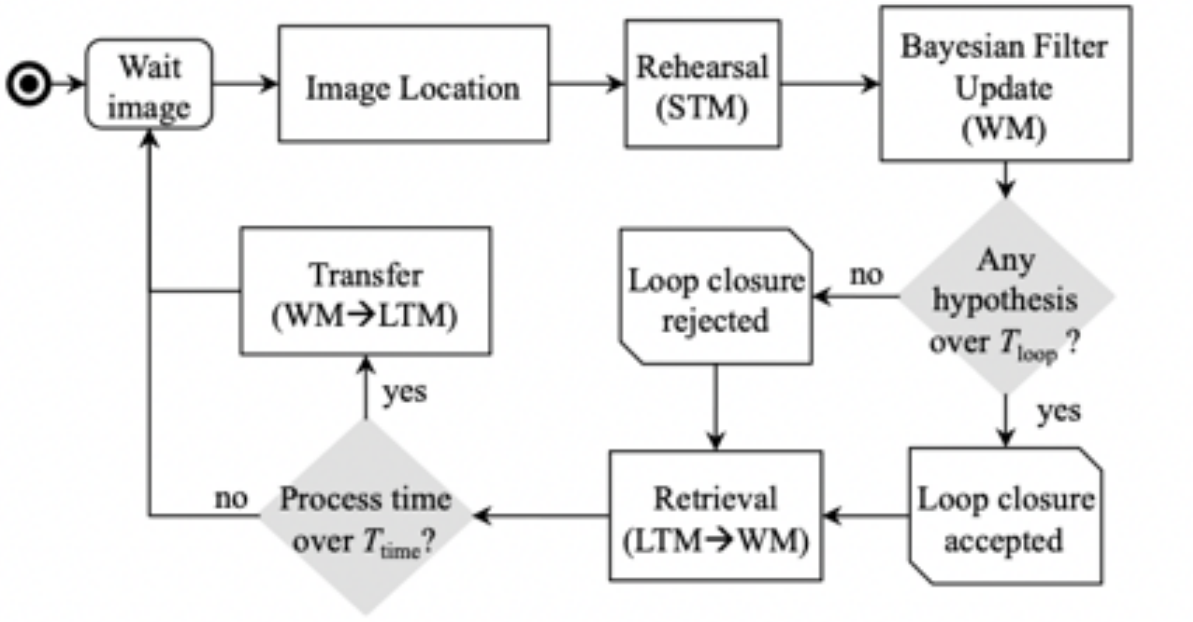
Dummies Guide to RTAB-MAP

Contents

Chapter 1
Introduction

RTAB-MAP or Real-Time Appearance-Based Mapping is a software used to detect loop closures irrespective of time and map size. It uses memory management techniques to achieve this, this encompasses STM (Short term memory), WM (Working memory) and LTM (Long term memory). The main idea is that only a certain number of locations are used for loop closure detection, while transferring those locations which are less likely to cause loop closures.



Chapter 2
How it works?

Initially, when a frame from a camera is sent to RTAB-MAP through the camera at t_0, the location/frame is converted into a bag of words and a weight one is assigned to the frame. This bag of words contains the key points and their descriptors. Subsequently, when the next location/frame at t_1 is sent to RTAB-MAP a similarity function is used to evaluate whether this location is significant enough to be considered a new location or not. If the location is similar to the past frame, it merged with that frame and the weight of the frame is incremented by one. This essentially converts all incoming frames to keyframes, while also ensuring locations very close to each other do not cause loop closures. These frames are stored in the STM, therefore frames in the STM are not used for loop closures. The size of STM can be set manually, in case we have a robot that moves very slowly the STM can be increased to ensure RTAB-MAP performs as intended. When the STM is full the oldest keyframe is sent to the WM, this is where the Loop closure hypothesis occurs. Whenever a location is deemed as a location already visited using the similarity function, the loop closure hypothesis is accepted, and the weight of that location is updated. Whenever the WM gets full those locations with the least weight is transferred to the LTM, here there is no loop closure done. If a location is accepted as a loop closure those locations which are close to it are transferred back from the LTM to WM. This process continues until the mapping session is terminated.

Chapter 3
Mapping

Mapping on RTAB-MAP depends on the best parameters needed to achieve loop closure. Each significant parameter is given below-

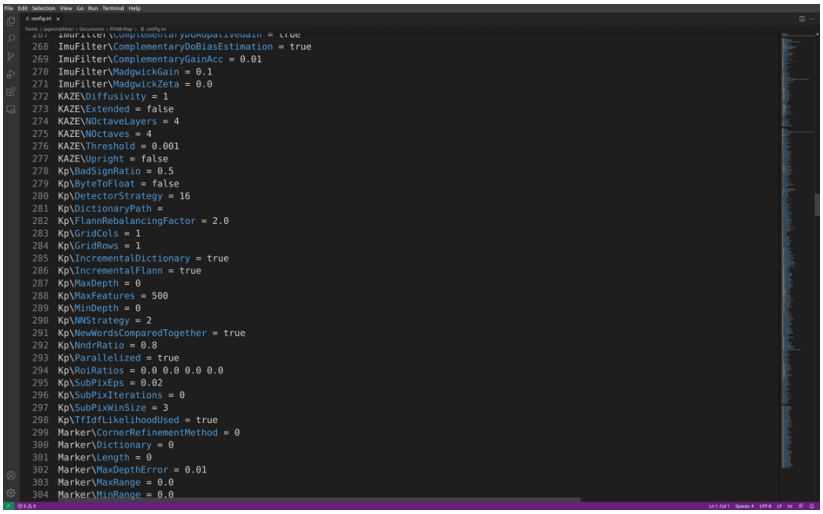
1) General Settings (GUI)
    a) Save/Load Settings
2) 3D Rendering Settings
    a) Decimation
    b) Maximum depth
3) Logging
4) Source
    a) Main source
    b) Input rate
    c) Stereo
    d) Database
5) RTAB-Map Settings
    a) Detection Rate
6) Memory
    a) STM size
7) Vocabulary
    a) Use odometry features
    b) Feature detector strategy
    c) Max features
    d) Nearest Neighbor Distance Ratio (NNDR)
    e) Nearest Neighbor strategy

8) Graph Optimization
    a) Strategy
    b) Robust graph optimization
    c) Optimize max error
    d) Gravity sigma
9) Visual Registration
    a) Feature matching
    b) NNDR ratio
    c) Feature detector
    d) Max features
10) Optimizer
    a) Variance ignored
    b) Optimizer algorithm

Now do a quick breakdown of each parameter mentioned above-

- General Settings (GUI)

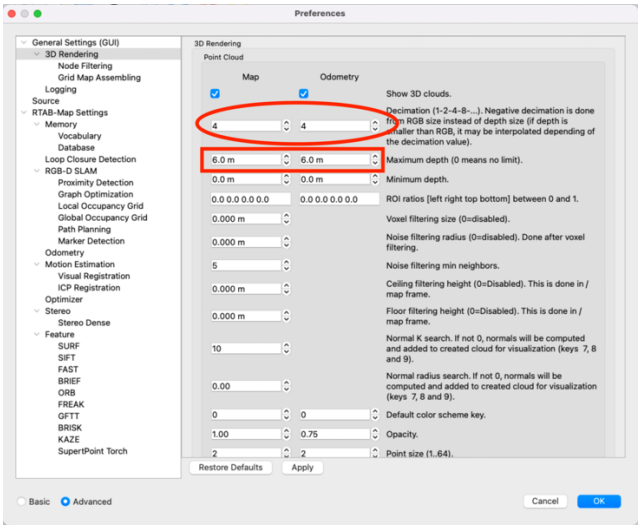
Save/Load Settings – Allows us to save a config file (.ini) and load them whenever needed. A sample file is attached below



- 3D rendering settings

1) Decimation – The default is 4 and shouldn't be changed lower decimations increases the computational power and anything higher results in an unrefined PCL.
2) Maximum depth – The default is set to 0 (infinite), setting this value to 6-8 ensures the best rendering of the PCL. Keeping the value any higher results in the PCL going all over the place and setting a lower value doesn't render the complete PLC

