3D Mapping with OctoMap

http://octomap.github.io

Armin Hornung

Joint work with K.M. Wurm, M. Bennewitz, C. Stachniss, W. Burgard

Robots in 3D Environments

EU project ROVINA

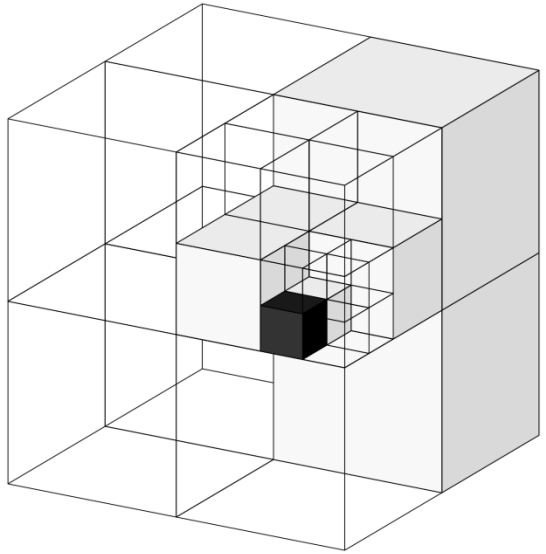
Eric J. Tilford, US Navy

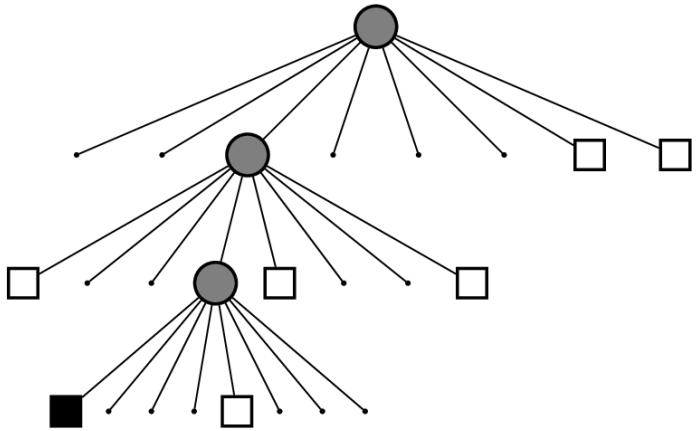
Requirements on a 3D Representation

- Probabilistic representation to
  - Handle sensor noise and dynamic changes
  - Fuse multiple sensors
- Representation of free and unknown areas
  - Collision-free navigation only in free space
  - Exploration of unmapped areas
- Efficiency
  - Compact in memory and on disk
  - Efficient access and queries

Octree

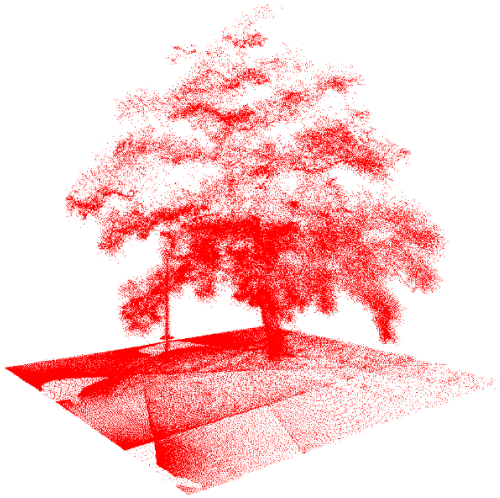
- Tree-based data structure
- Recursive subdivision of space into octants
- Volumes allocated as needed
- Multi-resolution



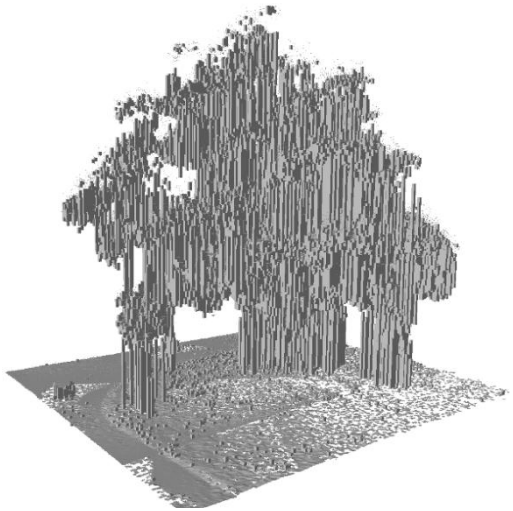


Octrees for 3D Occupancy Maps

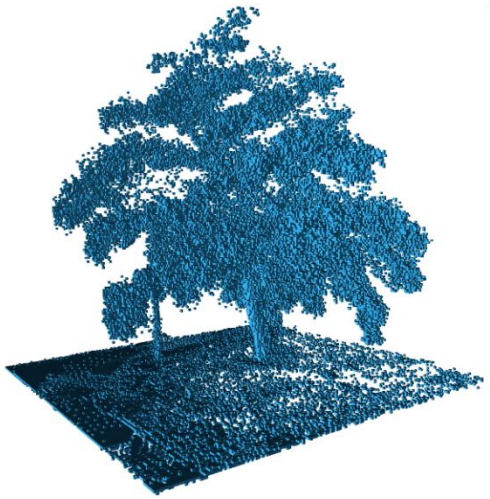
- Store occupancy probability in nodes
- Volumetric 3D model
- Probabilistic integration
- Memory-efficient
- Flexible extension of mapped area



Point cloud



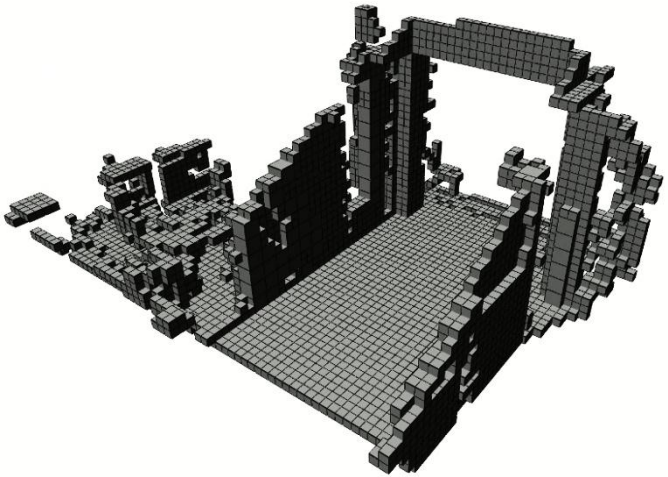
Elevation- / MLS-map

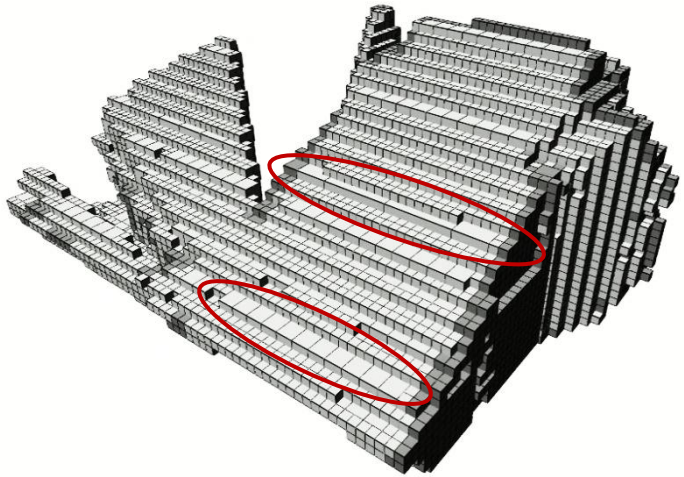


Octree / 3D grid

OctoMap Framework

- Based on octrees
- Probabilistic representation of occupancy including free and unknown areas
- Supports multi-resolution map queries
- Lossless compression
- Compact map files





OctoMap Framework

- Open source (BSD) implementation as C++ library available at octomap.github.io
- Fully documented
- Stand-alone, self-contained library for Linux, Mac, and Windows
- Pre-built Debian packages for ROS *electric* to *hydro*, see www.ros.org/wiki/octomap
- ROS integration in packages octomap_ros, octomap_msgs, and octomap_server
- Collision checks in FCL / MoveIt!

