TurtleBot2 & ROS Workshop: 3D Environment Mapping

Michal Španěl, Vít Štancl, Rostislav Hulík

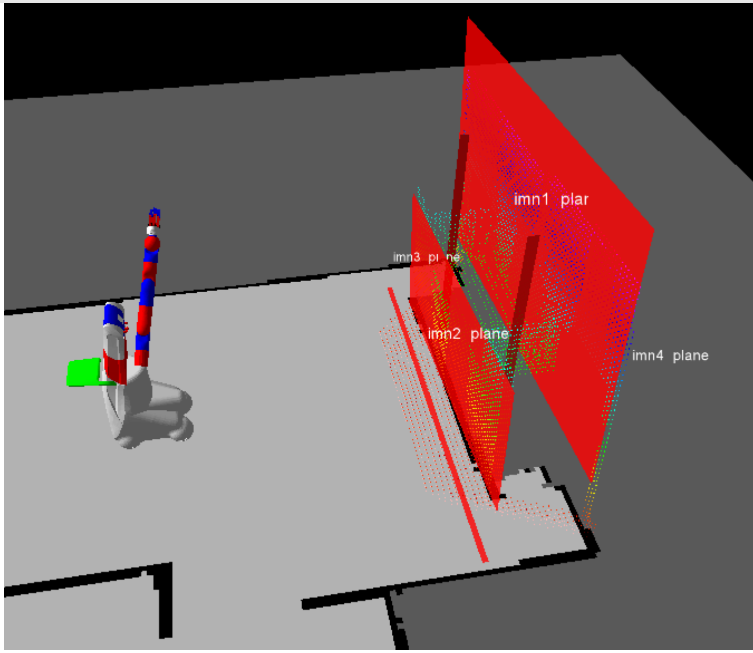
Faculty of Information Technology, Brno University of Technology
Bozetechova 2, 612 66 Brno
spanel@fit.vutbr.cz

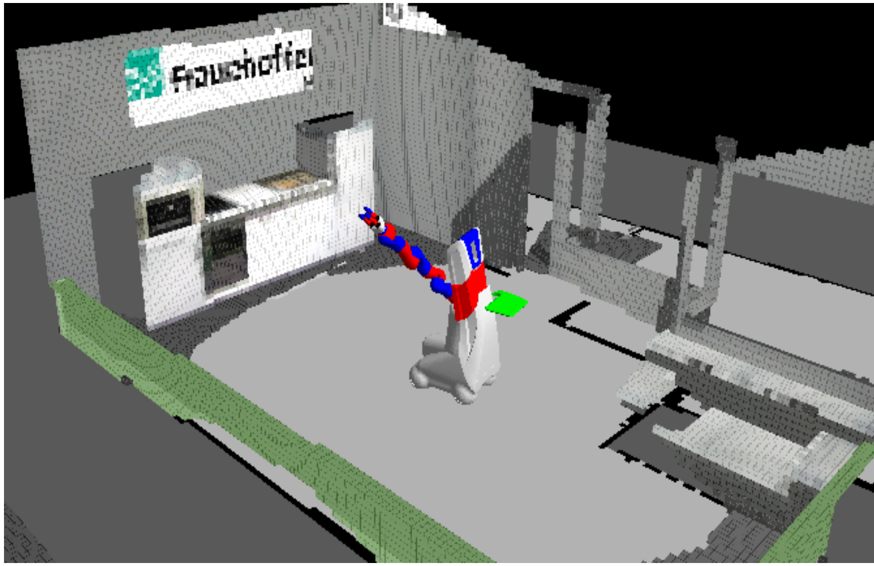
2.10.2012

Outline

- Point cloud processing and geometric mapping
  - Intro to the PCL library
  - Plane detection using PCL
  - (SRS) geometric mapping

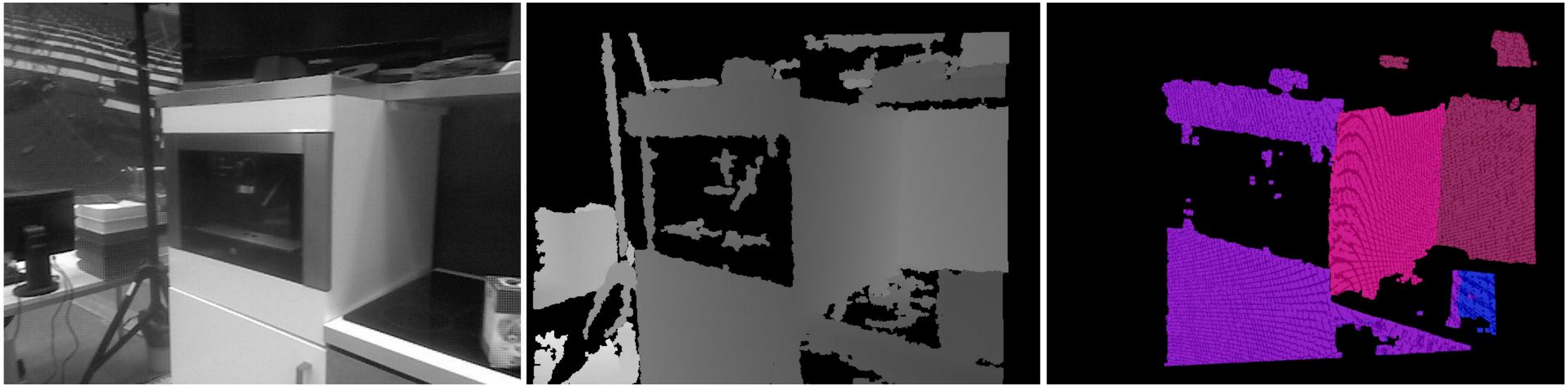
- Global voxel-based map of the environment
  - Intro to the Octomap library
  - Example node
  - (SRS) env. model