A sample dialog box of the 3D rendering is below



- Logging

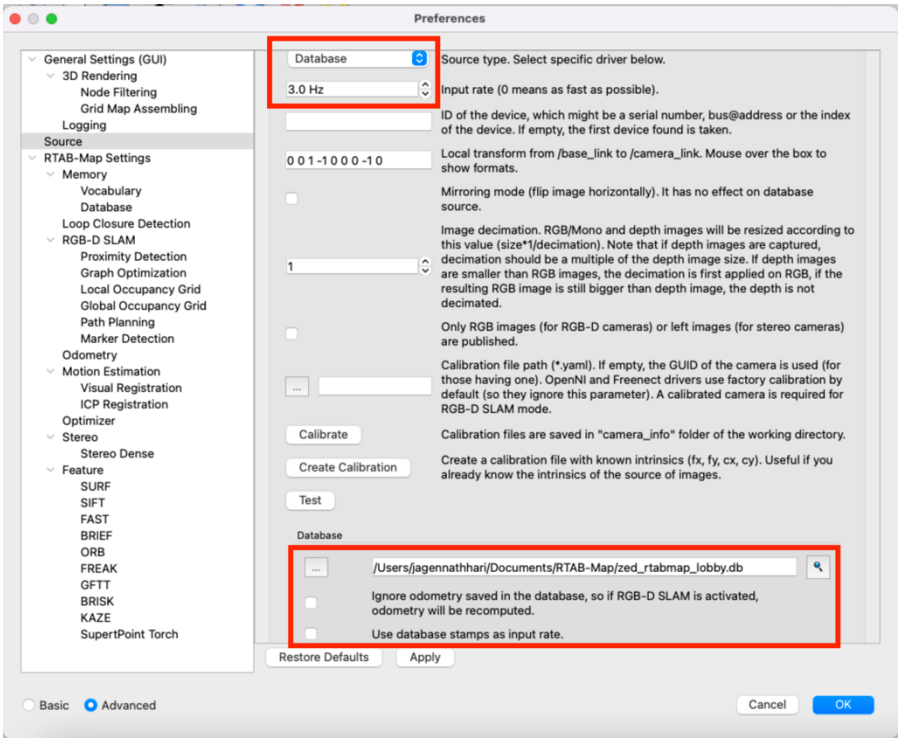
Very Useful for debugging during code changes of the RTAB-MAP build

- Source

1) Main Source – A selection can be made between Camera and database. Camera has three modes, RGB, RGB-D and stereo. The Zed M camera uses stereo camera. For using the video feed from past mapping session database can be selected in the main source.

2) Input rate – For active mapping sessions best to choose the input rate as 0 which is the default. For database mode best choose 3Hz speed to accelerate mapping session.
3) Stereo – Appropriate drivers must be selected for the stereo driver choose ZED SDK while using the ZED M camera. For IMU data again use the ZED SDK while using the ZED M camera.
4) Database – A file must be provided so source a past database file. Always deselect "ignore odometry saved in database" as selecting this option will start RTABMAP visual odometry and causes odometry to be lost while mapping through database. For an accelerated mapping session for the database deselect "use database stamps as input rate" and change input rate above as mentioned.

Given below screenshot of database mode



- RTAB-Map Settings

Detection Rate – On default it is at 0Hz better leave it unchanged as RTAB-MAP will automatically adjust the detection rate this way during active mapping sessions. Keeping this value too high or too low may cause map update rate to be affected.

- Memory

STM size – On default it is kept at 10, change this value only if the velocity of the camera is too slow.

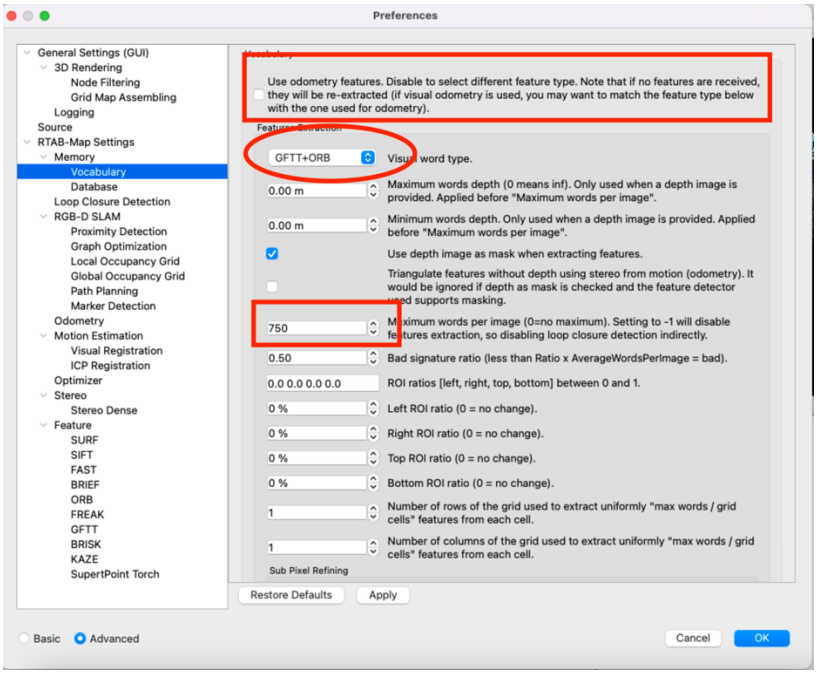
- Vocabulary

1) Use odometry features – Deselect the option as the odometry used is coming from the ZED M camera's IMU.
2) Feature extraction – There are 15 different feature extraction techniques
   a) SURF
   b) SIFT
   c) ORB
   d) FAST + FREAK
   e) FAST + BRIEF
   f) GFTT + FREAK
   g) GFTT + BRIEF
   h) BRISK
   i) GFTT + ORB
   j) KAZE
   k) ORB OCTREE
   l) SURF + FREAK
   m) Super point Touch
   n) GFTT + DAISY
   o) SURF + DAISY

Among these 15 the best feature method was **GFTT + ORB** which gives a good balance for both outdoor and indoor environments.

3) Max features – This value is set at 500 as default, the best value is around 750. Having infinite features cause computation power to increase while giving those features which around corners or edges.
4) NNDR ratio – By default it is set at 0.80, decreasing this value causes the loop closures to not get detected.
5) Nearest Neighbor Strategy – This is the strategy to find nearest neighbors while quantizing the descriptors. By default, it is at FLANN KDTree, other FLANN methods do not cause any difference. Changing to a brute force method causes computational power to increase for certain maps.