OctoMap Framework

- Details in publication:

  A. Hornung, K.M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard:

  "OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees"

  in *Autonomous Robots* Vol 34, 2013

Auton Robot (2013) 34:189–206
DOI 10.1007/s10514-012-9321-0

OctoMap: an efficient probabilistic 3D mapping framework based on octrees

Armin Hornung · Kai M. Wurm · Maren Bennewitz · Cyrill Stachniss · Wolfram Burgard

Received: 30 April 2012 / Accepted: 31 December 2012 / Published online: 7 February 2013
© Springer Science+Business Media New York 2013

Abstract Three-dimensional models provide a volumetric representation of space which is important for a variety of robotic applications including flying robots and robots that are equipped with manipulators. In this paper, we present an open-source framework to generate volumetric 3D environment models. Our mapping approach is based on octrees and uses probabilistic occupancy estimation. It explicitly represents not only occupied space, but also free and unknown areas. Furthermore, we propose an octree map compression method that keeps the 3D models compact. Our framework is available as an open-source C++ library and has already been successfully applied in several robotics projects. We present a series of experimental results carried out with real robots and on publicly available real-world datasets. The results demonstrate that our approach is able to update the representation efficiently and models the data consistently while keeping the memory requirement at a minimum.

Keywords 3D · Probabilistic · Mapping · Navigation

1 Introduction

Several robotic applications require a 3D model of the environment. These include airborne, underwater, outdoor, or extra-terrestrial missions. However, 3D models are also relevant for many domestic scenarios, for example, for mobile manipulation and also navigation tasks.

Although 3D mapping is an integral component of many robotic systems, there exist few readily available, reliable, and efficient implementations. The lack of such implementations leads to the re-creation of basic software components and, thus, can be seen as a bottleneck in robotics research. We therefore believe that the development of an open-source 3D mapping framework will greatly facilitate the development of robotic systems that require a 3D geometric representation of the environment.

Most robotics applications require a probabilistic representation, modeling of free, occupied, and unmapped areas, and additionally efficiency with respect to runtime and memory usage. We will now discuss these three requirements in detail.

Probabilistic representation: To create 3D maps, mobile robots sense the environment by taking 3D range measurements. Such measurements are afflicted with uncertainty: Typically, the error in the range measurements is in the order of centimeters. But there may also be seemingly random measurements that are caused by reflections or dynamic obstacles. When the task is to create an accurate model of the environment from such noisy measurements, the underlying uncertainty has to be taken into account probabilistically. Multiple uncertain measurements can then be fused into a robust estimate of the true state of the environment. Another important aspect is that probabilistic sensor fusion allows for the integration of data from multiple sensors and of multiple robots.

Modeling of unmapped areas: In autonomous navigation tasks, a robot can plan collision-free paths only for those areas that have been covered by sensor measurements and detected to be free. Unmapped areas, in contrast, need to be avoided and for this reason the map has to represent such areas. Furthermore, the knowledge about unmapped areas is essential during exploration. When maps are created autonomously, the robot has to plan

A. Hornung (✉) · K. M. Wurm · M. Bennewitz · C. Stachniss · W. Burgard
Department of Computer Science, University of Freiburg, Georges-Koehler-Allee, 79110 Freiburg, Germany
e-mail: hornung@informatik.uni-freiburg.de

Springer

- Preprint available on octomap.github.io

Probabilistic Map Update

- Occupancy modeled as recursive binary Bayes filter [Moravec '85]

$$P(n \mid z_{1:t}) = \left[1 + \frac{1 - P(n \mid z_t)}{P(n \mid z_t)} \frac{1 - P(n \mid z_{1:t-1})}{P(n \mid z_{1:t-1})} \frac{P(n)}{1 - P(n)}\right]^{-1}$$

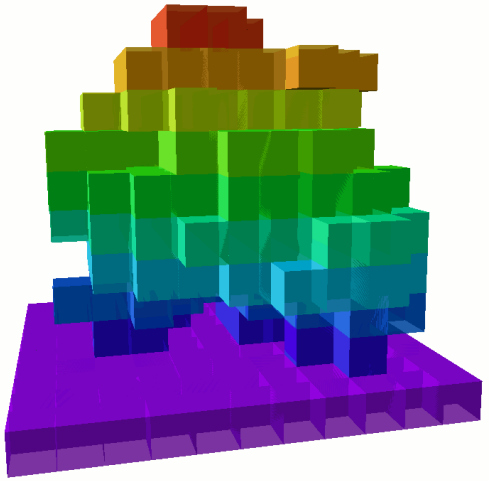
- Efficient update using log-odds

$$L(n \mid z_{1:t}) = L(n \mid z_{1:t-1}) + L(n \mid z_t)$$

Map Update

- Clamping policy ensures updatability [Yguel '07]

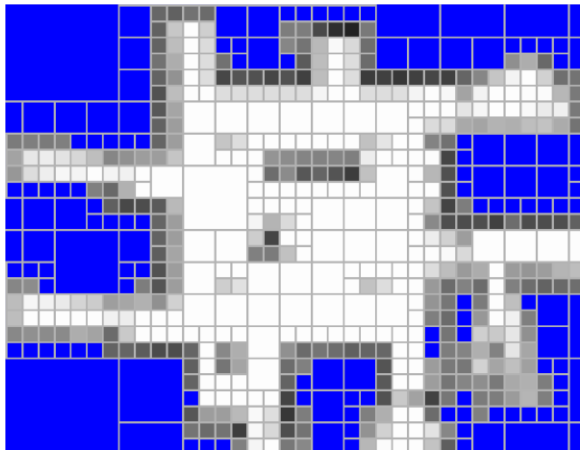
$$L(n) \in [l_{\text{min}}, l_{\text{max}}]$$



- Update of inner nodes enables multi-resolution queries

$$L(n) = \max_{i=1..8} L(n_i)$$

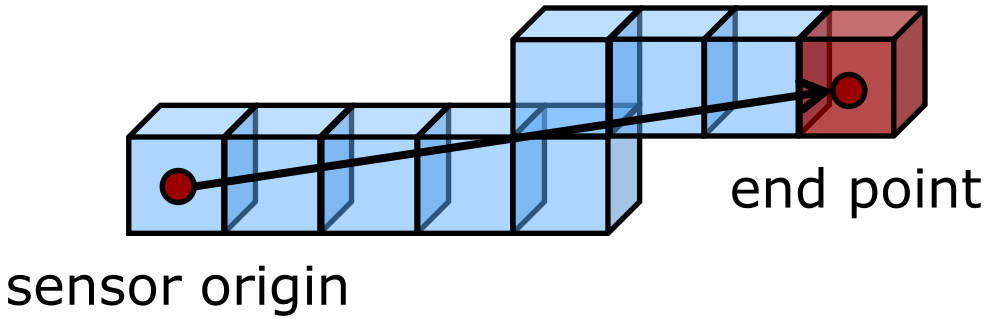
- Compression by pruning a node's identical children



[Kraetzschmar '04]

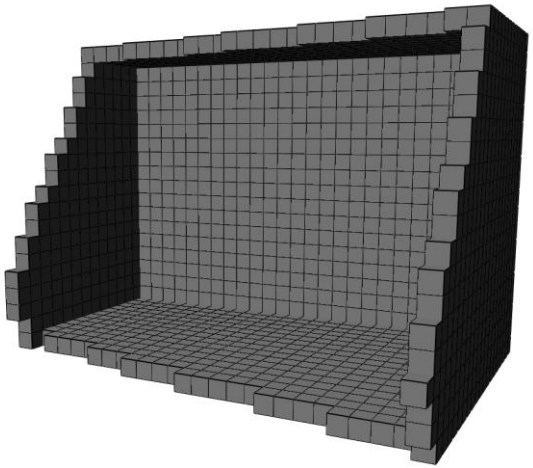
Sensor Model for Single Rays

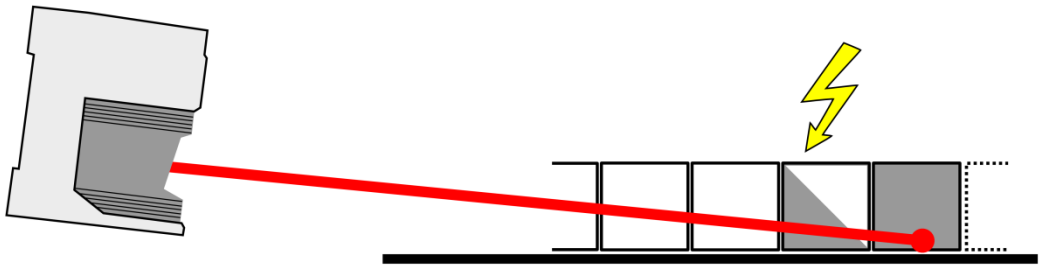
- Ray casting from sensor origin to end point
- Mark last voxel as occupied, all other voxels on ray as free
- Measurements are integrated probabilistically
- Implemented in `OcTree::computeRay(...)` and `OcTree::insertRay(...)`



Sensor Model for 3D Scans

- Sweeping sensor, discretization into voxels
- Planes observed at shallow angle may disappear in a volumetric map
- Solution: Update each voxel of a point cloud at most once, preferring occupied endpoints
- Implemented in `OcTree::insertScan(...)`





