Cartographer Documentation

The Cartographer Authors

Mar 11, 2021

Contents

CHAPTER 1

Configuration

1.1 cartographer.common.proto.CeresSolverOptions

bool use_nonmonotonic_steps Configure the Ceres solver. See the Ceres documentation for more information: https://code.google.com/p/ceres-solver/

int32 max_num_iterations Not yet documented.

int32 num_threads Not yet documented.

1.2 cartographer.mapping.pose_graph.proto.ConstraintBuilderOptions

double sampling_ratio A constraint will be added if the proportion of added constraints to potential constraints drops below this number.

double max_constraint_distance Threshold for poses to be considered near a submap.

double min_score Threshold for the scan match score below which a match is not considered. Low scores indicate that the scan and map do not look similar.

double global_localization_min_score Threshold below which global localizations are not trusted.

double loop_closure_translation_weight Weight used in the optimization problem for the translational component of loop closure constraints.

double loop_closure_rotation_weight Weight used in the optimization problem for the rotational component of loop closure constraints.

bool log_matches If enabled, logs information of loop-closing constraints for debugging.

cartographer.mapping_2d.scan_matching.proto.FastCorrelativeScanMatcherOptions fast_correlative_scan_matcher_options Options for the internally used scan matchers.

cartographer.mapping_2d.scan_matching.proto.CeresScanMatcherOptions ceres_scan_matcher_options Not yet documented.

cartographer.mapping_3d.scan_matching.proto.FastCorrelativeScanMatcherOptions fast_correlative_scan_matcher_options_3
Not yet documented.

cartographer.mapping_3d.scan_matching.proto.CeresScanMatcherOptions ceres_scan_matcher_options_3d
Not yet documented.

1.3 cartographer.mapping.pose_graph.proto.OptimizationProblemOptions

double huber_scale Scaling parameter for Huber loss function.

double acceleration_weight Scaling parameter for the IMU acceleration term.

double rotation_weight Scaling parameter for the IMU rotation term.

double local_slam_pose_translation_weight Scaling parameter for translation between consecutive nodes based on the local SLAM pose.

double local_slam_pose_rotation_weight Scaling parameter for rotation between consecutive nodes based on the local SLAM pose.

double odometry_translation_weight Scaling parameter for translation between consecutive nodes based on the odometry.

double odometry_rotation_weight Scaling parameter for rotation between consecutive nodes based on the odometry.

double fixed_frame_pose_translation_weight Scaling parameter for the FixedFramePose translation.

double fixed_frame_pose_rotation_weight Scaling parameter for the FixedFramePose rotation.

bool log_solver_summary If true, the Ceres solver summary will be logged for every optimization.

cartographer.common.proto.CeresSolverOptions ceres_solver_options Not yet documented.

1.4 cartographer.mapping.proto.MapBuilderOptions

bool use_trajectory_builder_2d Not yet documented.

bool use_trajectory_builder_3d Not yet documented.

int32 num_background_threads Number of threads to use for background computations.

cartographer.mapping.proto.PoseGraphOptions pose_graph_options Not yet documented.

1.5 cartographer.mapping.proto.MotionFilterOptions

double max_time_seconds Threshold above which range data is inserted based on time.

double max_distance_meters Threshold above which range data is inserted based on linear motion.

double max_angle_radians Threshold above which range data is inserted based on rotational motion.

1.6 cartographer.mapping.proto.PoseGraphOptions

int32 optimize_every_n_nodes Online loop closure: If positive, will run the loop closure while the map is built.

cartographer.mapping.pose_graph.proto.ConstraintBuilderOptions constraint_builder_options Options for the constraint builder.

double matcher_translation_weight Weight used in the optimization problem for the translational component of non-loop-closure scan matcher constraints.

double matcher_rotation_weight Weight used in the optimization problem for the rotational component of non-loop-closure scan matcher constraints.

cartographer.mapping.pose_graph.proto.OptimizationProblemOptions optimization_problem_options Options for the optimization problem.

int32 max_num_final_iterations Number of iterations to use in 'optimization_problem_options' for the final optimization.

double global_sampling_ratio Rate at which we sample a single trajectory's nodes for global localization.

bool log_residual_histograms Whether to output histograms for the pose residuals.

double global_constraint_search_after_n_seconds If for the duration specified by this option no global contraint has been added between two trajectories, loop closure searches will be performed globally rather than in a smaller search window.