Point Cloud Processing

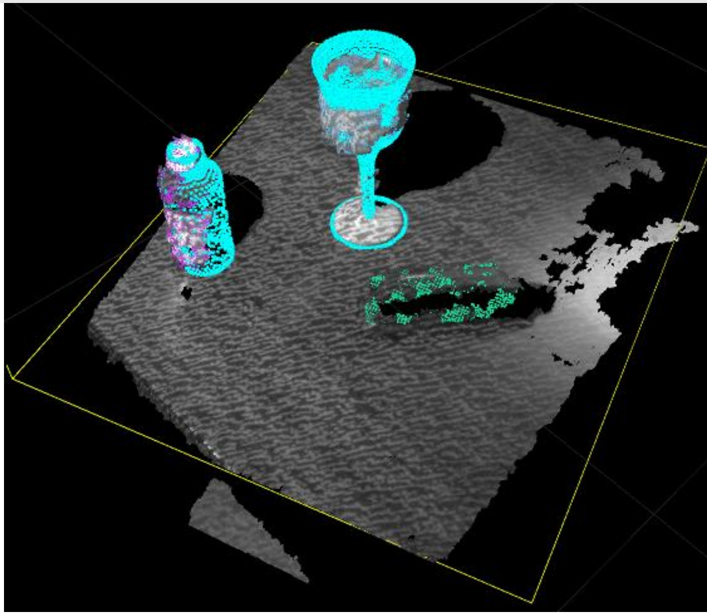
- Kinect device – depth map
- Using camera calibration matrix, the point cloud is computed

