeMin,                float,  0.0,     "Minimum
range from sensor.");
    RTABMAP_PARAM(Grid, RangeMax,                float,  5.0,     "Maximum
range from sensor. 0=inf.");
    RTABMAP_PARAM_STR(Grid, DepthRoiRatios,      "0.0 0.0 0.0 0.0",
uFormat("[%s=true] Region of interest ratios [left, right, top, bottom].",
kGridFromDepth().c_str()));
    RTABMAP_PARAM(Grid, FootprintLength,         float,  0.0,     "Footprint
length used to filter points over the footprint of the robot.");
    RTABMAP_PARAM(Grid, FootprintWidth,          float,  0.0,     "Footprint
width used to filter points over the footprint of the robot. Footprint
length should be set.");
    RTABMAP_PARAM(Grid, FootprintHeight,         float,  0.0,     "Footprint
height used to filter points over the footprint of the robot. Footprint
length and width should be set.");
    RTABMAP_PARAM(Grid, ScanDecimation,          int,    1,       uFormat("
[%s=false] Decimation of the laser scan before creating cloud.",
kGridFromDepth().c_str()));
    RTABMAP_PARAM(Grid, CellSize,                float,  0.05,
"Resolution of the occupancy grid.");
```

```
    RTABMAP_PARAM(Grid, PreVoxelFiltering,       bool,   true,
uFormat("Input cloud is downsampled by voxel filter (voxel size is \"%s\")
before doing segmentation of obstacles and ground.",
kGridCellSize().c_str()));
    RTABMAP_PARAM(Grid, MapFrameProjection,      bool,   false,
"Projection in map frame. On a 3D terrain and a fixed local camera transform
(the cloud is created relative to ground), you may want to disable this to
do the projection in robot frame instead.");
    RTABMAP_PARAM(Grid, NormalsSegmentation,     bool,   true,    "Segment
ground from obstacles using point normals, otherwise a fast passthrough is
used.");
    RTABMAP_PARAM(Grid, MaxObstacleHeight,       float,  0.0,     "Maximum
obstacles height (0=disabled).");
    RTABMAP_PARAM(Grid, MinGroundHeight,         float,  0.0,     "Minimum
ground height (0=disabled).");
    RTABMAP_PARAM(Grid, MaxGroundHeight,         float,  0.0,
uFormat("Maximum ground height (0=disabled). Should be set if \"%s\" is
false.", kGridNormalsSegmentation().c_str()));
    RTABMAP_PARAM(Grid, MaxGroundAngle,          float,  45,      uFormat("
[%s=true] Maximum angle (degrees) between point's normal to ground's normal
to label it as ground. Points with higher angle difference are considered as
obstacles.", kGridNormalsSegmentation().c_str()));
    RTABMAP_PARAM(Grid, NormalK,                 int,    20,      uFormat("
[%s=true] K neighbors to compute normals.",
kGridNormalsSegmentation().c_str()));
    RTABMAP_PARAM(Grid, ClusterRadius,           float,  0.1,     uFormat("
[%s=true] Cluster maximum radius.", kGridNormalsSegmentation().c_str()));
    RTABMAP_PARAM(Grid, MinClusterSize,          int,    10,      uFormat("
[%s=true] Minimum cluster size to project the points.",
kGridNormalsSegmentation().c_str()));
    RTABMAP_PARAM(Grid, FlatObstacleDetected,    bool,   true,    uFormat("
[%s=true] Flat obstacles detected.", kGridNormalsSegmentation().c_str()));
#ifdef RTABMAP_OCTOMAP
    RTABMAP_PARAM(Grid, 3D,                      bool,   true,    uFormat("A
3D occupancy grid is required if you want an OctoMap (3D ray tracing). Set
to false if you want only a 2D map, the cloud will be projected on xy plane.
A 2D map can be still generated if checked, but it requires more memory and
time to generate it. Ignored if laser scan is 2D and \"%s\" is false.",
kGridFromDepth().c_str()));
#else
    RTABMAP_PARAM(Grid, 3D,                      bool,   false,   uFormat("A
3D occupancy grid is required if you want an OctoMap (3D ray tracing). Set
to false if you want only a 2D map, the cloud will be projected on xy plane.
```

```
A 2D map can be still generated if checked, but it requires more memory and
time to generate it. Ignored if laser scan is 2D and \"%s\" is false.",
kGridFromDepth().c_str()));
#endif
    RTABMAP_PARAM(Grid, GroundIsObstacle,           bool,   false,
uFormat("[%s=true] Ground segmentation (%s) is ignored, all points are
obstacles. Use this only if you want an OctoMap with ground identified as an
obstacle (e.g., with an UAV).", kGrid3D().c_str(),
kGridNormalsSegmentation().c_str()));
    RTABMAP_PARAM(Grid, NoiseFilteringRadius,       float,  0.0,    "Noise
filtering radius (0=disabled). Done after segmentation.");
    RTABMAP_PARAM(Grid, NoiseFilteringMinNeighbors, int,    5,      "Noise
filtering minimum neighbors.");
    RTABMAP_PARAM(Grid, Scan2dUnknownSpaceFilled,   bool,   false,
uFormat("Unknown space filled. Only used with 2D laser scans. Use %s to set
maximum range if laser scan max range is to set.",
kGridRangeMax().c_str()));
    RTABMAP_PARAM(Grid, RayTracing,                 bool,   false,
uFormat("Ray tracing is done for each occupied cell, filling unknown space
between the sensor and occupied cells. If %s=true, RTAB-Map should be built
with OctoMap support, otherwise 3D ray tracing is ignored.",
kGrid3D().c_str()));

    RTABMAP_PARAM(GridGlobal, FullUpdate,           bool,   true,   "When
the graph is changed, the whole map will be reconstructed instead of moving
individually each cells of the map. Also, data added to cache won't be
released after updating the map. This process is longer but more robust to
drift that would erase some parts of the map when it should not.");
    RTABMAP_PARAM(GridGlobal, UpdateError,          float,  0.01,   "Graph
changed detection error (m). Update map only if poses in new optimized graph
have moved more than this value.");
    RTABMAP_PARAM(GridGlobal, FootprintRadius,      float,  0.0,
"Footprint radius (m) used to clear all obstacles under the graph.");
    RTABMAP_PARAM(GridGlobal, MinSize,              float,  0.0,
"Minimum map size (m).");
    RTABMAP_PARAM(GridGlobal, Eroded,               bool,   false,  "Erode
obstacle cells.");
    RTABMAP_PARAM(GridGlobal, MaxNodes,             int,    0,
"Maximum nodes assembled in the map starting from the last node
(0=unlimited).");
    RTABMAP_PARAM(GridGlobal, OctoMapOccupancyThr,  float,  0.5,
"OctoMap occupancy threshold (value between 0 and 1).");
```

在rtabmap_ros中，设置rtabmap的参数有两种方式：

直接在launch文件中设置, 如设置计算里程计的使用的库：

```
    <node pkg="rtabmap_ros"  type="rgbd_odometry"
name="rgbd_odometry" output="screen">
        <param name="Odom/Strategy"  type="string" value="0"/> <!--
0=Frame-to-Map, 1=Frame=to=Frame -->
    </node>

    <node pkg="rtabmap_ros"  type="rgbd_odometry"
name="rgbd_odometry" output="screen" args=" --Odom/Strategy 1">
    </node>
```

在配置文件中配置，首先在node的参数中设置好配置文件路径，然后再在配置文件设置，如：

```
        <param name="config_path"  type="string" value="～/rtabmap.ini"/>
        ～/rtabmap.ini：
        [Odom]
        Strategy=1
```