Accessing Map Data

- Traverse nodes with iterators

```
for(OcTree::leaf_iterator it = octree.begin_leafs(),
          end=octree.end_leafs(); it!= end; ++it)
{  // access node, e.g.:
  std::cout << "Node center: " << it.getCoordinate();
  std::cout << " value: " << it->getValue() << "\n";
}
```

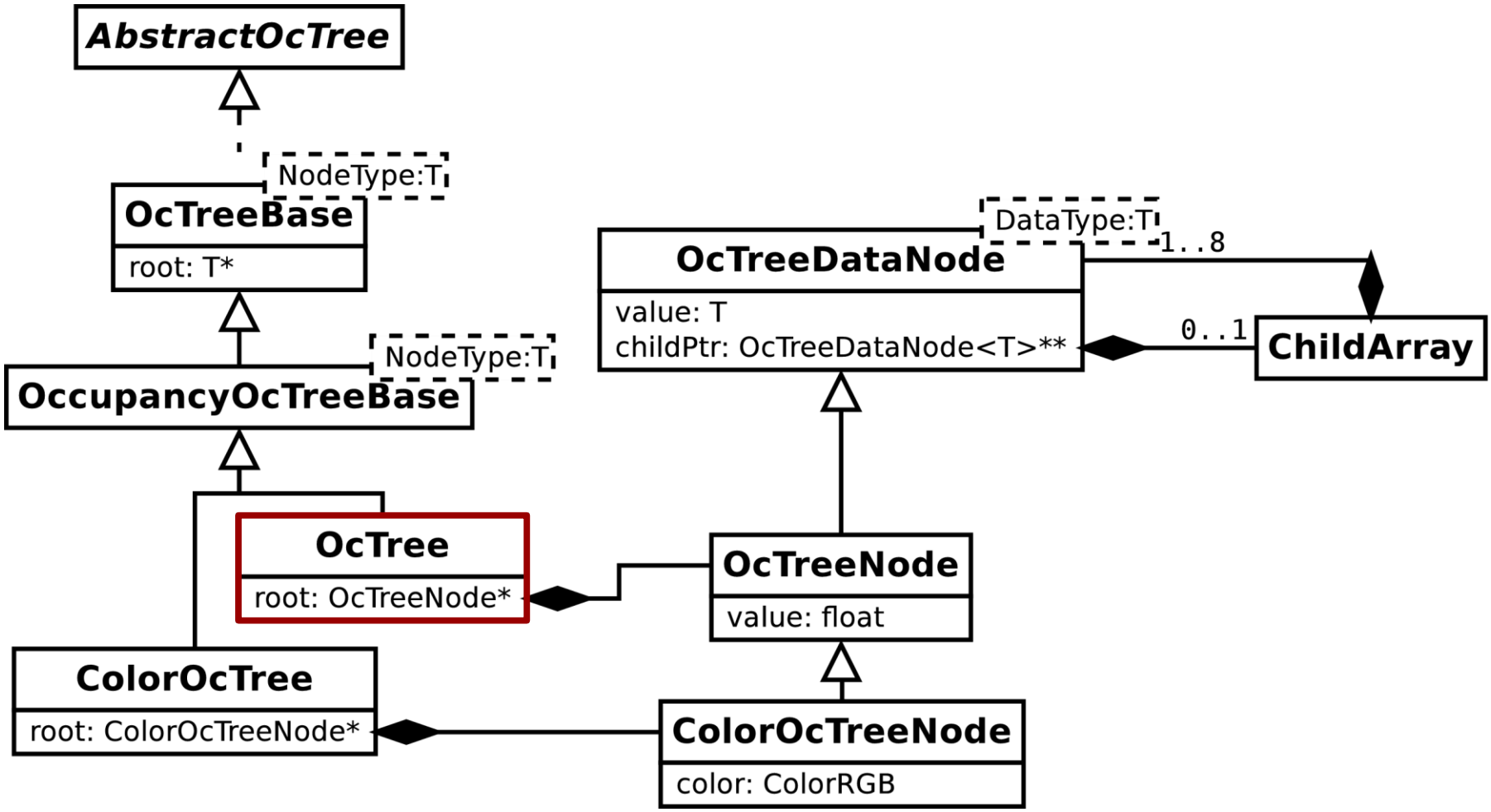
- Ray intersection queries
  - `octree.castRay(...)`
- Access single nodes by searching

```
OcTreeNode* n = octree.search(x,y,z);
if (n){
  std::cout << "Value: " << n->getValue() << "\n";
}
```

Occupancy and Sensor Model

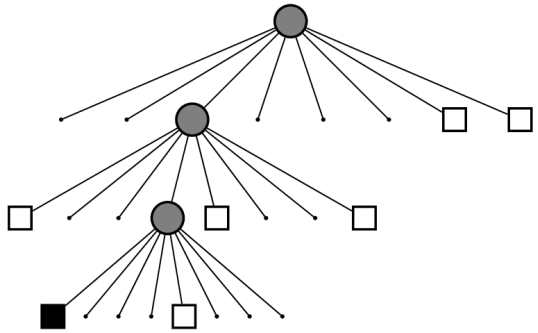
- Set occupancy parameters in octree
  - `octree.setOccupancyThres(0.5);`
  - `octree.setProbHit(0.7); // ...setProbMiss(0.3)`
  - `octree.setClampingThresMin(0.1); / ...Max(0.95)`
- Check if a node is free or occupied
  - `octree.isNodeOccupied(n);`
- Check if a node is “clamped”
  - `octree.isNodeAtThreshold(n);`

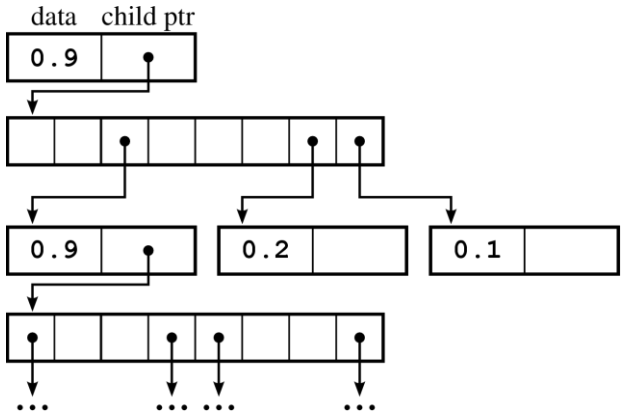
Implementation Details

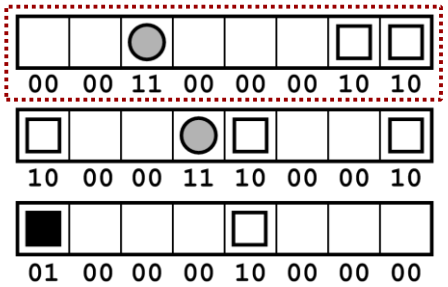


Writing Map Files (Serialization)

- Full probabilities encoded in .ot file format
  - `octree.write(file);`
- Maximum-likelihood map stored as compact bitstream in .bt file
  - Occupied, free, and unknown areas
  - Small file sizes
  - `octree.writeBinary(file);`







2 byte

Reading Map Files (Deserialization)

- Read from .ot file (any kind of octree):

```
AbstractOcTree* tree = AbstractOcTree::read(filename);
if(tree){ // read error returns NULL
  OcTree* ot = dynamic_cast<OcTree*>(tree);
  if (ot){ // cast succeeds if correct type
    // do something....
  }
}
```

- Read from .bt file (OcTree):

```
OcTree* octree = new OcTree(filename);
```

(De-)Serialization in ROS

- octomap_msgs/Octomap.msg contains binary stream and header information
- Use octomap_msgs/conversions.h to convert between octrees and messages
- Serialize:

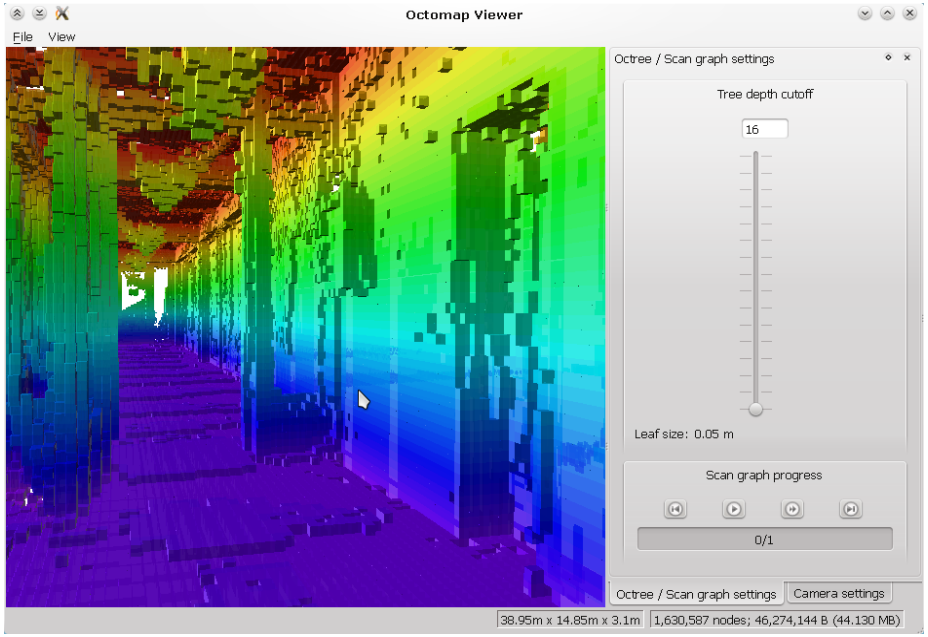
```
octomap_msgs::Octomap map_msg, bmap_msg;
octomap_msgs::fullMapToMsg(octree, map_msg); // (.ot)
octomap_msgs::binaryMapToMsg(octree, bmap_msg); // (.bt)
```