1.7 cartographer.mapping.proto.TrajectoryBuilderOptions

cartographer.mapping_2d.proto.LocalTrajectoryBuilderOptions trajectory_builder_2d_options Not yet documented.

cartographer.mapping_3d.proto.LocalTrajectoryBuilderOptions trajectory_builder_3d_options Not yet documented.

bool pure_localization Not yet documented.

1.8 cartographer.mapping_2d.proto.LocalTrajectoryBuilderOptions

float min_range Rangefinder points outside these ranges will be dropped.

float max_range Not yet documented.

float min_z Not yet documented.

float max_z Not yet documented.

float missing_data_ray_length Points beyond 'max_range' will be inserted with this length as empty space.

int32 num_accumulated_range_data Number of range data to accumulate into one unwarped, combined range data to use for scan matching.

float voxel_filter_size Voxel filter that gets applied to the range data immediately after cropping.

cartographer.sensor.proto.AdaptiveVoxelFilterOptions adaptive_voxel_filter_options Voxel filter used to compute a sparser point cloud for matching.

cartographer.sensor.proto.AdaptiveVoxelFilterOptions loop_closure_adaptive_voxel_filter_options Voxel filter used to compute a sparser point cloud for finding loop closures.

bool use_online_correlative_scan_matching Whether to solve the online scan matching first using the correlative scan matcher to generate a good starting point for Ceres.

cartographer.mapping_2d.scan_matching.proto.RealTimeCorrelativeScanMatcherOptions real_time_correlative_scan_matche Not yet documented.

cartographer.mapping_2d.scan_matching.proto.CeresScanMatcherOptions ceres_scan_matcher_options Not yet documented.

cartographer.mapping.proto.MotionFilterOptions motion_filter_options Not yet documented.

double imu_gravity_time_constant Time constant in seconds for the orientation moving average based on observed gravity via the IMU. It should be chosen so that the error 1. from acceleration measurements not due to gravity (which gets worse when the constant is reduced) and 2. from integration of angular velocities (which gets worse when the constant is increased) is balanced.

cartographer.mapping_2d.proto.SubmapsOptions submaps_options Not yet documented.

bool use_imu_data True if IMU data should be expected and used.

1.9 cartographer.mapping_2d.proto.RangeDataInserterOptions

double hit_probability Probability change for a hit (this will be converted to odds and therefore must be greater than 0.5).

double miss_probability Probability change for a miss (this will be converted to odds and therefore must be less than 0.5).

bool insert_free_space If 'false', free space will not change the probabilities in the occupancy grid.

1.10 cartographer.mapping_2d.proto.SubmapsOptions

double resolution Resolution of the map in meters.

int32 num_range_data Number of range data before adding a new submap. Each submap will get twice the number of range data inserted: First for initialization without being matched against, then while being matched.

cartographer.mapping_2d.proto.RangeDataInserterOptions range_data_inserter_options Not yet documented.

1.11 cartographer.mapping_2d.scan_matching.proto.CeresScanMatcherOption

double occupied_space_weight Scaling parameters for each cost functor.

double translation_weight Not yet documented.

double rotation_weight Not yet documented.

cartographer.common.proto.CeresSolverOptions ceres_solver_options Configure the Ceres solver. See the Ceres documentation for more information: https://code.google.com/p/ceres-solver/

1.12 cartographer.mapping_2d.scan_matching.proto.FastCorrelativeScanMatch

double linear_search_window Minimum linear search window in which the best possible scan alignment will be found.

double angular_search_window Minimum angular search window in which the best possible scan alignment will be found.

int32 branch_and_bound_depth Number of precomputed grids to use.

1.13 cartographer.mapping_2d.scan_matching.proto.RealTimeCorrelativeScanI

double linear_search_window Minimum linear search window in which the best possible scan alignment will be found.

double angular_search_window Minimum angular search window in which the best possible scan alignment will be found.

double translation_delta_cost_weight Weights applied to each part of the score.

double rotation_delta_cost_weight Not yet documented.