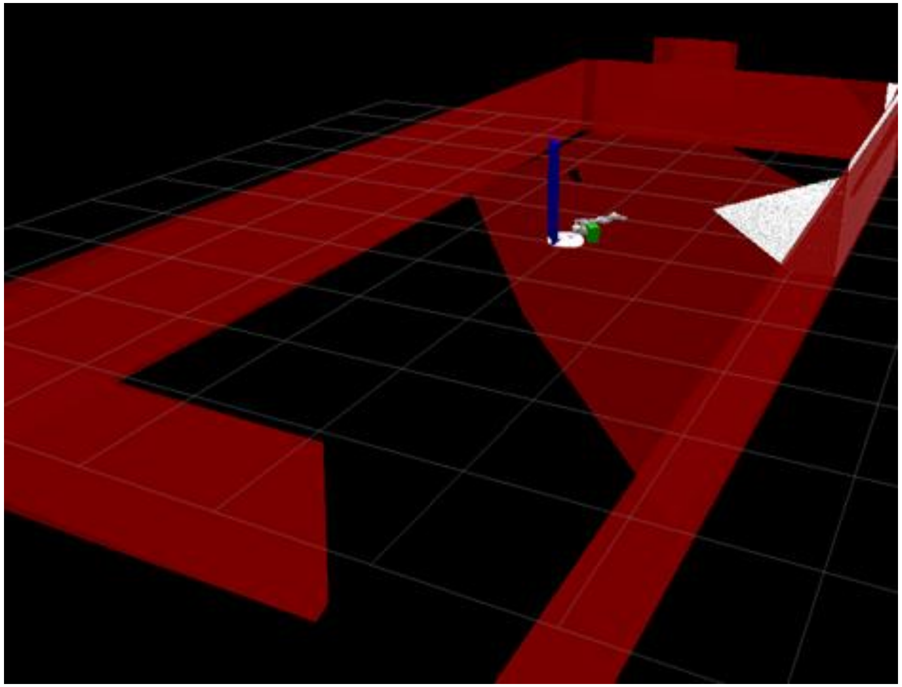


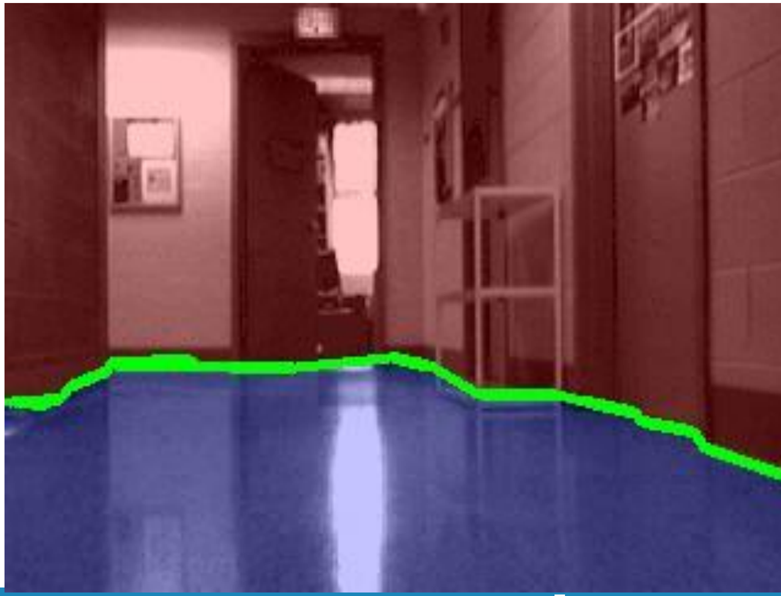
- Our task – extract information from this point cloud
  - geometry
  - shapes (planes)

Why the Plane Detection?

- Table-top detection,
- Dominant plane detection,
- Environment mapping (geometric map)







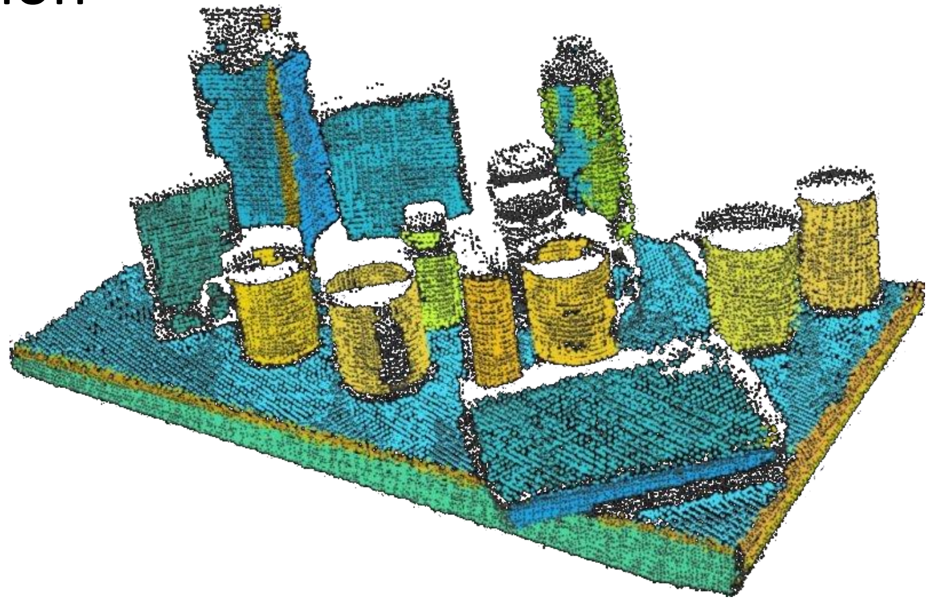
Introduction to the PCL (Point Cloud Library)

- http://pointclouds.org/
- Standalone open project for n-d point clouds and 3D geometry processing
- BSD license
- Supported by **Willow garage**, nVidia, Google etc.

- **Support in ROS**
  - http://www.ros.org/wiki/pcl

PCL – Point Cloud Processing

- Filtering (noise)
- Feature estimation (normals, curvatures etc)
- Keypoint detection
- Registration
- KD-Tree
- Model fitting and segmentation
- Sample consensus methods
- Surface reconstruction



PCL in ROS

- http://www.ros.org/wiki/pcl/Overview

- Point cloud definition:

```
pcl::PointCloud<pcl::PointXYZ> pointcloud;
pointcloud.push_back(…);
pointcloud.points[];
```

- Point cloud message

```
ros::Subscriber point_cloud = n.subscribe(“/cam3d/depth/points“,
QUEUE_SIZE, &callback);

void callback(const sensor_msgs::PointCloud2ConstPtr& cloud)
{
}
```

PCL Sample Node

- Subscribes TB2 point cloud
- Executes PCL‘s RANSAC for plane detection

```
pcl::ModelCoefficients::Ptr coefficients (new
pcl::ModelCoefficients); pcl::PointIndices::Ptr inliers (new
pcl::PointIndices); pcl::SACSegmentation<pcl::PointXYZ> seg;
seg.setOptimizeCoefficients (true);
seg.setModelType (pcl::SACMODEL_PLANE);
seg.setMethodType (pcl::SAC_RANSAC);
seg.setDistanceThreshold (0.01);
seg.setInputCloud (cloud);
seg.segment (*inliers, *coefficients);
```

- Advertises MarkerArray of found planes for visualization

PCL Sample Node, contd.