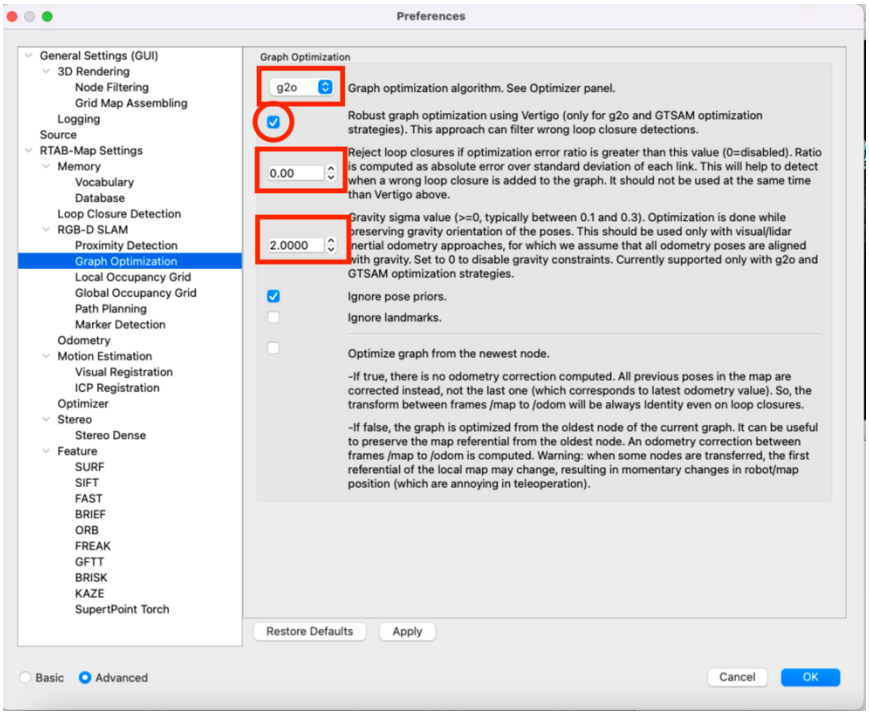
Given below are the best parameters



- Graph Optimization

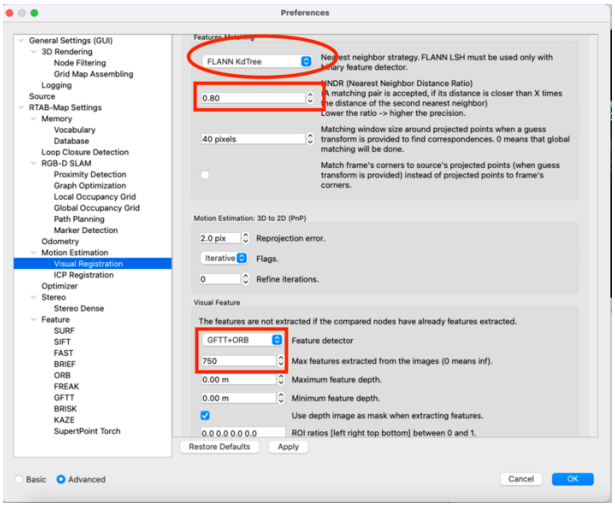
1) Algorithm – By default it is set to TORO, though g2o gives the best result.
2) Robust graph optimization – It is unchecked by default, setting to true gives better map optimization.

3) Optimize max error – This by default is set to 3, change this to 0 for using robust optimization mentioned above.
4) Gravity sigma – A value of 3 is set by default, no considerable change was noticed by reducing or increasing this value.



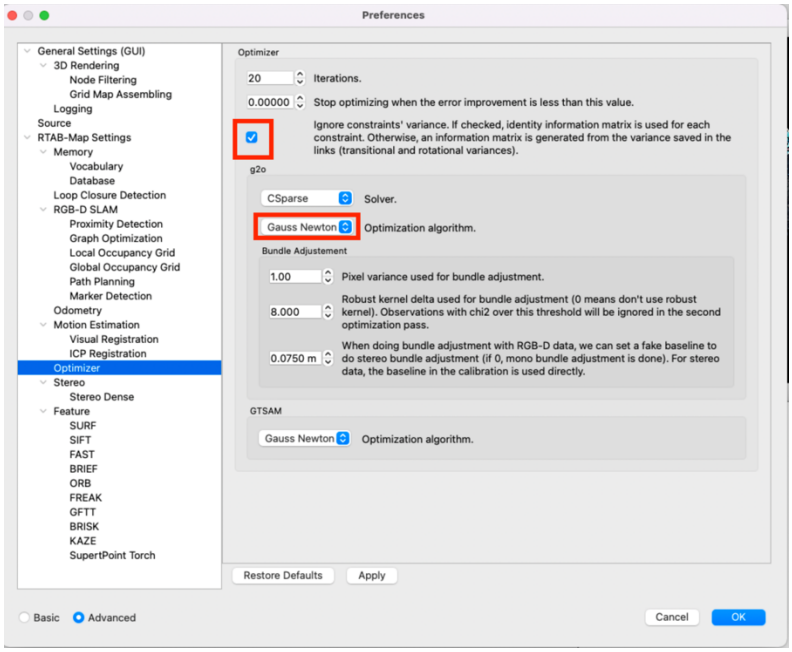
- Visual Registration

Set the same parameters used for vocabulary to avoid recomputing the set parameters.



- Optimizer

1) Variance Ignored – A key parameter with determines whether actual loop closures are accepted or not. Set this to true/checked to ensure loop closures.
2) Optimizer algorithm – By default it is on Levenberg changing it to Gauss Newton ensures better loop closures.



Chapter 3
Mapping Results

The successful mapping sessions are given below with the parameters mentioned above were used to generate these maps.

5 MetroTech outdoor

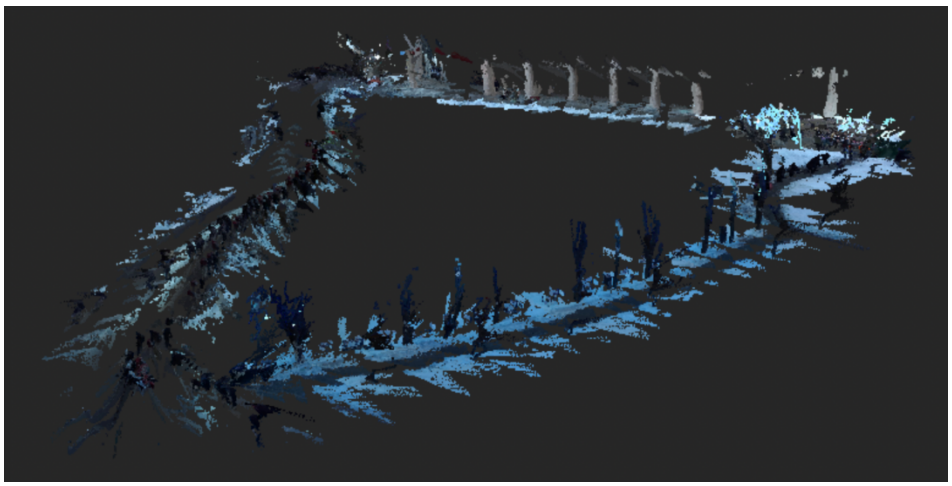


Figure Point Cloud

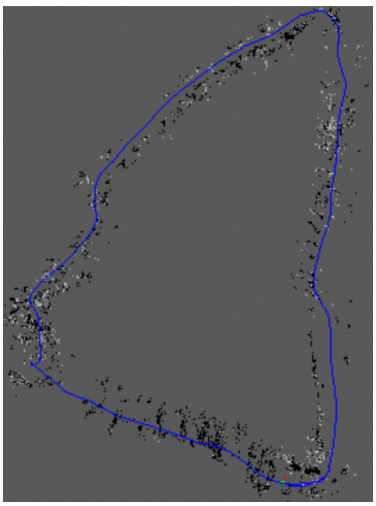


Figure Graph

Parameters	Value
Visual feature strategy	GFTT + ORB
Max features	750
Map optimizer	g2o
Robust Graph Optimization by vertigo	True
Optimizer Max error	0
Gravity sigma	1
Variance ignored	True*
g2o Optimization algorithm	Gauss Newton

Library indoor

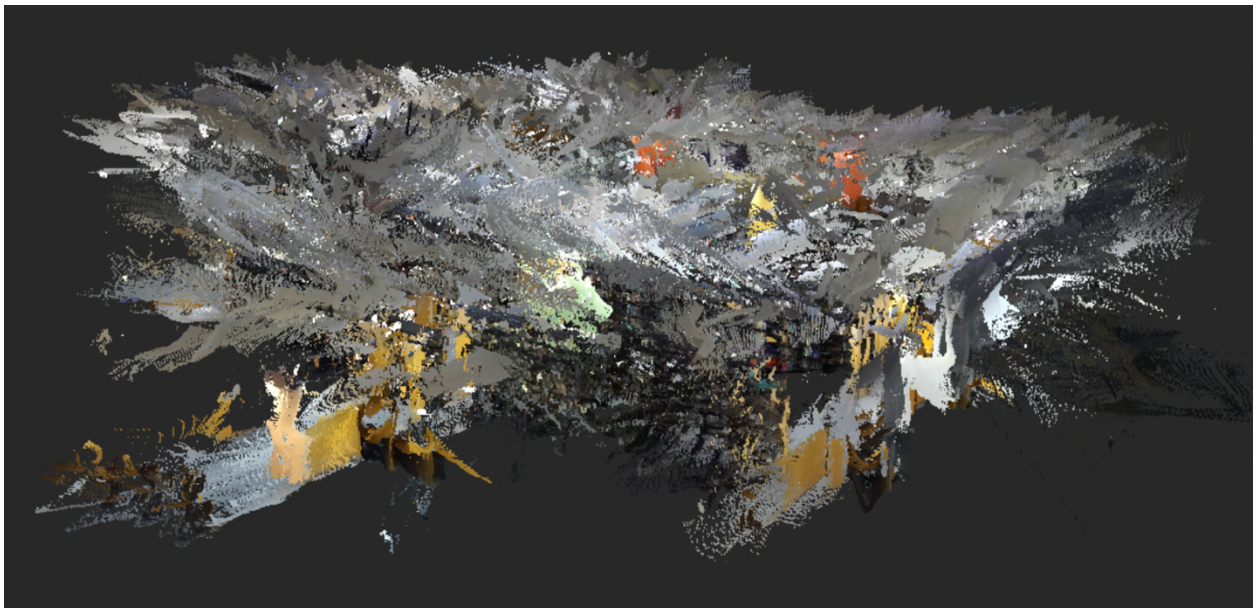


Figure Point Cloud

Parameters	Value
Visual feature strategy	GFTT + ORB
Max features	750
Map optimizer	g2o
Robust Graph Optimization by vertigo	True
Optimizer Max error	0
Gravity sigma	1
Variance ignored	True*
g2o Optimization algorithm	Gauss Newton