1.14 cartographer.mapping_3d.proto.LocalTrajectoryBuilderOptions

float min_range Rangefinder points outside these ranges will be dropped.

float max_range Not yet documented.

int32 num_accumulated_range_data Number of range data to accumulate into one unwarped, combined range data to use for scan matching.

float voxel_filter_size Voxel filter that gets applied to the range data immediately after cropping.

cartographer.sensor.proto.AdaptiveVoxelFilterOptions high_resolution_adaptive_voxel_filter_options Voxel filter used to compute a sparser point cloud for matching.

cartographer.sensor.proto.AdaptiveVoxelFilterOptions low_resolution_adaptive_voxel_filter_options Not yet documented.

bool use_online_correlative_scan_matching Whether to solve the online scan matching first using the correlative scan matcher to generate a good starting point for Ceres.

cartographer.mapping_2d.scan_matching.proto.RealTimeCorrelativeScanMatcherOptions real_time_correlative_scan_matche Not yet documented.

cartographer.mapping_3d.scan_matching.proto.CeresScanMatcherOptions ceres_scan_matcher_options Not yet documented.

cartographer.mapping.proto.MotionFilterOptions motion_filter_options Not yet documented.

double imu_gravity_time_constant Time constant in seconds for the orientation moving average based on observed gravity via the IMU. It should be chosen so that the error 1. from acceleration measurements not due to gravity (which gets worse when the constant is reduced) and 2. from integration of angular velocities (which gets worse when the constant is increased) is balanced.

int32 rotational_histogram_size Number of histogram buckets for the rotational scan matcher.

cartographer.mapping_3d.proto.SubmapsOptions submaps_options Not yet documented.

1.15 cartographer.mapping_3d.proto.RangeDataInserterOptions

double hit_probability Probability change for a hit (this will be converted to odds and therefore must be greater than 0.5).

double miss_probability Probability change for a miss (this will be converted to odds and therefore must be less than 0.5).

int32 num_free_space_voxels Up to how many free space voxels are updated for scan matching. 0 disables free space.

1.16 cartographer.mapping_3d.proto.SubmapsOptions

double high_resolution Resolution of the 'high_resolution' map in meters used for local SLAM and loop closure.

double high_resolution_max_range Maximum range to filter the point cloud to before insertion into the 'high_resolution' map.

double low_resolution Resolution of the 'low_resolution' version of the map in meters used for local SLAM only.

int32 num_range_data Number of range data before adding a new submap. Each submap will get twice the number of range data inserted: First for initialization without being matched against, then while being matched.

cartographer.mapping_3d.proto.RangeDataInserterOptions range_data_inserter_options Not yet documented.

1.17 cartographer.mapping_3d.scan_matching.proto.CeresScanMatcherOption

double occupied_space_weight Scaling parameters for each cost functor.

double translation_weight Not yet documented.

double rotation_weight Not yet documented.

bool only_optimize_yaw Whether only to allow changes to yaw, keeping roll/pitch constant.

cartographer.common.proto.CeresSolverOptions ceres_solver_options Configure the Ceres solver. See the Ceres documentation for more information: https://code.google.com/p/ceres-solver/

1.18 cartographer.mapping_3d.scan_matching.proto.FastCorrelativeScanMatch

int32 branch_and_bound_depth Number of precomputed grids to use.

int32 full_resolution_depth Number of full resolution grids to use, additional grids will reduce the resolution by half each.

double min_rotational_score Minimum score for the rotational scan matcher.

double min_low_resolution_score Threshold for the score of the low resolution grid below which a match is not considered. Only used for 3D.

double linear_xy_search_window Linear search window in the plane orthogonal to gravity in which the best possible scan alignment will be found.

double linear_z_search_window Linear search window in the gravity direction in which the best possible scan alignment will be found.

double angular_search_window Minimum angular search window in which the best possible scan alignment will be found.

1.19 cartographer.sensor.proto.AdaptiveVoxelFilterOptions

float max_length 'max_length' of a voxel edge.

float min_num_points If there are more points and not at least 'min_num_points' remain, the voxel length is reduced trying to get this minimum number of points.

float max_range Points further away from the origin are removed.

CHAPTER 2

Evaluation

Performing evaluation is a crucial part of developing a SLAM system. For this purpose, Cartographer offers built-in tools that can aid the tuning process or can be used for quality assurance purposes.