- Compilation

```
rosmake tb2_3d_env_mapping
```

- Run with TB2 simulation

```
roslaunch tb2_3d_env_mapping planedet_example_test_with_sim_tb2.launch
```

- Run keyboard teleop

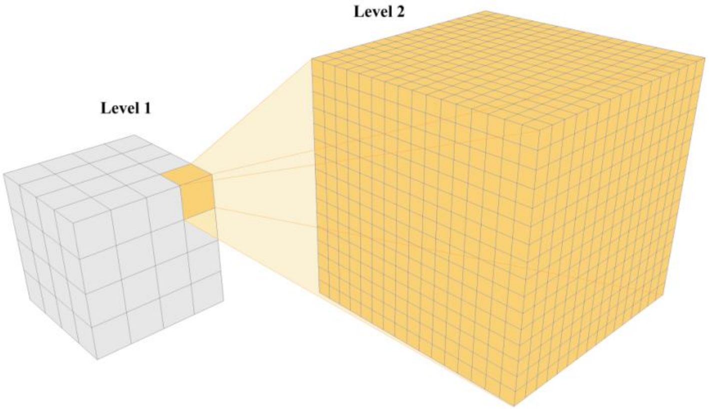
```
rosrun btb_teleop keyboard_teleop.launch
```

PCL Sample Node, contd.

- Let’s see what’s behind…

(SRS) Geometric Mapping

- Continuous Plane Detection based on 3D Hough Transform
  - Our approach – hierarchical hough space representation
  - Saves 95% - 97,5% of memory requirements in each test

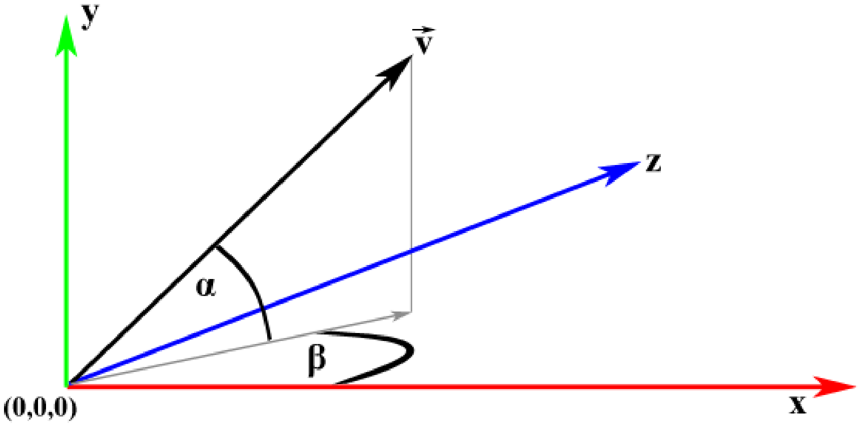


- Multiple memory/speed optimizations
  - Randomized HT accumulation
  - Hierarchical structure
  - Cache space

(SRS) Plane Detection based on 3D HT

- Randomized normal estimation
  - PCL‘s fast integral image normal computation

```
pcl::IntegralImageNormalEstimation<pcl::PointXYZ, pcl::Normal> ne;
ne.setNormalEstimationMethod (ne.AVERAGE_3D_GRADIENT);
ne.setMaxDepthChangeFactor(0.02f);
ne.setNormalSmoothingSize(10.0f
ne.setInputCloud(cloud);
ne.compute(*normals);
```