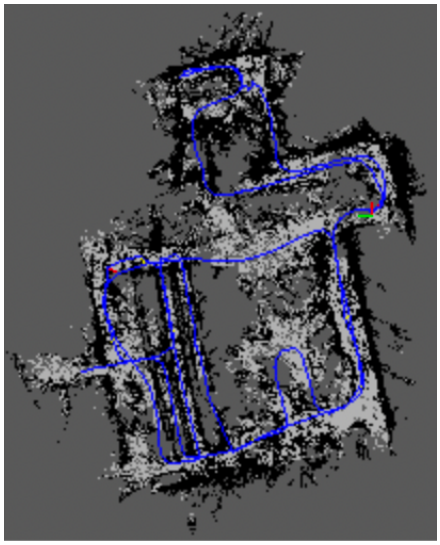


Figure Graph

Lobby indoor

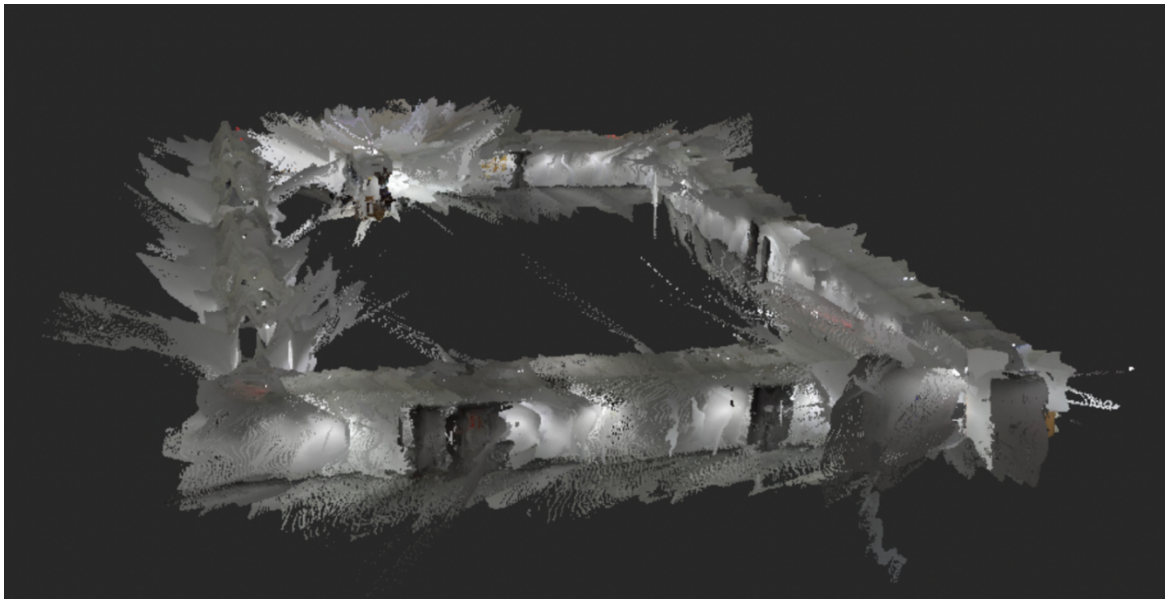


Figure Point Cloud

Parameters	Value
Visual feature strategy	GFTT + ORB
Max features	750
Map optimizer	g2o
Robust Graph Optimization by vertigo	True
Optimizer Max error	0
Gravity sigma	1
Variance ignored	True*
g2o Optimization algorithm	Gauss Newton

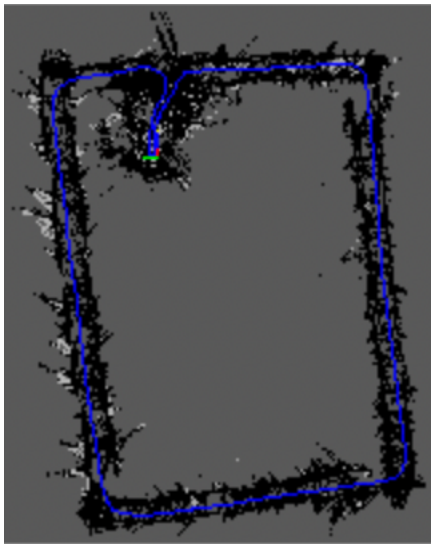


Figure Graph

Chapter 5
Changing RTAB-MAP Build

There are a few successful changes to the build of RTAB-MAP, these changes include-

- Adding a custom feature detector strategy
- Changing descriptor size
- Adding a custom feature matching technique
-

The implementation of these is given below-

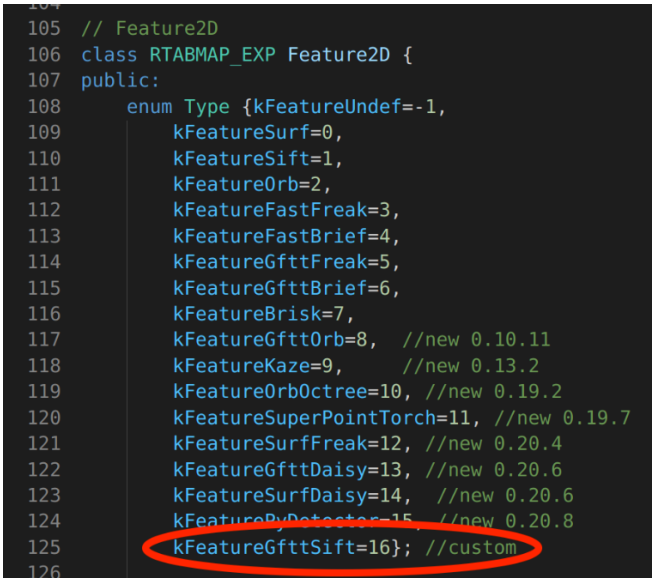
1. Adding a custom feature detector strategy

The procedure is by changing the following files in the build

- Features2d.h & Features2d.cpp (main file for adding custom detector)
- PreferencesDialog.cpp & preferencesDialog.ui (GUI changes to RTAB-MAP)

Features2d.h

Add new feature detector in the Enum type to assign it to a new value.



In switch case, add a case for new feature detector. In our case it is “KFeatureGfttSift” and it returns “GFTT+SIFT”.

```
127     static std::string typeName(Type type)
128     {
129         switch(type){
130         case kFeatureSurf:
131             return "SURF";
132         case kFeatureSift:
133             return "SIFT";
134         case kFeatureOrb:
135             return "ORB";
136         case kFeatureFastFreak:
137             return "FAST+FREAK";
138         case kFeatureFastBrief:
139             return "FAST+BRIEF";
140         case kFeatureGfttFreak:
141             return "GFTT+Freak";
142         case kFeatureGfttBrief:
143             return "GFTT+Brief";
144         case kFeatureBrisk:
145             return "BRISK";
146         case kFeatureGfttOrb:
147             return "GFTT+ORB";
148         case kFeatureKaze:
149             return "KAZE";
150         case kFeatureOrbOctree:
151             return "ORB-OCTREE";
152         case kFeatureSuperPointTorch:
153             return "SUPERPOINT";
154         case kFeatureSurfFreak:
155             return "SURF+Freak";
156         case kFeatureGfttDaisy:
157             return "GFTT+Daisy";
158         case kFeatureSurfDaisy:
159             return "SURF+Daisy";
160         case kFeatureGfttSift: //custom
161             return "GFTT+SIFT";
162         default:
163             return "Unknown";
164         }
165     }
```