- Deserialize:

```
AbstractOcTree* tree = octomap_msgs::msgToMap(map_msg);
OcTree octree* = dynamic_cast<OcTree*>(tree);
if (octree){ // can be NULL
  ...
}
```

Map Visualization

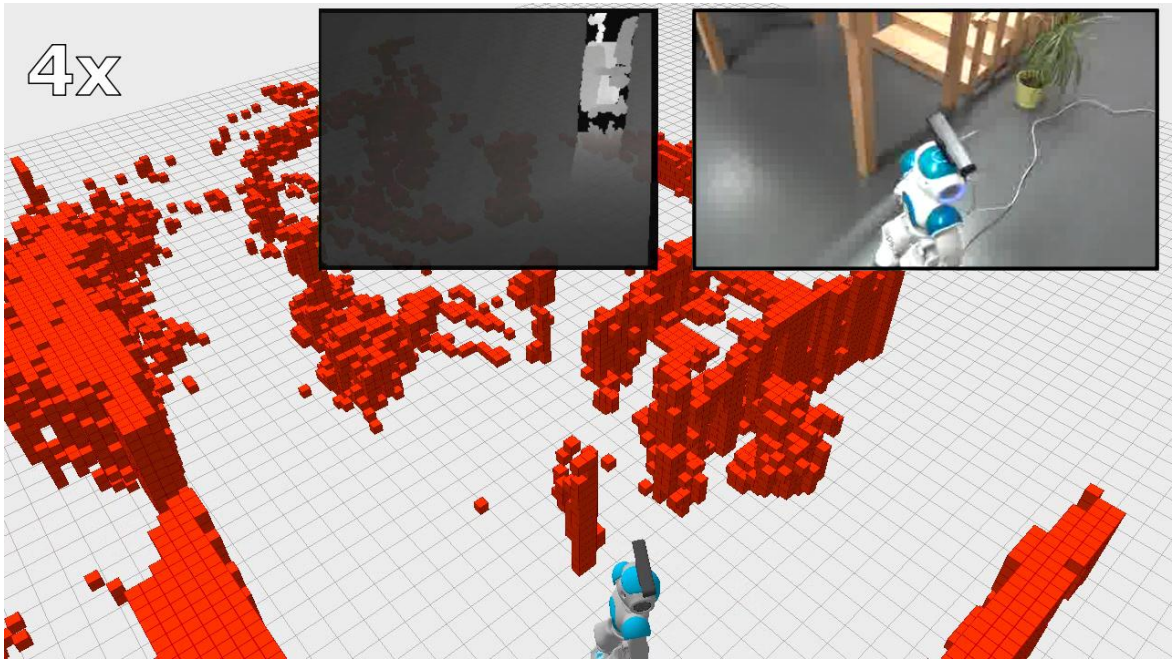
- Native OctoMap visualization: octovis



- RViz:
  - MarkerArray display from octomap_server
  - octomap_rviz_displays
  - MoveIt planning scene

3D Mapping in ROS (Outline)

- Build maps incrementally from point clouds with octomap_server
- Remap topic "cloud_in" to your sensor's PointCloud2
- Requires tf from map frame to sensor frame
- Example launch file in octomap_server



[Maier et al., HUMANOIDS '12]

OctoMap for Navigation

- OctoMap is a mapping framework, expecting registered sensor poses
  - Converts point clouds into 3D occupancy maps
  - **Not** an integrated 3D SLAM solution

- Requires tf from sensor to map frame
  - Example sources: localization, good odometry, rgbdslam, or any other SLAM package

Using OctoMap in Your Project

- Standard CMake (stand-alone or in ROS)
- In CMakeLists.txt:

```
find_package(octomap REQUIRED)
include_directories(${OCTOMAP_INCLUDE_DIRS})
link_libraries(${PROJECT_NAME} ${OCTOMAP_LIBRARIES})
```

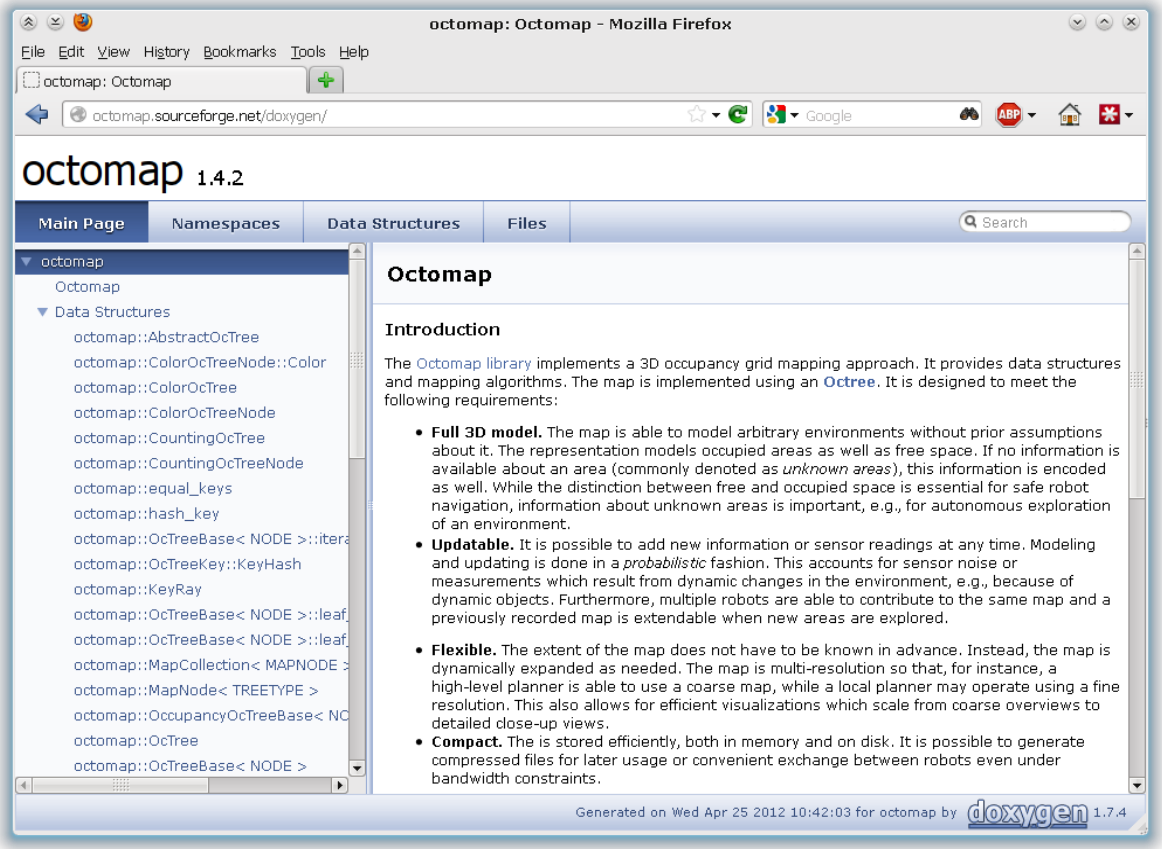
- For ROS:
  - manifest.xml (rosbuild): `<rosdep name="octomap" />`
  - package.xml (catkin):

```
<build_depend>octomap</build_depend>
<run_depend>octomap</run_depend>
```

- Additional ROS packages for integration
  - octomap_msgs: ROS messages & serialization
  - octomap_ros: conversions from native ROS types