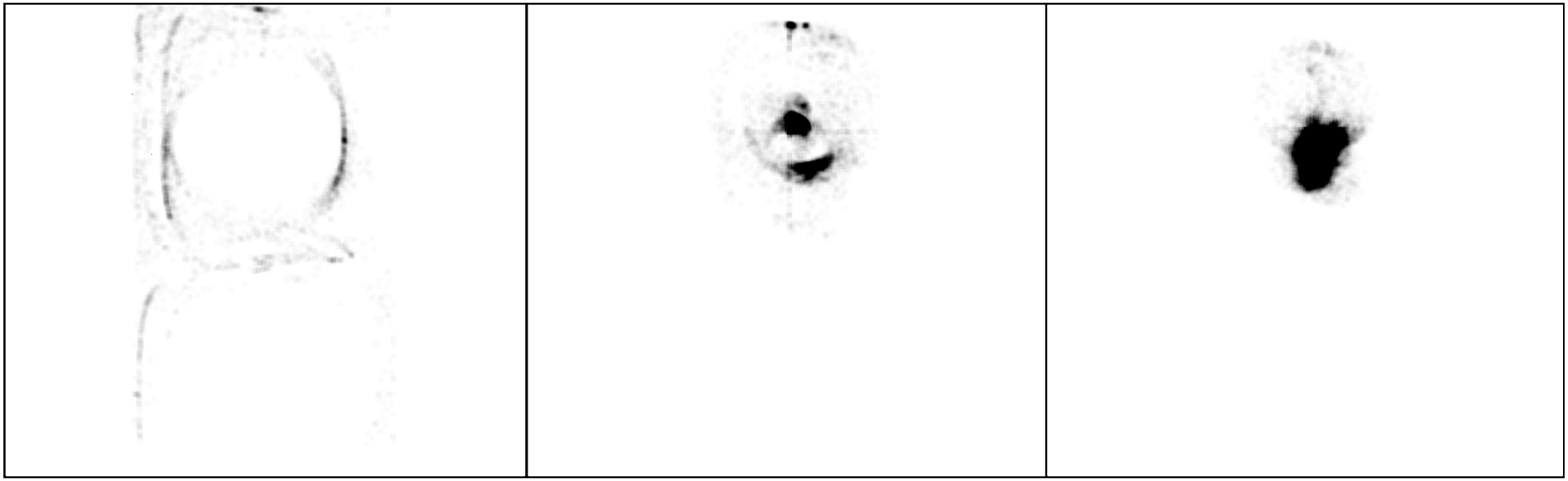


- HS accumulation

$$HS = (\alpha, \beta, d) = [-\pi; \pi]^1 \times [-\pi; \pi]^1 \times \mathbb{R}^1$$

(SRS) Plane Detection based on 3D HT, contd.

- Noise vs. detectable plane vs. clear maximum



(SRS) Sample Plane Detection Node

- Able to add info from multiple frames
- Must have transform to world coordinates

```
tfListener = new tf::TransformListener();

message_filters::Subscriber<sensor_msgs::PointCloud2> points(n, TOPIC, 1);

transform_filter = new tf::MessageFilter<sensor_msgs::PointCloud2> (points,
*tfListener, TARGET_FRAME, 1);

transform_filter->registerCallback(boost::bind(&callback, _1));
```

(SRS) Sample Plane Detection Node

- Transformation into world coordinates

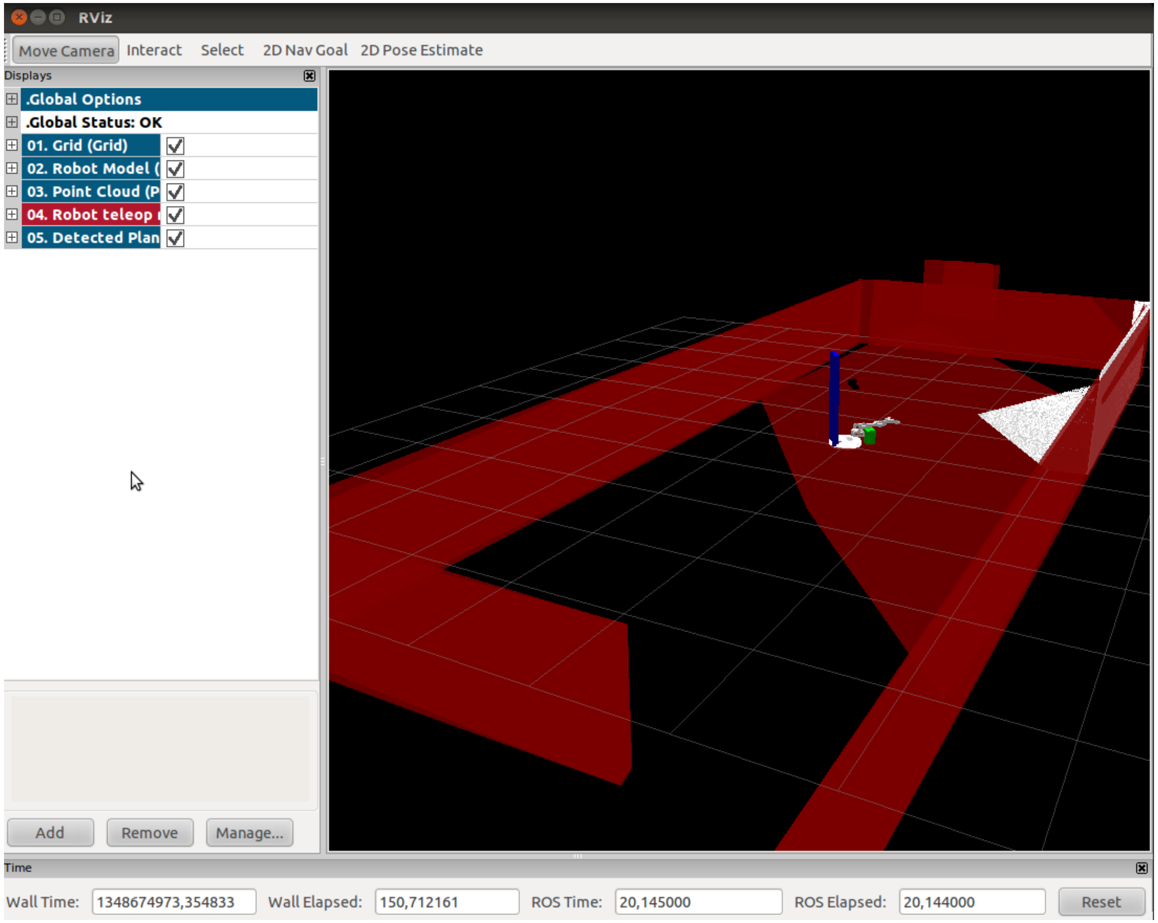
```
tf::StampedTransform sensorToWorldTf;
try {
  tfListener->waitForTransform( settings.param_output_frame,
      cloud->header.frame_id, cloud->header.stamp,
      ros::Duration(2.0));
  tfListener->lookupTransform( settings.param_output_frame,
      cloud->header.frame_id, cloud->header.stamp,
      sensorToWorldTf);
}
catch( tf::TransformException& ex ) { … }

Eigen::Matrix4f sensorToWorld;
pcl_ros::transformAsMatrix(sensorToWorldTf, sensorToWorld);
pcl::transformPointCloud(pointcloud, pointcloud, sensorToWorld);
```