Initialize a new class for custom feature detector which includes a function to generate descriptors and has all parameters required to generate keypoints.

```
659 //GFTT_SIFT (Custom)
660 class RTABMAP_EXP GFTT_SIFT : public GFTT
661 {
662 public:
663     GFTT_SIFT(const ParametersMap & parameters = ParametersMap());
664     virtual ~GFTT_SIFT();
665
666     virtual void parseParameters(const ParametersMap & parameters);
667     virtual Feature2D::Type getType() const {return kFeatureGfttSift;}
668
669 private:
670     virtual cv::Mat generateDescriptorsImpl(const cv::Mat & image, std::vector<cv::KeyPoint> & keypoints) const;
671
672 private:
673     int nOctaveLayers_;
674     double contrastThreshold_;
675     double edgeThreshold_;
676     double sigma_;
677     bool rootSIFT_;
678
679     cv::Ptr<CV_SIFT> _sift;
680 };
681
682
683 }
684
685 #endif /* FEATURES2D_H_ */
686
```

Features2d.cpp

The major change in here implementing a new feature detector called “GFTT+SIFT” which is creating a new class which inherits the GFTT public members for generating key points and using the SIFT function in OpenCV to compute the descriptors. The descriptors are returned for further computation for RTAB-MAP for certain functions like motion estimation and loop closure detection.

```
2216 //////////////////////////
2217 //GFTT-SIFT (custom)
2218 //////////////////////////
2219
2220
2221 GFTT_SIFT::GFTT_SIFT(const ParametersMap & parameters) :
2222     GFTT(parameters),
2223     nOctaveLayers_(Parameters::defaultSIFTNOctaveLayers()),
2224     contrastThreshold_(Parameters::defaultSIFTContrastThreshold()),
2225     edgeThreshold_(Parameters::defaultSIFTEdgeThreshold()),
2226     sigma_(Parameters::defaultSIFTSigma()),
2227     rootSIFT_(Parameters::defaultSIFTRootSIFT())
2228 {
2229     parseParameters(parameters);
2230 }
2231 GFTT_SIFT::~GFTT_SIFT()
2232 {
2233 }
2234
2235 void GFTT_SIFT::parseParameters(const ParametersMap & parameters)
2236 {
2237     GFTT::parseParameters(parameters);
2238     Parameters::parse(parameters, Parameters::kSIFTContrastThreshold(), contrastThreshold_);
2239     Parameters::parse(parameters, Parameters::kSIFTEdgeThreshold(), edgeThreshold_);
2240     Parameters::parse(parameters, Parameters::kSIFTNOctaveLayers(), nOctaveLayers_);
2241     Parameters::parse(parameters, Parameters::kSIFTSigma(), sigma_);
2242     Parameters::parse(parameters, Parameters::kSIFTRootSIFT(), rootSIFT_);
2243
2244
2245 #if CV_MAJOR_VERSION < 3 || (CV_MAJOR_VERSION == 4 && CV_MINOR_VERSION <= 3) || (CV_MAJOR_VERSION == 3 && (CV_MINOR_VERSION < 4 || (CV_MINOR_VERSION==4 && CV_SUBMINOR_VERSION<1)
2246 #ifdef RTABMAP_NONFREE
2247 #if CV_MAJOR_VERSION < 3
2248     _sift = cv::Ptr<CV_SIFT>(new CV_SIFT(this->getMaxFeatures(), nOctaveLayers_, contrastThreshold_, edgeThreshold_, sigma_));
2249 #else
2250     _sift = CV_SIFT::create(this->getMaxFeatures(), nOctaveLayers_, contrastThreshold_, edgeThreshold_, sigma_);
2251 #endif
2252 #else
2253     UWARN("RTAB-Map is not built with OpenCV nonfree module so SIFT cannot be used!");
2254 #endif
2255 #else // >=4.4, >=3.4.11
2256     _sift = CV_SIFT::create(this->getMaxFeatures(), nOctaveLayers_, contrastThreshold_, edgeThreshold_, sigma_);
2257 #endif
2258 }
```

```
2261 cv::Mat GFTT_SIFT::generateDescriptorsImpl(const cv::Mat & image, std::vector<cv::KeyPoint> & keypoints) const
2262 {
2263     UASSERT(!image.empty() && image.channels() == 1 && image.depth() == CV_8U);
2264     cv::Mat descriptors;
2265 #if CV_MAJOR_VERSION < 3 || (CV_MAJOR_VERSION == 4 && CV_MINOR_VERSION <= 3) || (CV_MAJOR_VERSION == 3 && (CV_MINOR_VERSION < 4 || (CV_MINOR_VERSION==4 && CV_SUBMINOR_VERSION<1)
2266 #ifdef RTABMAP_NONFREE
2267     _sift->compute(image, keypoints, descriptors);
2268 #else
2269     UWARN("RTAB-Map is not built with OpenCV nonfree module so SIFT cannot be used!");
2270 #endif
2271 #else // >=4.4, >=3.4.11
2272     _sift->compute(image, keypoints, descriptors);
2273 #endif
2274     if( rootSIFT_ && !descriptors.empty())
2275     {
2276         UDEBUG("Performing RootSIFT...");
2277         // see http://www.pyimagesearch.com/2015/04/13/implementing-rootsift-in-python-and-opencv/
2278         // apply the Hellinger kernel by first L1-normalizing and taking the
2279         // square-root
2280         for(int i=0; i<descriptors.rows; ++i)
2281         {
2282             // By taking the L1 norm, followed by the square-root, we have
2283             // already L2 normalized the feature vector and further normalization
2284             // is not needed.
2285             descriptors.row(i) = descriptors.row(i) / cv::sum(descriptors.row(i))[0];
2286             cv::sqrt(descriptors.row(i), descriptors.row(i));
2287         }
2288     }
2289     return descriptors;
2290 }
2291
```

PreferencesDialog.cpp

```
// SIFT
#if CV_MAJOR_VERSION < 3 || (CV_MAJOR_VERSION == 4 && CV_MINOR_VERSION <= 3) || (CV_MAJOR_VERSION == 3 && (CV_MINOR_VERSION < 4 || (CV_MINOR_VERSION==4 && CV_SUBMINOR_VERSION<1
#ifndef RTABMAP_NONFREE
    _ui->comboBox_detector_strategy->setItemData(1, 0, Qt::UserRole - 1);
    _ui->comboBox_detector_strategy->setItemData(16, 0, Qt::UserRole - 1);
    _ui->vis_feature_detector->setItemData(16, 0, Qt::UserRole - 1);
#endif
#endif
```

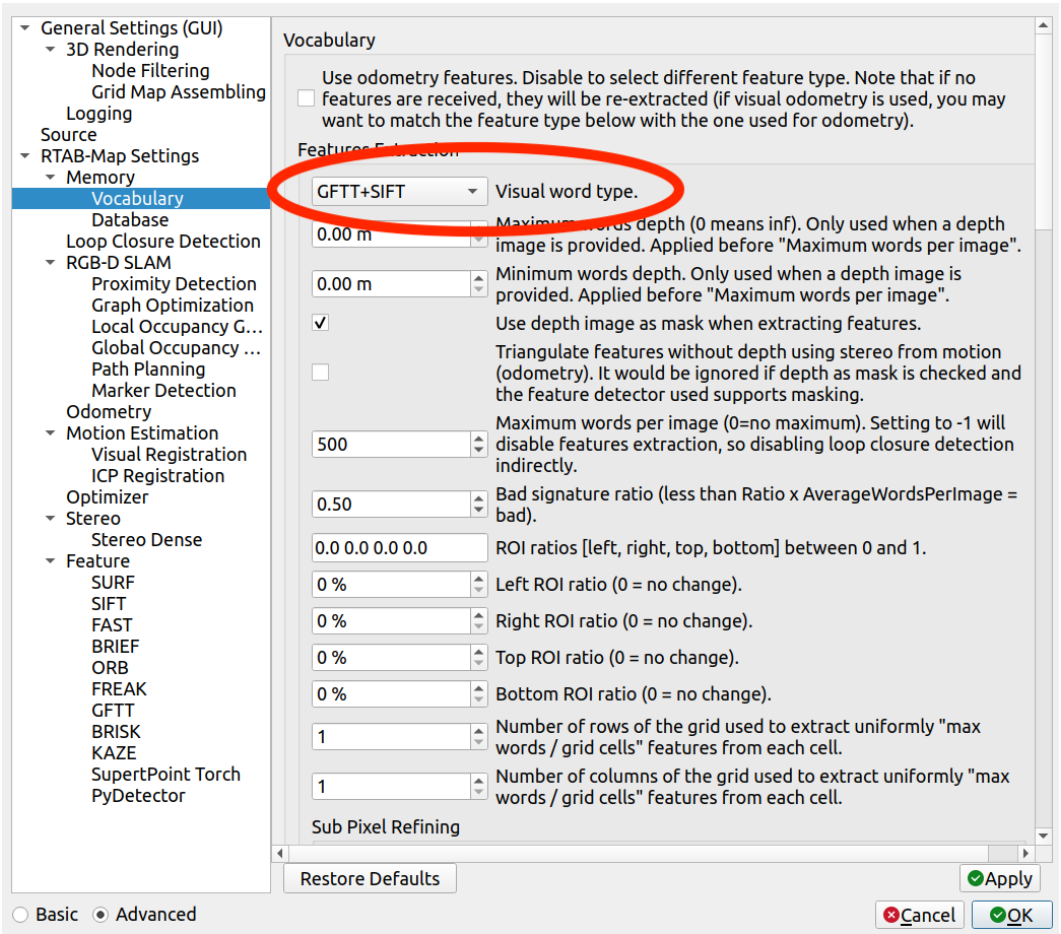
preferencesDialog.ui

```
<item>
 <property name="text">
  <string>GFTT+SIFT</string>
 </property>
</item>
</widget>
</item>
```