These tools can be used to assess the SLAM result even when no dedicated ground truth is available. This is in contrast to public SLAM benchmarks like e.g the KITTI dataset[1] or the TUM RGB-D dataset[2], where highly-precise ground truth states (GPS-RTK, motion capture) are available as a reference.

2.1 Concept

The process comprises two steps:

1. auto-generation of "ground truth" relations
2. evaluation of the test data against the generated ground truth

The evaluation is based on the pose relations metric proposed in[3]. Rather than comparing the pose of a trajectory node directly to the corresponding ground truth pose, it compares the relative poses between two trajectory nodes in the probe data to the corresponding relation of two trajectory nodes in the ground truth trajectory.

In Cartographer, we can generate such ground truth relations from trajectories with loop closures. Let an optimized trajectory with loop closures be the input for the ground truth generation. We select the relations from loop closure constraints that satisfy the following criteria:

- `min_covered_distance`: Minimum covered distance in meters before a loop closure is considered a candidate for autogenerated ground truth.
- `outlier_threshold_meters`: Distance in meters beyond which constraints are considered outliers.
- `outlier_threshold_radians`: Distance in radians beyond which constraints are considered outliers.

[1] Andreas Geiger, Philip Lenz and Raquel Urtasun. *Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite*. CVPR, 2012.

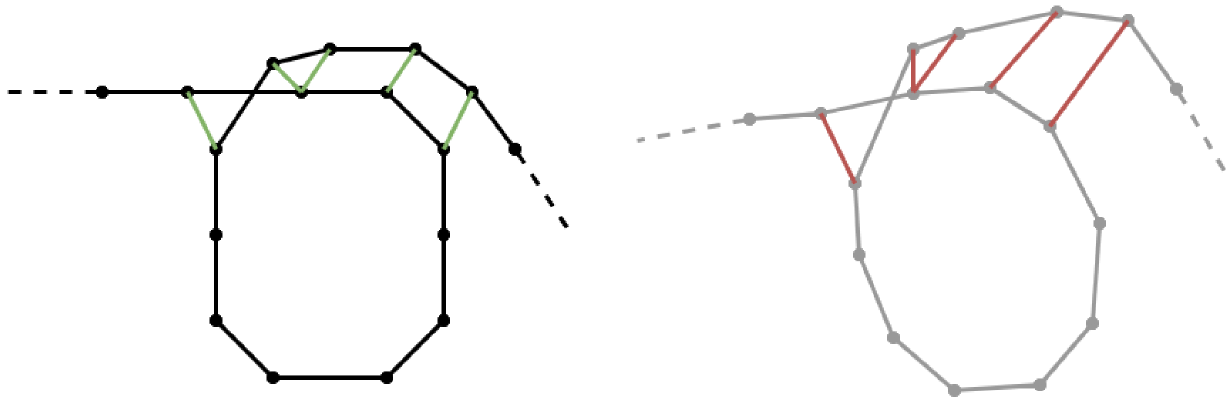
[2] Jürgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard and Daniel Cremers. *A Benchmark for the Evaluation of RGB-D SLAM Systems*. IROS, 2012.

[3] Rainer Kümmerle, Bastian Steder, Christian Dornhege, Michael Ruhnke, Giorgio Grisetti, Cyrill Stachniss and Alexander Kleiner. *On measuring the accuracy of SLAM algorithms*. Autonomous Robots 27(4), pp.387-407, 2009.

We can assume the pose relations of neighboring trajectory nodes fulfilling these requirements to be locally correct in a fully optimized trajectory. Although this is not a ground truth in the sense of an independent input from another source, we can now use it to evaluate the quality of local SLAM results that were generated without loop closure optimization.

The following figure illustrates the concept. On the left side, the ground truth relations are visualized as green connections between trajectory nodes of a fully optimized trajectory. On the right side, the corresponding relations in a non-optimized trajectory are shown in red.

The actual metric that is computed is the difference between the ground truth (green) and the probe (red) relations.



2.2 Advantages & Limitations

The first obvious advantage is the easier data collection process compared to a cumbersome ground truth setup. Another great advantage of this methodology is that the SLAM system can be evaluated in any custom sensor configuration (compared to public benchmarks where we are restricted to the data and the sensor configuration of the authors).