(SRS) Sample Plane Detection Node

- Run node with TB simulation in WG model:

```
rosmake srs_env_model_percp        …UPGM people only!
roslaunch srs_env_model_percp planedet_demo_with_sim_wg.launch
```



Wall Time: 1348674973,354833 Wall Elapsed: 150,712161 ROS Time: 20,145000 ROS Elapsed: 20,144000

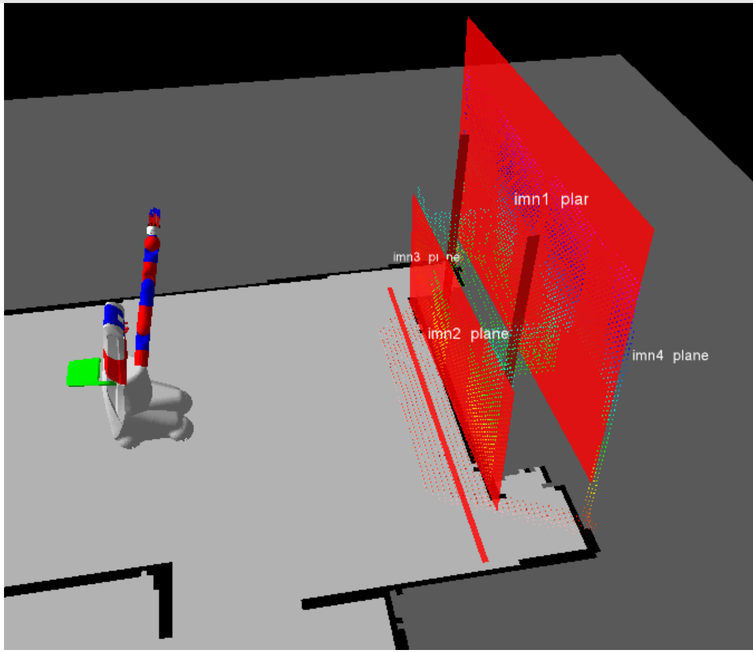
(SRS) Sample Plane Detection Node

- Parameters
  srs_env_model_percp/config/planedet_params.yaml

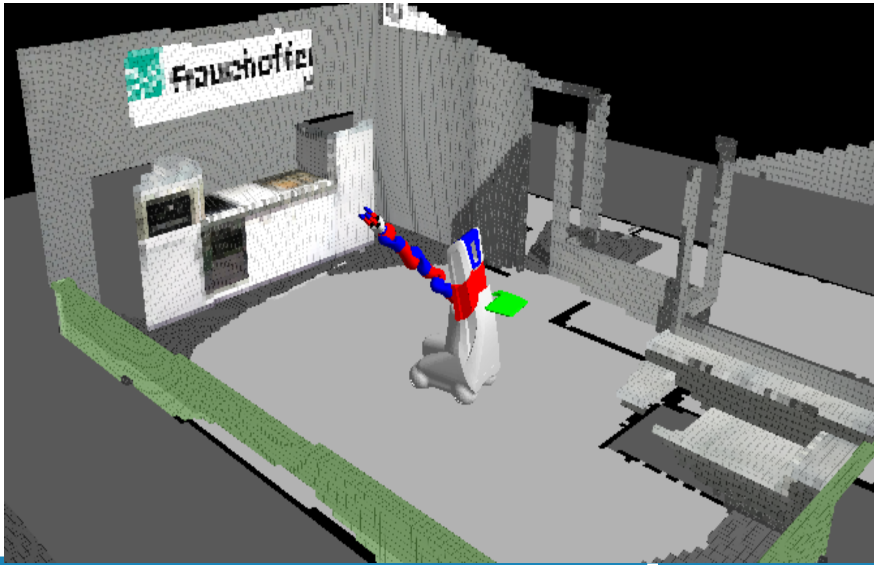
```
global_frame                       - world frame id
original_frame                     - original point cloud id (not necessary)
planedet_ht_keep_track             - if 0, HS is cleared at each frame
planedet_ht_minshift               - HS minimum d dimension
planedet_ht_maxshift               - HS maximum d dimension
planedet_ht_angle_res              - HS angle resolution (number of chunks)
planedet_ht_shift_res              - HS d resolution
planedet_ht_gauss_angle_res        - HS smoothing kernel resolution (angle)
planedet_ht_gauss_shift_res        - HS smoothing kernel resolution (d)
planedet_ht_gauss_angle_sigma      - HS smoothng kernel angle sigma
planedet_ht_gauss_shift_sigma      - HS smoothng kernel d sigma
```