Add at two locations for Kp/Vis

This essentially changes the GUI of RTAB-MAP to accommodate the new changes else the feature method will go to default which is SURF (if available).

Result

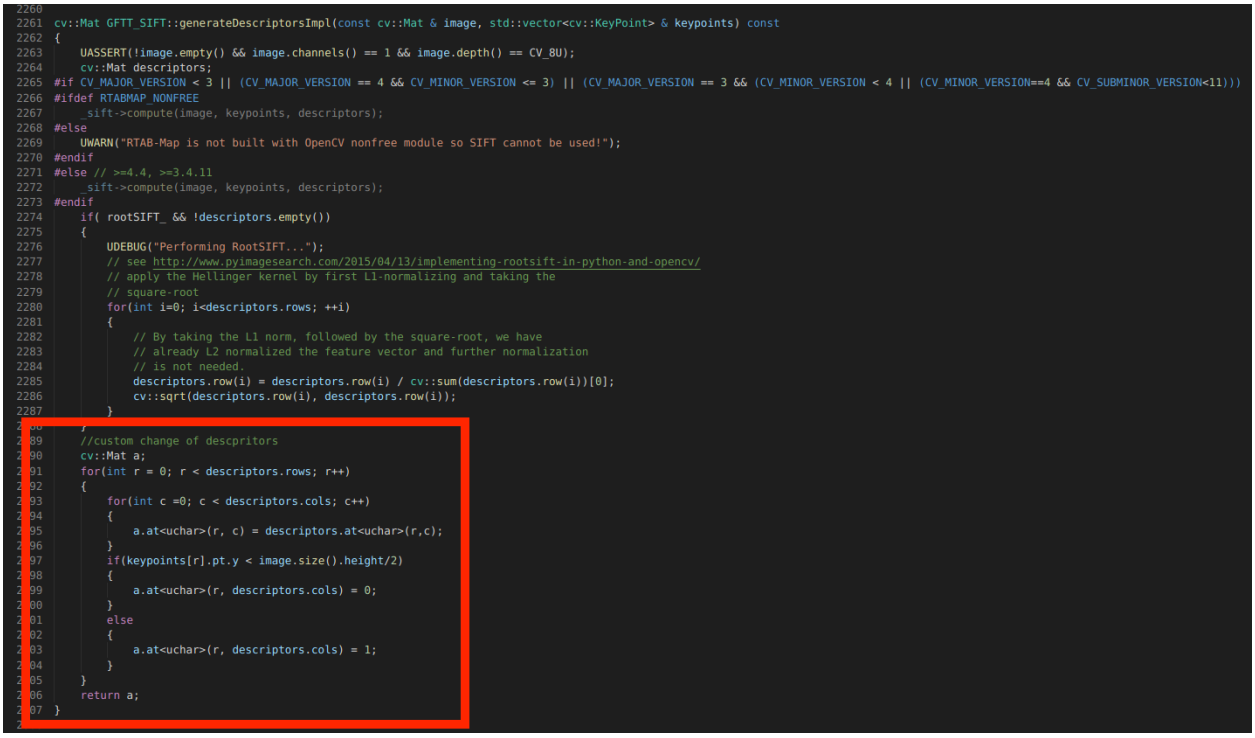


2. Changing descriptor size

The idea is creating a fake semantic class, for test purposes those keypoints above half the height of the image have been given a class of 1 and those below 0. For test purposes the fake semantic class have been added at the last column of the descriptor, values of 1 or 0 are small so they don't interfere with the regular functioning of RTAB-MAP, moreover these changes have only been added to the custom feature detector mentioned above so that the main build can still be accessed and operated normally if needed. These changes are not visible to the eye as they are completely internal, but a method for visualization can be incorporated to start coloring the keypoints depending on their location has been devised but yet to be implemented.

Features2d.cpp

Create a new matrix with extra column to add semantic information.



```
cv::Mat GFTT_SIFT::generateDescriptorsImpl(const cv::Mat & image, std::vector<cv::KeyPoint> & keypoints) const
{
    UASSERT(!image.empty() && image.channels() == 1 && image.depth() == CV_8U);
    cv::Mat descriptors;
#if CV_MAJOR_VERSION < 3 || (CV_MAJOR_VERSION == 4 && CV_MINOR_VERSION <= 3) || (CV_MAJOR_VERSION == 3 && (CV_MINOR_VERSION < 4 || (CV_MINOR_VERSION==4 && CV_SUBMINOR_VERSION<11)))
#ifdef RTABMAP_NONFREE
    _sift->compute(image, keypoints, descriptors);
#else
    UWARN("RTAB-Map is not built with OpenCV nonfree module so SIFT cannot be used!");
#endif
#else // >=4.4, >=3.4.11
    _sift->compute(image, keypoints, descriptors);
#endif
    if( rootSIFT_ && !descriptors.empty())
    {
        UDEBUG("Performing RootSIFT...");
        // see http://www.pyimagesearch.com/2015/04/13/implementing-rootsift-in-python-and-opencv/
        // apply the Hellinger kernel by first L1-normalizing and taking the
        // square-root
        for(int i=0; i<descriptors.rows; ++i)
        {
            // By taking the L1 norm, followed by the square-root, we have
            // already L2 normalized the feature vector and further normalization
            // is not needed.
            descriptors.row(i) = descriptors.row(i) / cv::sum(descriptors.row(i))[0];
            cv::sqrt(descriptors.row(i), descriptors.row(i));
        }
    }
    //custom change of descpritors
    cv::Mat a;
    for(int r = 0; r < descriptors.rows; r++)
    {
        for(int c =0; c < descriptors.cols; c++)
        {
            a.at<uchar>(r, c) = descriptors.at<uchar>(r,c);
        }
        if(keypoints[r].pt.y < image.size().height/2)
        {
            a.at<uchar>(r, descriptors.cols) = 0;
        }
        else
        {
            a.at<uchar>(r, descriptors.cols) = 1;
        }
    }
    return a;
}
```

3. Adding a custom feature matching technique

Moving forward a new matching strategy must be formulated to somehow use the semantic class meaningfully, but before that a new feature matching must be added. To start with a FLANNKdTree has been copied and added back into RTAB-MAP for selection, this obviously can be fully customized later to incorporate the semantic classes. The procedure is to add this given below

- VWDictionary.h & VWDictionary.cpp (main file for adding custom matcher)
- PreferencesDialog.cpp & preferencesDialog.ui (GUI changes to RTAB-MAP)