However, this type of self-evaluation is not suitable for measuring the accuracy of the full SLAM system with all optimizations enabled - only an evaluation with *real* ground truth states can provide that. Furthermore, trajectory nodes outside of loop closure areas can't be considered.

2.3 How-To

Given a serialized state of a fully optimized trajectory (here: `optimized.pbstream` file), the ground truth relations can be generated with the following command:

```
cd <build>  # (directory where Cartographer's binaries are located)
./cartographer_autogenerate_ground_truth -pose_graph_filename optimized.pbstream -
↪output_filename relations.pbstream -min_covered_distance 100 -outlier_threshold_
↪meters 0.15 -outlier_threshold_radians 0.02
```

Then, a non-optimized trajectory `test.pbstream` can be evaluated against the generated relations with:

```
./cartographer_compute_relations_metrics -relations_filename relations.pbstream -pose_
↪graph_filename test.pbstream
```

This will produce output in this form:

```
Abs translational error 0.01944 +/- 0.01819 m
Sqr translational error 0.00071 +/- 0.00189 m^2
Abs rotational error 0.11197 +/- 0.12432 deg
Sqr rotational error 0.02799 +/- 0.07604 deg^2
```

2.4 References

CHAPTER 3

Terminology

This documents a few common patterns that exist in the Cartographer codebase.

3.1 Frames

global map frame This is the frame in which global SLAM results are expressed. It is the fixed map frame including all loop closure and optimization results. The transform between this frame and any other frame can jump when new optimization results are available. Its z-axis points upwards, i.e. the gravitational acceleration vector points in the -z direction, i.e. the gravitational component measured by an accelerometer is in the +z direction.

local map frame This is the frame in which local SLAM results are expressed. It is the fixed map frame excluding loop closures and the pose graph optimization. For a given point in time, the transform between this and the global map frame may change, but the transform between this and all other frames does not change.

submap frame Each submap has a separate fixed frame.

tracking frame The frame in which sensor data is expressed. It is not fixed, i.e. it changes over time. It is also different for different trajectories.

gravity-aligned frame Only used in 2D. A frame colocated with the tracking frame but with a different orientation that is approximately aligned with gravity, i.e. the gravitational acceleration vector points approximately in the -z direction. No assumption about yaw (rotation around the z axis between this and the tracking frame) should be made. A different gravity-aligned frame is used for different trajectory nodes, e.g. yaw can change arbitrarily between gravity-aligned frames of consecutive nodes.

3.2 Transforms

local_pose Transforms data from the tracking frame (or a submap frame, depending on context) to the local map frame.

global_pose Transforms data from the tracking frame (or a submap frame, depending on context) to the global map frame.

local_submap_pose Transforms data from a submap frame to the local map frame.

global_submap_pose Transforms data from a submap frame to the global map frame.

CHAPTER 4

Cost functions

4.1 Relative Transform Error 2D

Given two poses $\mathbf{p}_i = [\mathbf{x}_i; \theta_i] = [x_i, y_i, \theta_i]^T$ and $\mathbf{p}_j = [\mathbf{x}_j; \theta_j] = [x_j, y_j, \theta_j]^T$ the transformation $\mathbf{T}$ from the coordinate frame j to the coordinate frame i has the following form

$$\mathbf{T}(\mathbf{p}_i, \mathbf{p}_j) = \begin{bmatrix} R(\theta_i)^T (\mathbf{x}_j - \mathbf{x}_i) \\ \theta_j - \theta_i \end{bmatrix}$$

where $R(\theta_i)^T$ is the rotation matrix of θ_i.

The weighted error $f : \mathbb{R}^6 \mapsto \mathbb{R}^3$ between $\mathbf{T}$ and the measured transformation $\mathbf{T}_{ij}^m = [\mathbf{x}_{ij}^m; \theta_j^m]$ from the coordinate frame j to the coordinate frame i can be computed as

$$\mathbf{f}_{\text{relative}}(\mathbf{p}_i, \mathbf{p}_j) = [w_{\text{t}} \; w_{\text{r}}] \left(\mathbf{T}_{ij}^m - \mathbf{T}(\mathbf{p}_i, \mathbf{p}_j) \right) = \begin{bmatrix} w_{\text{t}} \left(\mathbf{x}_{ij}^m - R(\theta_i)^T (\mathbf{x}_j - \mathbf{x}_i) \right) \\ w_{\text{r}} \left(\text{clamp}(\theta_{ij}^m - (\theta_j - \theta_i)) \right) \end{bmatrix}$$

where w_t and w_r are weights for translation and rotation respectively and $\text{clamp} : \mathbb{R} \mapsto [-\pi, \pi]$ normalizes the angle difference.