Outline

- Point cloud processing and geometric mapping
  - Intro to the PCL library
  - Plane detection using PCL
  - (SRS) geometric mapping

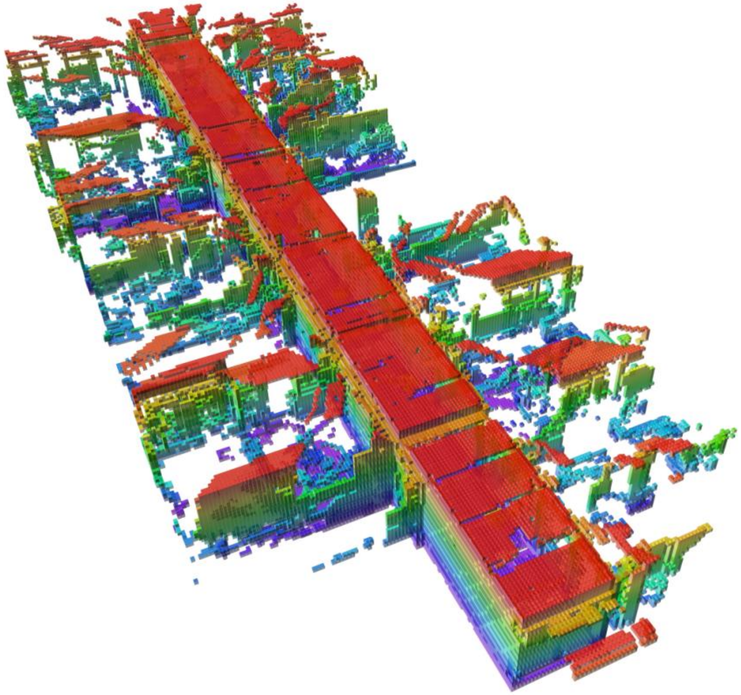


- Global voxel-based map of the environment
  - Intro to the Octomap library
  - Example node
  - (SRS) env. model



Introduction to the Octomap Library

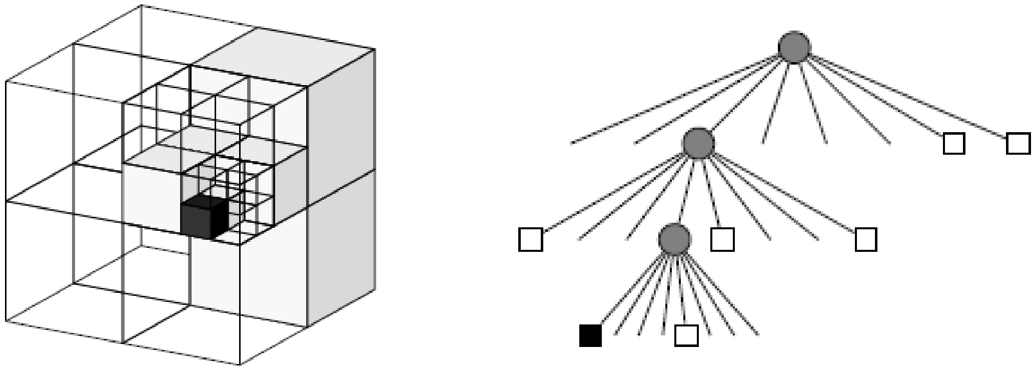
- **http://octomap.sourceforge.net/**
- Probabilistic 3D Mapping Framework
- Core features:
  - All information is stored in probabilistic manner (including „no info yet“ case
  - Flexibility – in size, in resolution
  - Efficiency – dynamic memory allocation, compressed file



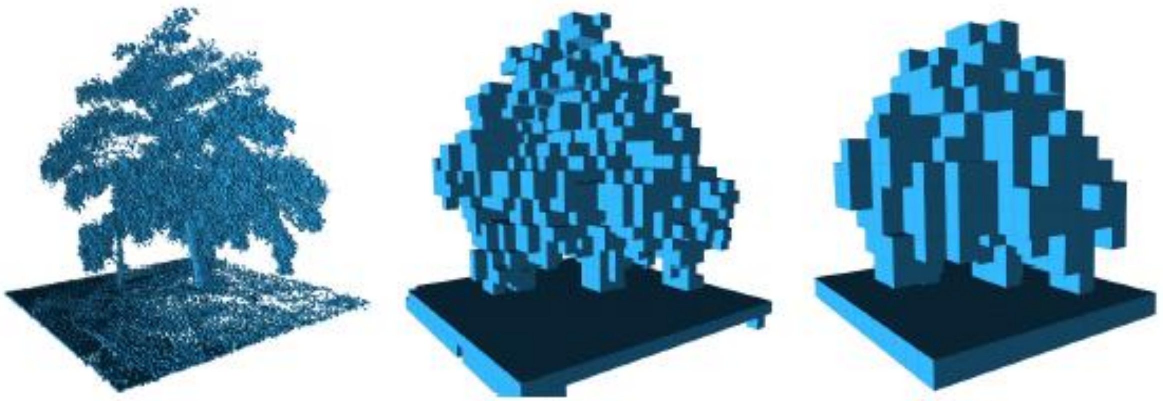
- Introduction and basics
  - http://www.youtube.com/watch?v=25nnJ64ED5Q

Octomap Library, contd.