VWDictionary.h

Add a new ENUM type for new feature matching technique and create new switch case for the same.

```
46  class RTABMAP_EXP VWDictionary
47  {
48  public:
49      enum NNStrategy{
50          kNNFlannNaive,
51          kNNFlannKdTree,
52          kNNFlannLSH,
53          kNNBruteForce,
54
55          kNNnewFlannKdTree, //custom
56
57      static const int ID_START;
58      static const int ID_INVALID;
59      static std::string nnStrategyName(NNStrategy strategy)
60      {
61          switch(strategy) {
62          case kNNFlannNaive:
63              return "FLANN NAIVE";
64          case kNNFlannKdTree:
65              return "FLANN KD-TREE";
66          case kNNFlannLSH:
67              return "FLANN LSH";
68          case kNNBruteForce:
69              return "BRUTE FORCE";
70          case kNNBruteForceGPU:
71
72          case kNNnewFlannKdTree: //custom
73              return "New FLANN KD-TREE";
74
75              return "Unknown";
76          }
77      }
78
```

VWDictionary.cpp

```
    // Verifying hypotheses strategy
    bool treeUpdated = false;
    if((iter=parameters.find(Parameters::kKpNNStrategy())) != parameters.end())
    {
        NNStrategy nnStrategy = (NNStrategy)std::atoi((*iter).second.c_str());
        treeUpdated = this->setNNStrategy(nnStrategy);
    }
    if(!treeUpdated && byteToFloat!=_byteToFloat && (_strategy == kNNFlannKdTree || _strategy == kNNnewFlannKdTree)) //custom
    {
        UWARN("KDTree: Binary to Float conversion approach has changed, re-initialize kd-tree.");
        _dataTree = cv::Mat();
        _notIndexedWords = uKeysSet(_visualWords);
        _removedIndexedWords.clear();
        this->update();
    }
```

```
if(_notIndexedWords.size() || _visualWords.size() == 0 || _removedIndexedWords.size())
{
    if(_incrementalFlann &&
       (_strategy < kNNBruteForce || _strategy == kNNnewFlannKdTree) &&        //custom
       _visualWords.size())
    {
        ULOGGER_DEBUG("Incremental FLANN: Removing %d words...", (int)_removedIndexedWords.size());
        for(std::set<int>::iterator iter=_removedIndexedWords.begin(); iter!=_removedIndexedWords.end(); ++iter)
        {
            UASSERT(uContains(_mapIdIndex, *iter));
            UASSERT(uContains(_mapIndexId, _mapIdIndex.at(*iter)));
            _flannIndex->removePoint(_mapIdIndex.at(*iter));
            _mapIndexId.erase(_mapIdIndex.at(*iter));
            _mapIdIndex.erase(*iter);
        }
        ULOGGER_DEBUG("Incremental FLANN: Removing %d words... done!", (int)_removedIndexedWords.size());
```

```
    if(_notIndexedWords.size())
    {
        ULOGGER_DEBUG("Incremental FLANN: Inserting %d words...", (int)_notIndexedWords.size());
        for(std::set<int>::iterator iter=_notIndexedWords.begin(); iter!=_notIndexedWords.end(); ++iter)
        {
            VisualWord* w = uValue(_visualWords, *iter, (VisualWord*)0);
            UASSERT(w);

            cv::Mat descriptor;
            if(w->getDescriptor().type() == CV_8U)
            {
                useDistanceL1_ = true;
                if(_strategy == kNNFlannKdTree || _strategy == kNNnewFlannKdTree) //custom
                {
                    descriptor = convertBinTo32F(w->getDescriptor(), byteToFloat);
                }
                else
                {
                    descriptor = w->getDescriptor();
                }
            }
```

```
int index = 0;
if(!_flannIndex->isBuilt())
{
    UDEBUG("Building FLANN index...");
    switch(_strategy)
    {
    case kNNFlannNaive:
        _flannIndex->buildLinearIndex(descriptor, useDistanceL1_, _rebalancingFactor);
        break;
    case kNNFlannKdTree:
        UASSERT_MSG(descriptor.type() == CV_32F, "To use KdTree dictionary, float descriptors are required!");
        _flannIndex->buildKDTreeIndex(descriptor, KDTREE_SIZE, useDistanceL1_, _rebalancingFactor);
        break;
    case kNNFlannLSH:
        UASSERT_MSG(descriptor.type() == CV_8U, "To use LSH dictionary, binary descriptors are required!");
        _flannIndex->buildLSHIndex(descriptor, 12, 20, 2, _rebalancingFactor);
        break;
    case kNNnewFlannKdTree: //custom
        UASSERT_MSG(descriptor.type() == CV_32F, "To use KdTree dictionary, float descriptors are required!");
        _flannIndex->buildKDTreeIndex(descriptor, KDTREE_SIZE, useDistanceL1_, _rebalancingFactor);
        break;
    default:
        UFATAL("Not supposed to be here!");
        break;
    }
    UDEBUG("Building FLANN index... done!");
}
```

```
else if((_strategy >= kNNBruteForce && _strategy != kNNnewFlannKdTree) && //custom
        _notIndexedWords.size() &&
        removedIndexedWords.size() == 0 &&
        _dataTree.rows)
{
    //just add not indexed words
    int i = _dataTree.rows;
    _dataTree.reserve(_dataTree.rows + _notIndexedWords.size());
    for(std::set<int>::iterator iter=_notIndexedWords.begin(); iter!=_notIndexedWords.end(); ++iter)
    {
        VisualWord* w = uValue(_visualWords, *iter, (VisualWord*)0);
        UASSERT(w);
        UASSERT(w->getDescriptor().cols == _dataTree.cols);
        UASSERT(w->getDescriptor().type() == _dataTree.type());
        _dataTree.push_back(w->getDescriptor());
        _mapIndexId.insert(_mapIndexId.end(), std::pair<int, int>(i, w->id()));
        std::pair<std::map<int, int>::iterator, bool> inserted = _mapIdIndex.insert(std::pair<int, int>(w->id(), i));
        UASSERT(inserted.second);
        ++i;
    }
}
```

```
int dim = _visualWords.begin()->second->getDescriptor().cols;
int type;
if(_visualWords.begin()->second->getDescriptor().type() == CV_8U)
{
    useDistanceL1_ = true;
    if(_strategy == kNNFlannKdTree || _strategy == kNNnewFlannKdTree) //custom
    {
        type = CV_32F;
        if(!_byteToFloat)
        {
            dim *= 8;
        }
    }
    else
    {
        type = _visualWords.begin()->second->getDescriptor().type();
    }
}
```

```
switch(_strategy)
{
case kNNFlannNaive:
    _flannIndex->buildLinearIndex(_dataTree, useDistanceL1_, _incrementalDictionary&&_incrementalFlann?_rebalancingFactor:1);
    break;
case kNNFlannKdTree:
    UASSERT_MSG(type == CV_32F, "To use KdTree dictionary, float descriptors are required!");
    _flannIndex->buildKDTreeIndex(_dataTree, KDTREE_SIZE, useDistanceL1_, _incrementalDictionary&&_incrementalFlann?_rebalancingFactor:1);
    break;
case kNNFlannLSH:
    UASSERT_MSG(type == CV_8U, "To use LSH dictionary, binary descriptors are required!");
    _flannIndex->buildLSHIndex(_dataTree, 12, 20, 2, _incrementalDictionary&&_incrementalFlann?_rebalancingFactor:1);
    break;
case kNNnewFlannKdTree: //custom
    UASSERT_MSG(type == CV_32F, "To use KdTree dictionary, float descriptors are required!");
    _flannIndex->buildKDTreeIndex(_dataTree, KDTREE_SIZE, useDistanceL1_, _incrementalDictionary&&_incrementalFlann?_rebalancingFactor:1);
    break;
default:
    break;
}
```

```
// now compare with the actual index
cv::Mat descriptors;
if(descriptorsIn.type() == CV_8U)
{
    useDistanceL1_ = true;
    if(_strategy == kNNFlannKdTree || _strategy == kNNnewFlannKdTree) //custom
    {
        descriptors = convertBinTo32F(descriptorsIn, _byteToFloat);
    }
    else
    {
        descriptors = descriptorsIn;
    }
}
else
{
    descriptors = descriptorsIn;
}
```

```
if(_flannIndex->isBuilt() || (!_dataTree.empty() && _dataTree.rows >= (int)k))
{
    //Find nearest neighbors
    UDEBUG("query.total()=%d ", descriptors.rows);

    if(_strategy == kNNFlannNaive || _strategy == kNNFlannKdTree || _strategy == kNNFlannLSH || _strategy == kNNnewFlannKdTree) //custom
    {
        flannIndex->knnSearch(descriptors, results, dists, k, KNN_CHECKS);
    }
    else if(_strategy == kNNBruteForce)
    {
        bruteForce = true;
        cv::BFMatcher matcher(descriptors.type()==CV_8U?cv::NORM_HAMMING:cv::NORM_L2SQR);
        matcher.knnMatch(descriptors, _dataTree, matches, k);
    }
```

```
// now compare with the actual index
cv::Mat query;
if(queryIn.type()==CV_8U)
{
    if(_strategy == kNNFlannKdTree || _strategy == kNNnewFlannKdTree) //custom
    {
        query = convertBinTo32F(queryIn, _byteToFloat);
    }
    else
    {
        query = queryIn;
    }
}
else
```