Jacobian matrix J_f is given by:

$$J_f(\mathbf{p}_i, \mathbf{p}_j) = \begin{bmatrix} \frac{\partial \mathbf{f}}{\partial x_i} & \frac{\partial \mathbf{f}}{\partial y_i} & \frac{\partial \mathbf{f}}{\partial \theta_i} & \frac{\partial \mathbf{f}}{\partial x_j} & \frac{\partial \mathbf{f}}{\partial y_j} & \frac{\partial \mathbf{f}}{\partial \theta_j} \end{bmatrix} \tag{4.1}$$

(4.2)

$$= \begin{bmatrix} w_{\text{t}} R^T(\theta_i) & -w_{\text{t}} \frac{\mathrm{d}R^T(\theta_i)}{\mathrm{d}\theta} (\mathbf{x}_j - \mathbf{x}_i) & -w_{\text{t}} R^T(\theta_i) & \mathbf{0} \\ \mathbf{0}^T & w_{\text{r}} & \mathbf{0}^T & -w_{\text{r}} \end{bmatrix} \tag{4.3}$$

4.2 Landmark Cost Function

Let $\mathbf{p}_o$ denote the global pose of the SLAM tracking frame at which a landmark with the global pose $\mathbf{p}_l$ is observed. The landmark observation itself is the measured transformation $\mathbf{T}_{ol}^m$ that was observed at time t_o.

As the landmark can be observed asynchronously, the pose of observation $\mathbf{p}_o$ is modeled in between two regular, consecutive trajectory nodes $\mathbf{p}_i, \mathbf{p}_j$. It is interpolated between $\mathbf{p}_i$ and $\mathbf{p}_j$ at the observation time t_o using a linear interpolation for the translation and a quaternion SLERP for the rotation:

$$\mathbf{p}_o = \text{interpolate}(\mathbf{p}_i, \mathbf{p}_j, t_o)$$

Then, the full weighted landmark cost function can be written as:

$$\begin{aligned}\mathbf{f}_{\text{landmark}}(\mathbf{p}_l, \mathbf{p}_i, \mathbf{p}_j) &= \mathbf{f}_{\text{relative}}(\mathbf{p}_l, \mathbf{p}_o) \qquad (4.4)\\ &= [w_{\text{t}}\ w_{\text{r}}]\,(\mathbf{T}^m_{ol} - \mathbf{T}(\mathbf{p}_o, \mathbf{p}_l)) \qquad (4.5)\end{aligned}$$

The translation and rotation weights $w_{\text{t}}, w_{\text{r}}$ are part of the landmark observation data that is fed into Cartographer.

CHAPTER 5

Migration tool for pbstream files

The pbstream serialization format for 3D has changed to include additional data (histograms) in each submap. Code to load old data by migrating on-the-fly will be removed soon. Once this happened, users who wish to migrate old pbstream files can use a migration tool.

The tool is shipped as part of Cartographer's pbstream tool (source) and once built can be invoked as follows::

```
cartographer_pbstream migrate old.pbstream new.pbstream
```

The tool assumes 3D data in the old submap format as input and converts it to the currently used format version.

5.1 Migrating pre-1.0 pbstream files

With the update of the pbstream serialization format as discussed in RFC-0021, previously serialized pbstream files are not loadable in Cartographer 1.0 anymore.