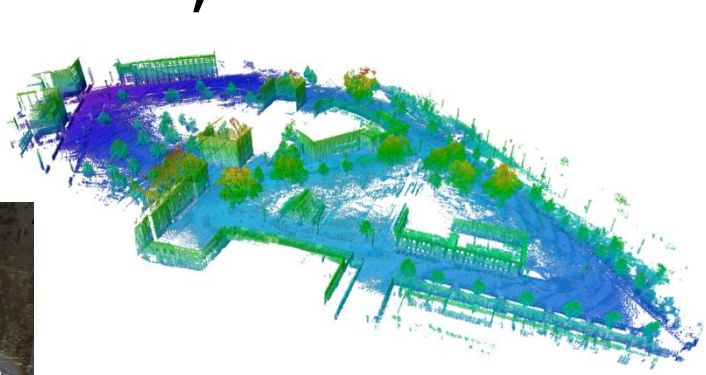
API Documentation

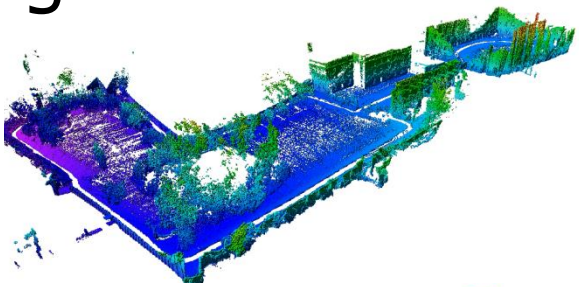
- Latest released version online: http://octomap.github.io/octomap/doc
- Generate from source: “make docs”

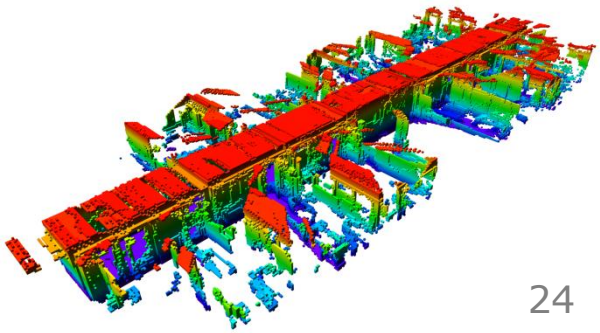


Example Data Sets

- Data set repository at http://ais.informatik.uni-freiburg.de/projects/datasets/octomap/
- Source data (3D laser scans) and final occupancy maps for evaluation
- In- and outoor, small and large scale

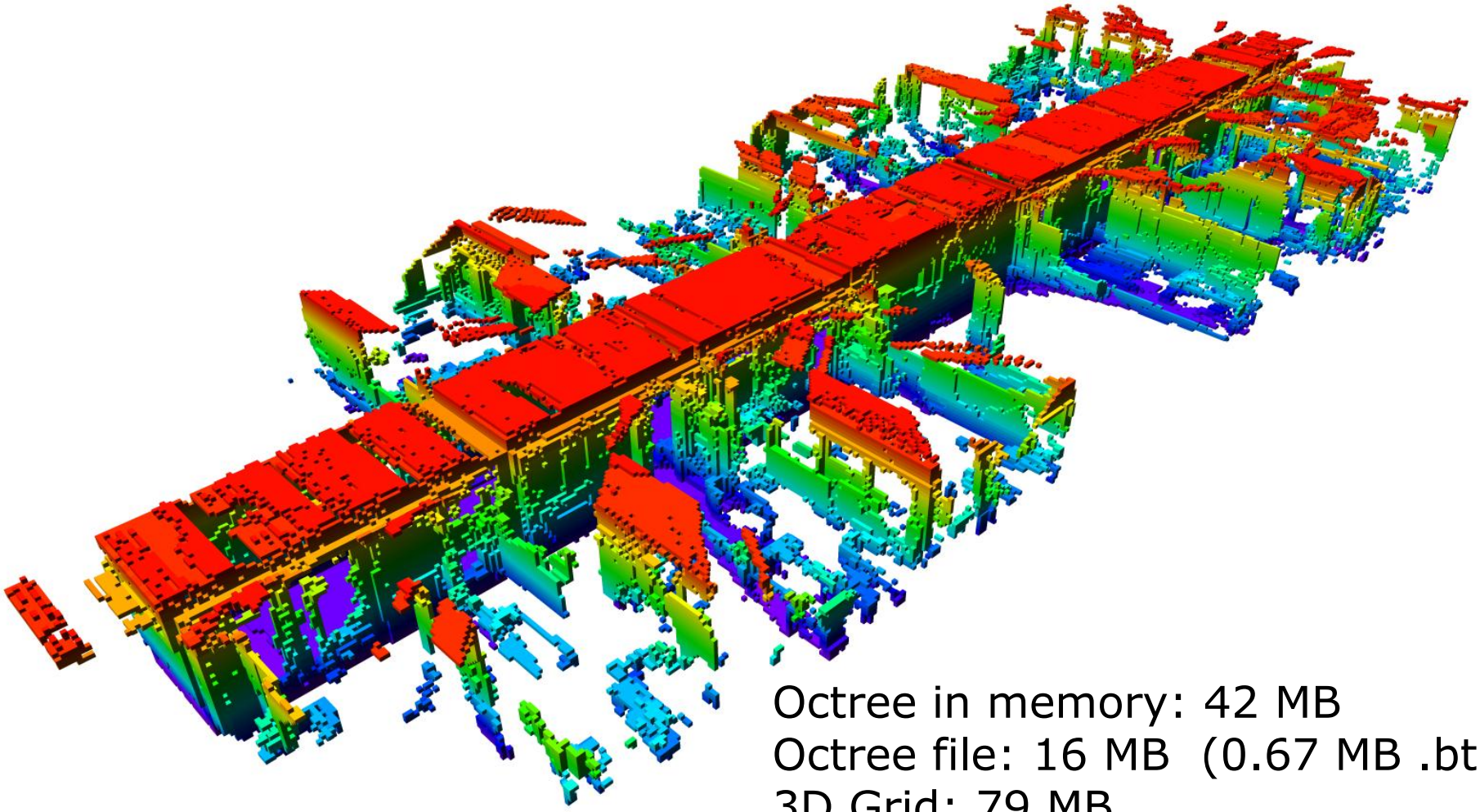






Example: Office Building

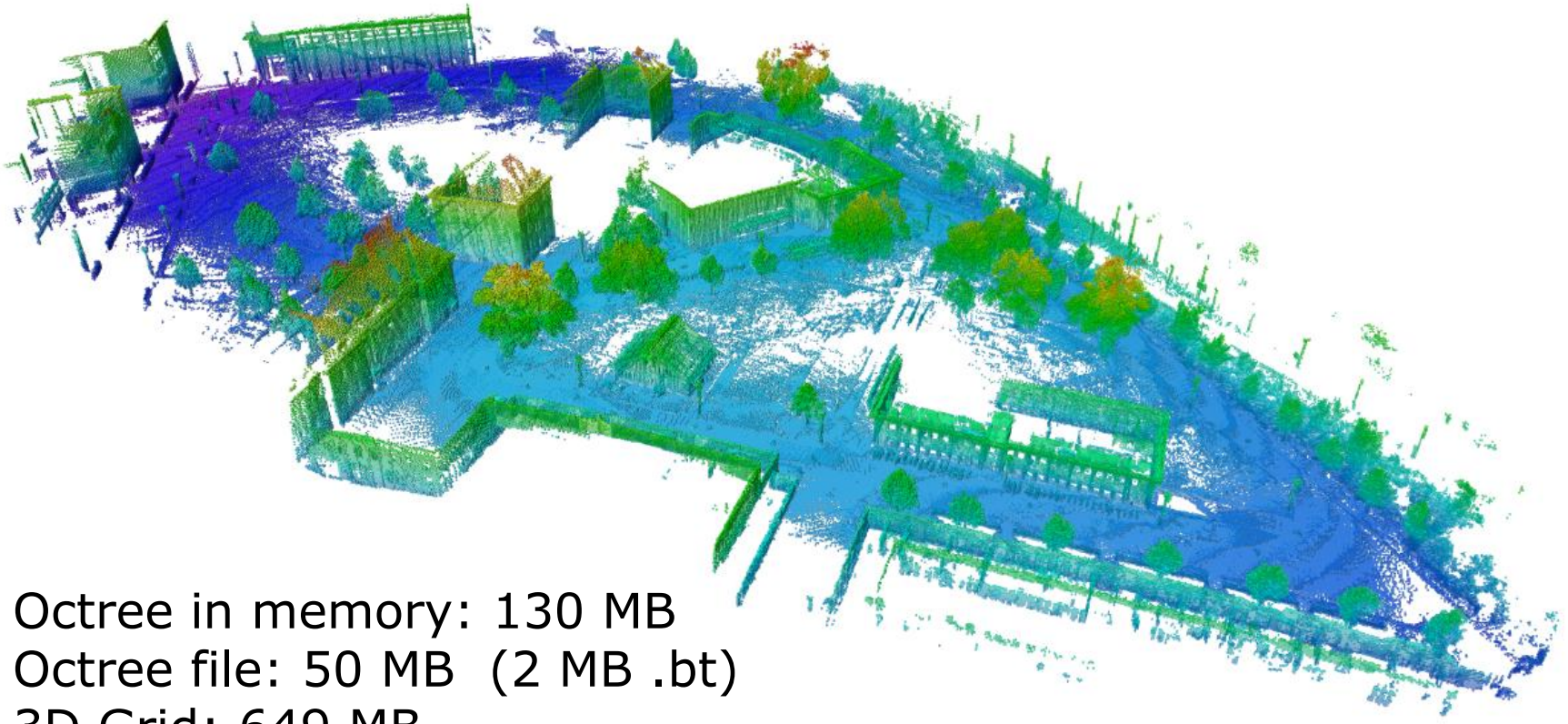
- FR-079 corridor (44 x 18 x 3 m³, 5 cm resolution)