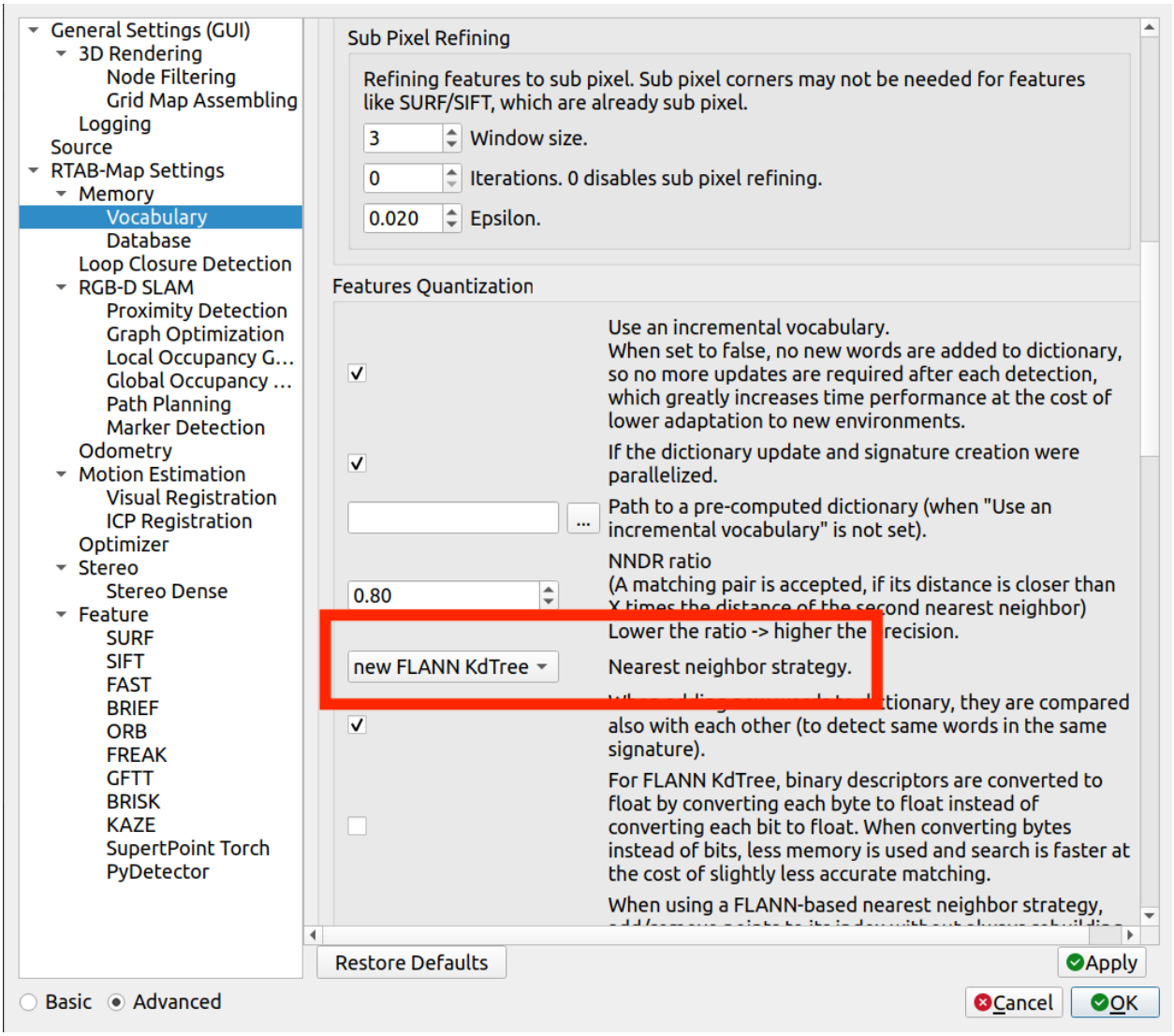
```
if(_flannIndex->isBuilt() || (!_dataTree.empty() && _dataTree.rows >= (int)k))
{
    //Find nearest neighbors
    UDEBUG("query.rows=%d ", query.rows);

    if(_strategy == kNNFlannNaive || _strategy == kNNFlannKdTree || _strategy == kNNFlannLSH || _strategy == kNNnewFlannKdTree) //custom
    {
        _flannIndex->knnSearch(query, results, dists, k, KNN_CHECKS);
    }
    else if(_strategy == kNNBruteForce)
    {
        bruteForce = true;
        cv::BFMatcher matcher(query.type()==CV_8U?cv::NORM_HAMMING:cv::NORM_L2SQR);
        matcher.knnMatch(query, _dataTree, matches, k);
    }
```

```
cv::Mat descriptor;
{
    if(_strategy == kNNFlannKdTree || _strategy == kNNnewFlannKdTree) //custom
    {
        descriptor = convertBinTo32F(vw->getDescriptor(), _byteToFloat);
    }
    else
    {
        descriptor = vw->getDescriptor();
    }
}
else
```

All these changes are required to add a new matching strategy. Though these changes are at many there is currently nothing new in this method as it is just a replication of FLANN. The next step would be to implement a Boolean check of the previously added fake semantic classes into the new FLANN strategy.

Result


General Settings (GUI)
3D Rendering
Node Filtering
Grid Map Assembling
Logging
Source
RTAB-Map Settings
Memory
Vocabulary
Database
Loop Closure Detection
RGB-D SLAM
Proximity Detection
Graph Optimization
Local Occupancy G…
Global Occupancy …
Path Planning
Marker Detection
Odometry
Motion Estimation
Visual Registration
ICP Registration
Optimizer
Stereo
Stereo Dense
Feature
SURF
SIFT
FAST
BRIEF
ORB
FREAK
GFTT
BRISK
KAZE
SuperPoint Torch
PyDetector
Sub Pixel Refining
Refining features to sub pixel. Sub pixel corners may not be needed for features like SURF/SIFT, which are already sub pixel.
3
Window size.
0
Iterations. 0 disables sub pixel refining.
0.020
Epsilon.
Features Quantization
Use an incremental vocabulary.
When set to false, no new words are added to dictionary, so no more updates are required after each detection, which greatly increases time performance at the cost of lower adaptation to new environments.
If the dictionary update and signature creation were parallelized.
...
Path to a pre-computed dictionary (when "Use an incremental vocabulary" is not set).
0.80
NNDR ratio
(A matching pair is accepted, if its distance is closer than
second nearest neighbor)
Lower the ratio -> higher the
recision.
new FLANN KdTree
Nearest neighbor strategy.
tionary, they are compared also with each other (to detect same words in the same signature).
For FLANN KdTree, binary descriptors are converted to float by converting each byte to float instead of converting each bit to float. When converting bytes instead of bits, less memory is used and search is faster at the cost of slightly less accurate matching.
When using a FLANN-based nearest neighbor strategy,
Restore Defaults
Apply
Basic
Advanced
Cancel
OK

REFERENCES –

RTABMAP –

Research papers:

https://introlab.3it.usherbrooke.ca/mediawiki-introlab/images/7/7a/Labbe18JFR_preprint.pdf

M. Labbé and F. Michaud, "Appearance-Based Loop Closure Detection for Online Large-Scale and Long-Term Operation," in *IEEE Transactions on Robotics*, vol. 29, no. 3, pp. 734-745, 2013.

Tutorials:

Tutorials · introlab/rtabmap Wiki · GitHub

Visual Features and descriptors –

Visual Feature Part 1: Computing Keypoints (Cyrill Stachniss)