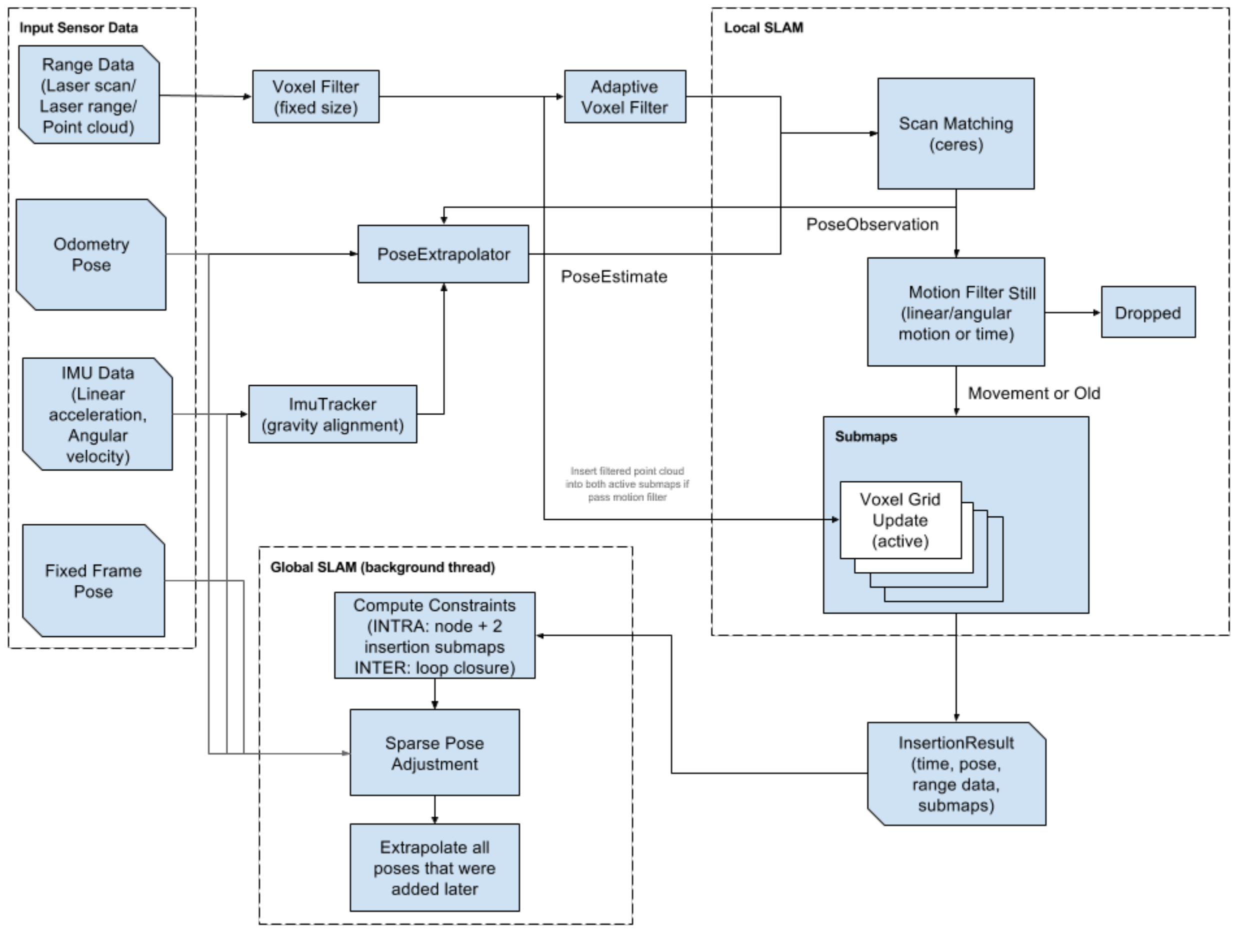
In order to enable users to reuse previously generated pbstream files, migration using an older version of the migration tool is necessary. The current tool does not support this migration anymore. Please use the version at Git SHA 6c889490e245cc5d9da15023249c6fc7119def3f.

Cartographer is a system that provides real-time simultaneous localization and mapping (SLAM) in 2D and 3D across multiple platforms and sensor configurations.

CHAPTER 6

Technical Overview

- High level system overview of Cartographer


Input Sensor Data
Range Data (Laser scan/ Laser range/ Point cloud)
Odometry Pose
IMU Data (Linear acceleration, Angular velocity)
Fixed Frame Pose
Voxel Filter (fixed size)
Adaptive Voxel Filter
PoseExtrapolator
ImuTracker (gravity alignment)
PoseEstimate
Local SLAM
Scan Matching (ceres)
PoseObservation
Motion Filter Still (linear/angular motion or time)
Dropped
Movement or Old
Submaps
Voxel Grid Update (active)
Insert filtered point cloud into both active submaps if pass motion filter
Global SLAM (background thread)
Compute Constraints (INTRA: node + 2 insertion submaps INTER: loop closure)
Sparse Pose Adjustment
Extrapolate all poses that were added later
InsertionResult (time, pose, range data, submaps)

CHAPTER 7

Getting started

Cartographer is a standalone C++ library. To get started quickly, use our ROS integration.