Octree in memory: 42 MB
Octree file: 16 MB (0.67 MB .bt)
3D Grid: 79 MB

Example: Large Outdoor Areas

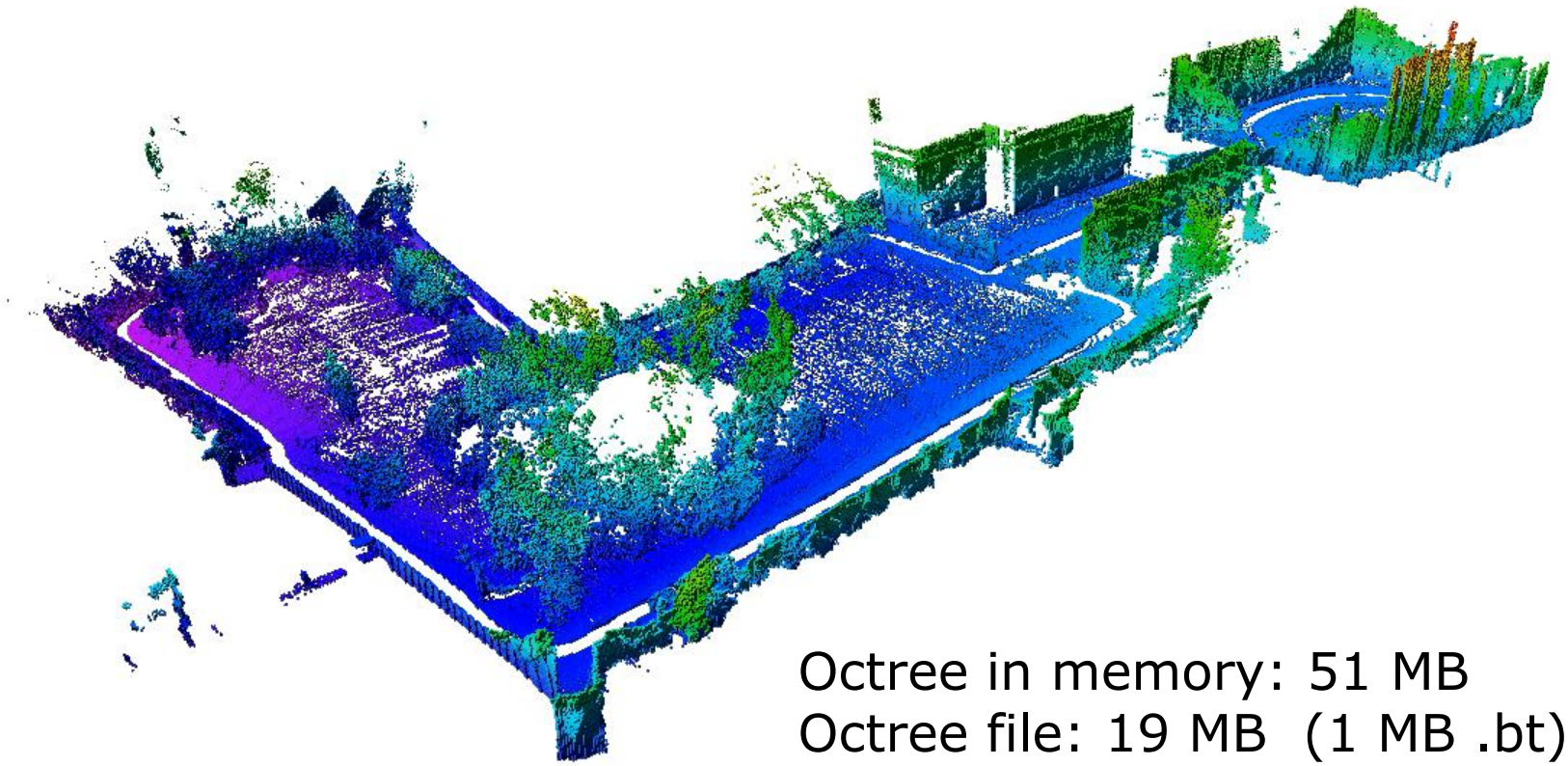
- Freiburg campus (292 x 167 x 28 m^3, 20 cm resolution)



Octree in memory: 130 MB
Octree file: 50 MB (2 MB .bt)
3D Grid: 649 MB

Example: Large Outdoor Areas

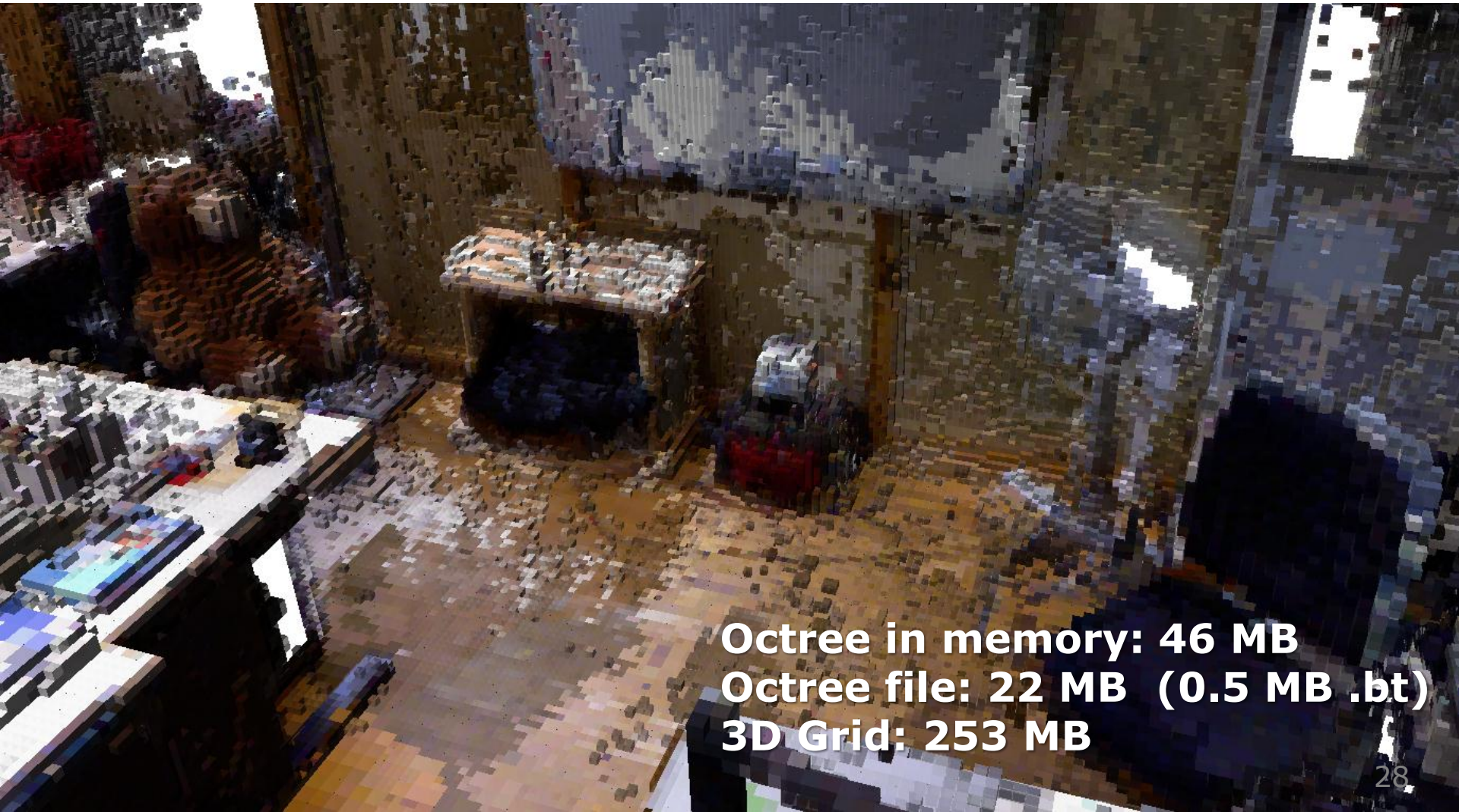
- New College (250 x 161 x 33 m^3, 20 cm resolution)



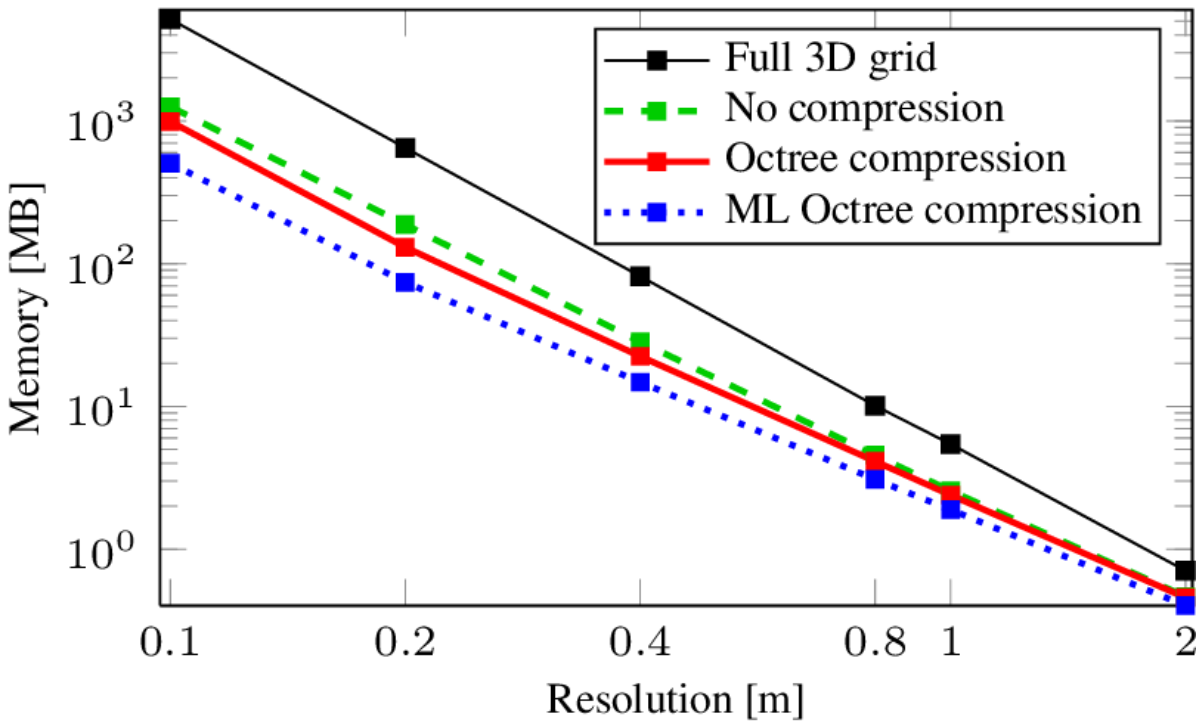
Octree in memory: 51 MB
Octree file: 19 MB (1 MB .bt)
3D Grid: 633 MB

Example: Indoor Environment

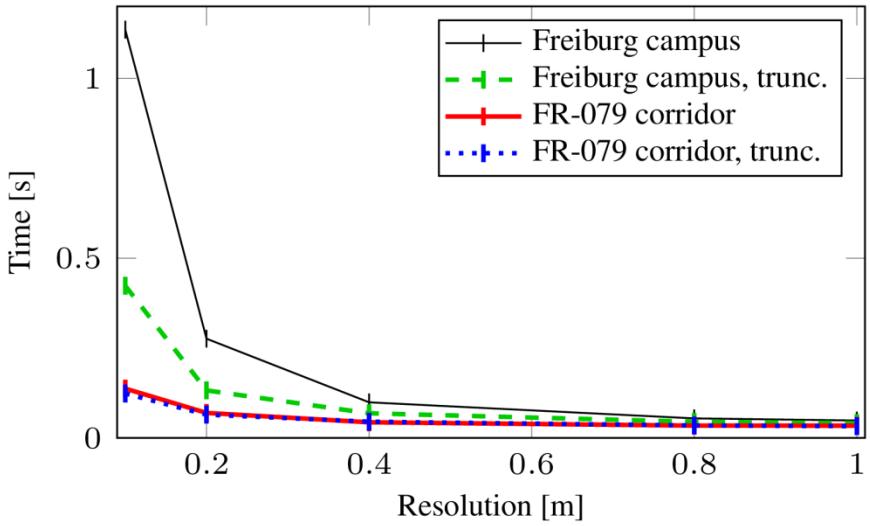
- RGBD freiburg1_360 (8 x 7 x 5 m³, 2 cm resolution)



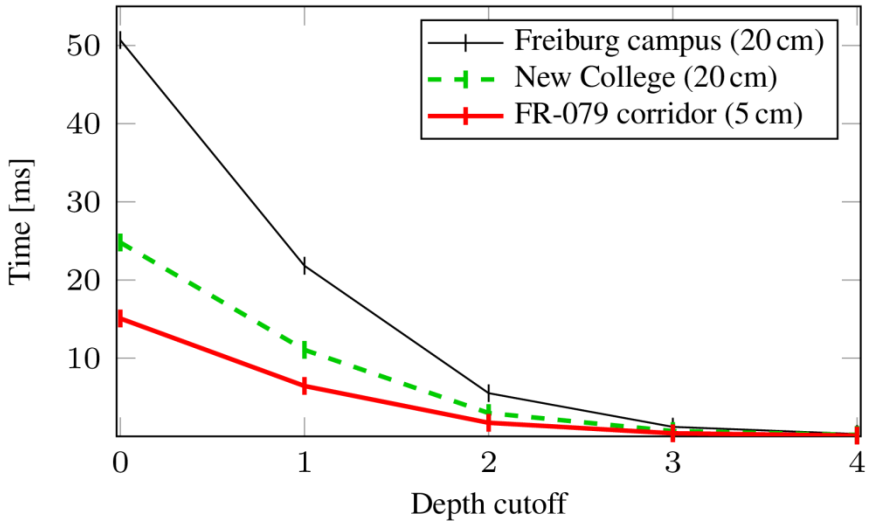
Memory Usage (Freiburg campus)



Update and Query Times



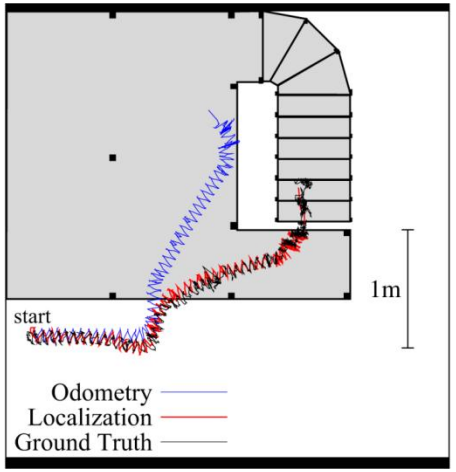
Map update
(Avg. over 100000 points)



Traverse all leaf nodes

Application: Localization

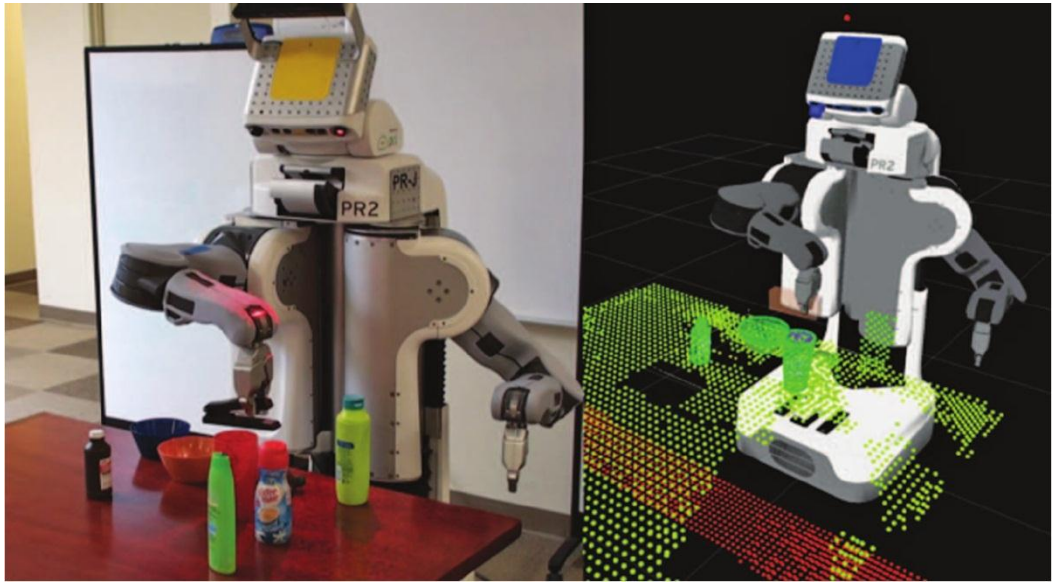
- 6D pose of a humanoid robot estimated in OctoMap
- Monte Carlo localization based on laser, IMU, and joint angle data
- Sensor model: ray casting in OctoMap