7.1 Getting started with ROS

ROS integration is provided by the Cartographer ROS repository. You will find complete documentation for using Cartographer with ROS at the Cartographer ROS Read the Docs site.

7.2 Getting started without ROS

Please see our ROS integration as a starting point for integrating your system with the standalone library. Currently, it is the best available reference.

On Ubuntu 16.04 (Xenial):

```
# Install the required libraries that are available as debs.
sudo apt-get update
sudo apt-get install -y \
    clang \
    cmake \
    g++ \
    git \
    google-mock \
    libboost-all-dev \
    libcairo2-dev \
    libcurl4-openssl-dev \
    libeigen3-dev \
    libgflags-dev \
    libgoogle-glog-dev \
    liblua5.2-dev \
    libsuitesparse-dev \
```

(continues on next page)

(continued from previous page)

```
    lsb-release \
    ninja-build \
    stow

# Install Ceres Solver and Protocol Buffers support if available.
# No need to build it ourselves.
if [[ "$(lsb_release -sc)" = "focal" || "$(lsb_release -sc)" = "buster" ]]
then
  sudo apt-get install -y python3-sphinx libgmock-dev libceres-dev protobuf-compiler
else
  sudo apt-get install -y python-sphinx
  if [[ "$(lsb_release -sc)" = "bionic" ]]
  then
    sudo apt-get install -y libceres-dev
  fi
fi
```

```
git clone https://github.com/abseil/abseil-cpp.git
cd abseil-cpp
git checkout d902eb869bcfacc1bad14933ed9af4bed006d481
mkdir build
cd build
cmake -G Ninja \
  -DCMAKE_BUILD_TYPE=Release \
  -DCMAKE_POSITION_INDEPENDENT_CODE=ON \
  -DCMAKE_INSTALL_PREFIX=/usr/local/stow/absl \
  ..
ninja
sudo ninja install
cd /usr/local/stow
sudo stow absl
```

```
VERSION="1.13.0"

# Build and install Ceres.
git clone https://ceres-solver.googlesource.com/ceres-solver
cd ceres-solver
git checkout tags/${VERSION}
mkdir build
cd build
cmake .. -G Ninja -DCXX11=ON
ninja
CTEST_OUTPUT_ON_FAILURE=1 ninja test
sudo ninja install
```

```
VERSION="v3.4.1"

# Build and install proto3.
git clone https://github.com/google/protobuf.git
cd protobuf
git checkout tags/${VERSION}
mkdir build
cd build
cmake -G Ninja \
  -DCMAKE_POSITION_INDEPENDENT_CODE=ON \
```

(continues on next page)

(continued from previous page)

```
  -DCMAKE_BUILD_TYPE=Release \
  -Dprotobuf_BUILD_TESTS=OFF \
  ../cmake
ninja
sudo ninja install
```

```
# Build and install Cartographer.
cd cartographer
mkdir build
cd build
cmake .. -G Ninja
ninja
CTEST_OUTPUT_ON_FAILURE=1 ninja test
sudo ninja install
```

CHAPTER 8

System Requirements

Although Cartographer may run on other systems, it is confirmed to be working on systems that meet the following requirements:

- 64-bit, modern CPU (e.g. 3rd generation i7)
- 16 GB RAM
- Ubuntu 16.04 (Xenial), 18.04 (Bionic), 20.04 (Focal)
- gcc version 4.8.4, 5.4.0, 7.5.0, 9.3.0

8.1 Known Issues

- 32-bit builds have libeigen alignment problems which cause crashes and/or memory corruptions.

CHAPTER 9

How to cite us

Background about the algorithms developed for Cartographer can be found in the following publication. If you use Cartographer for your research, we would appreciate it if you cite our paper.

W. Hess, D. Kohler, H. Rapp, and D. Andor, Real-Time Loop Closure in 2D LIDAR SLAM, in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE, 2016. pp. 1271–1278.