[Hornung et al., IROS '10]

Application: Tabletop Manipulation

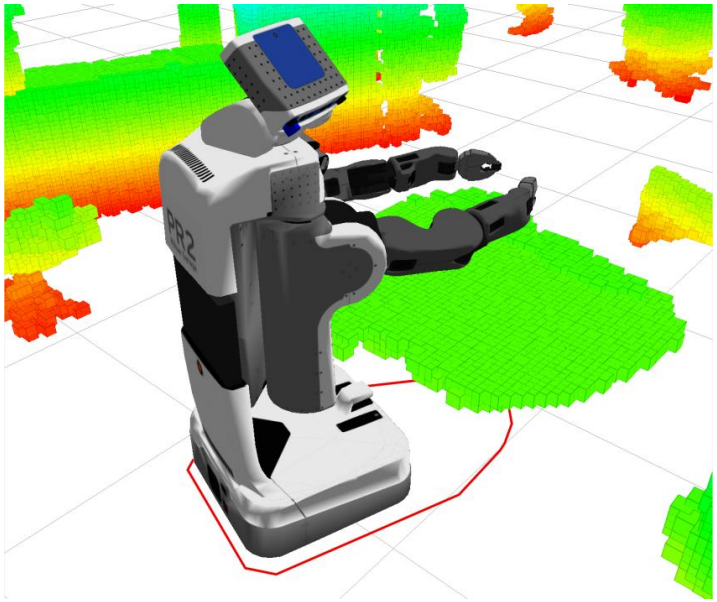
- collider package in ROS *fuerte*
- Directly integrated in MoveIt!
- OctoMap as probabilistic collision map
- Updates map from stereo and laser data
- Enables dynamic updates of the collision map



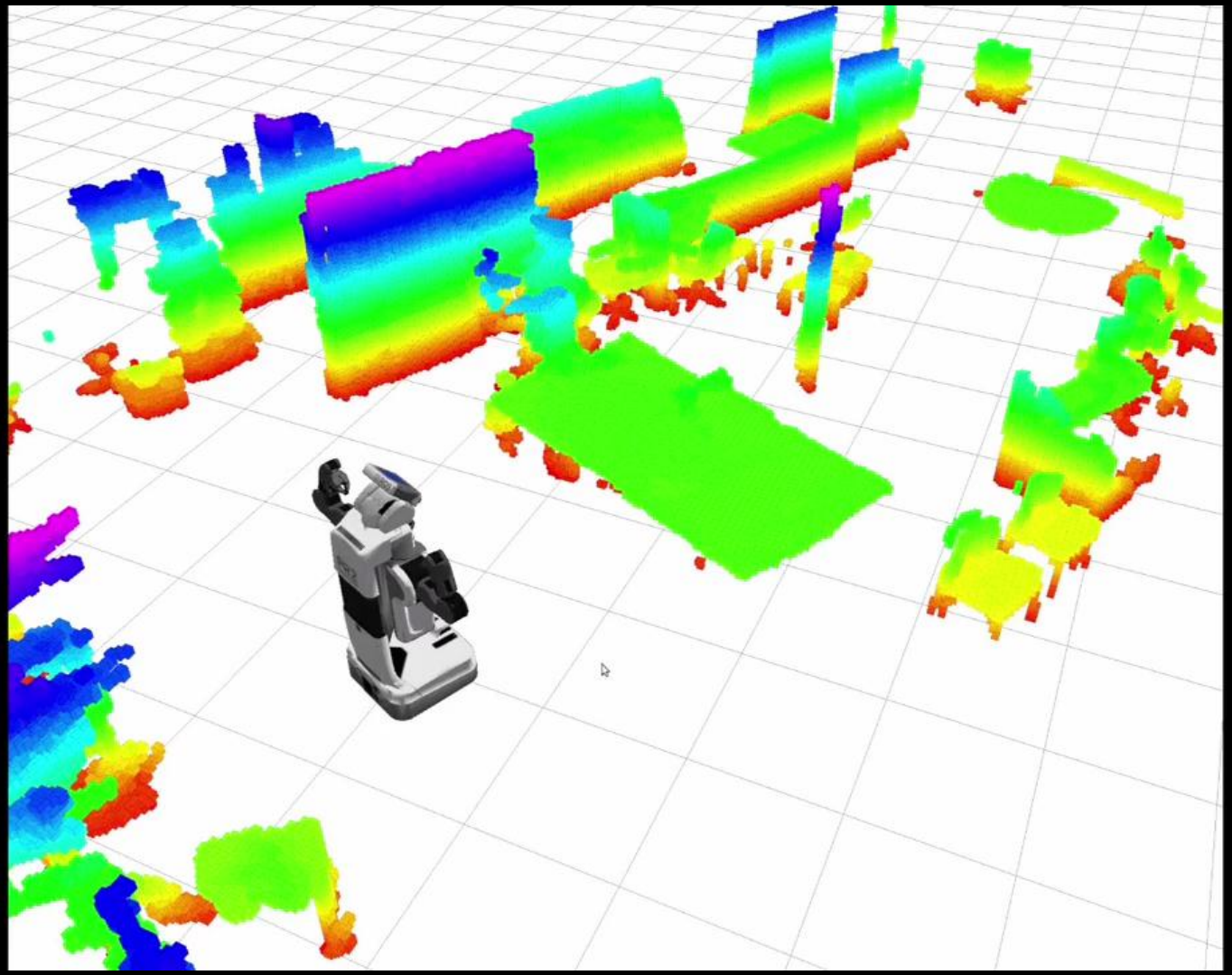
[Chitta et al., Robotics & Automation '12]

Application: Navigation in Clutter

- Collision map and obstacle avoidance for mobile manipulation
- Enables moving through narrow passages and docking tables
- Mapping in octomap_server
- Search-based planning with motion primitives and 2D / 3D collision checks in 3d_navigation stack



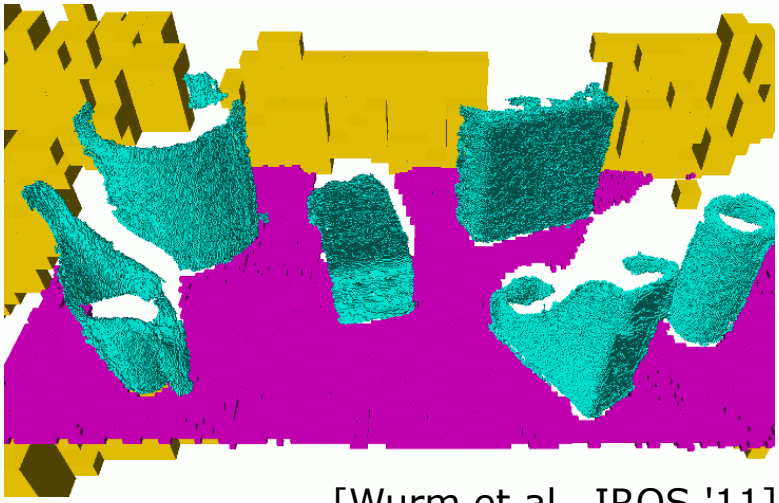
[Hornung et al., ICRA '12]



Extensions

Octree Hierarchies

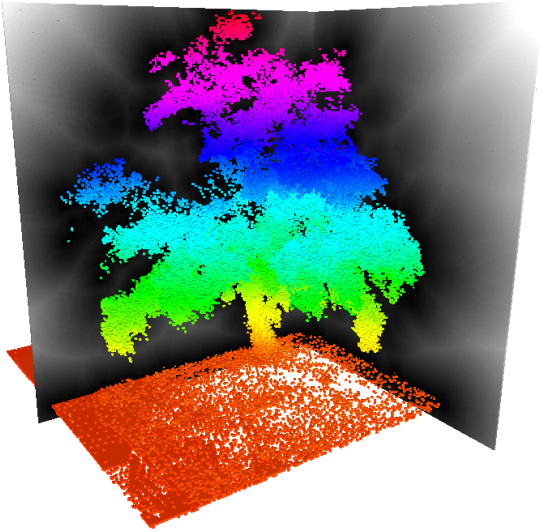
- Local submaps with different resolution and origin



[Wurm et al., IROS '11]

3D Distance Maps

- Incremental updates based on change detection on OctoMap
- Available in OctoMap: dynamicEDT3D



[Lau et al., Robotics and Autonomous Systems '12]

Summary

- Memory-efficient map data structure based on Octrees
- Volumetric representation of occupied, free, and unknown space
- Implementation of common map functionality: sensor updates, raycasting
- Open source code with integration into ROS and MoveIt!
- Can be used for localization, obstacle avoidance, manipulation, ...

Thanks for your attention!

octomap.github.io: Fork & contribute new features, report issues, discuss on the mailing list