- Hierarchical representation of data – octree structure



- Level of detail



Octomap Library, contd.

- Combination of probabilistic and discrete occupancy mapping
- Probabilistic node labeling – needed for „fuzzy“ data
  - Occupied
  - Free
  - Unknown
- Discrete labeling – enables pruning
  - Changing nodes
  - Stable nodes (defined by thresholds)

Octomap Library, contd.

- Example model
  - http://www.youtube.com/watch?v=O2TDNJuHMKo

K. M. Wurm, A. Hornung, M. Bennewitz, C. Stachniss and W. Burgard: *OctoMap: A probabilistic, flexible, and compact 3D map representation for robotic systems*, In Proc. of the ICRA 2010

Octomap Library in ROS

- Up to ROS Electric
  - http://www.ros.org/wiki/octomap_ros
  - http://www.ros.org/wiki/octomap_msgs
  - http://www.ros.org/wiki/octomap_server (map building and serving capabilities)
  - http://www.ros.org/wiki/octovis (visualization)

- Octomap in ROS Fuerte and later
  - Replaced with a rosdep system dependency

```
sudo apt-get install ros-fuerte-octomap
```

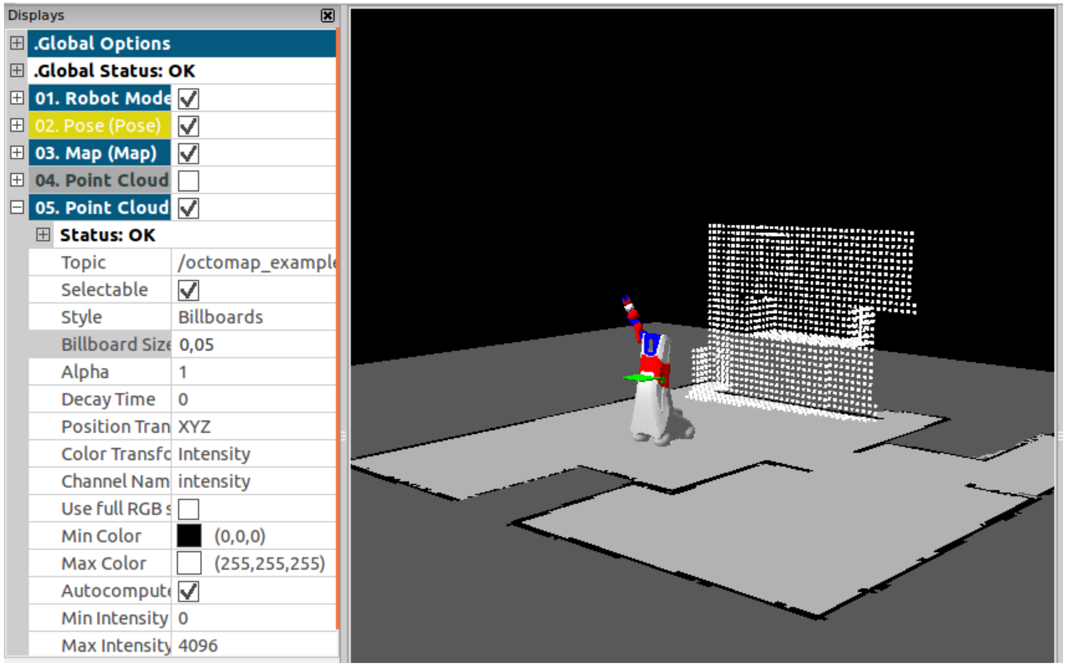
Octomap Sample Node

- Build examples

```
rosmake tb2_3d_env_mapping
```

- Run the launch file

```
roslaunch tb2_3d_env_mapping octomap_example_test_with_sim_tb2.launch
```



Displays

.Global Options	
.Global Status: OK	
01. Robot Mode	☑
02. Pose (Pose)	☑
03. Map (Map)	☑
04. Point Cloud	□
05. Point Cloud	☑
Status: OK	
Topic	/octomap_example
Selectable	☑
Style	Billboards
Billboard Size	0,05
Alpha	1
Decay Time	0
Position Tran	XYZ
Color Transfc	Intensity
Channel Nam	intensity
Use full RGB s	□
Min Color	■ (0,0,0)
Max Color	□ (255,255,255)
Autocompute	☑
Min Intensity	0
Max Intensity	4096

- Run the teleop

```
roslaunch btb_teleop keyboard_teleop.launch
```

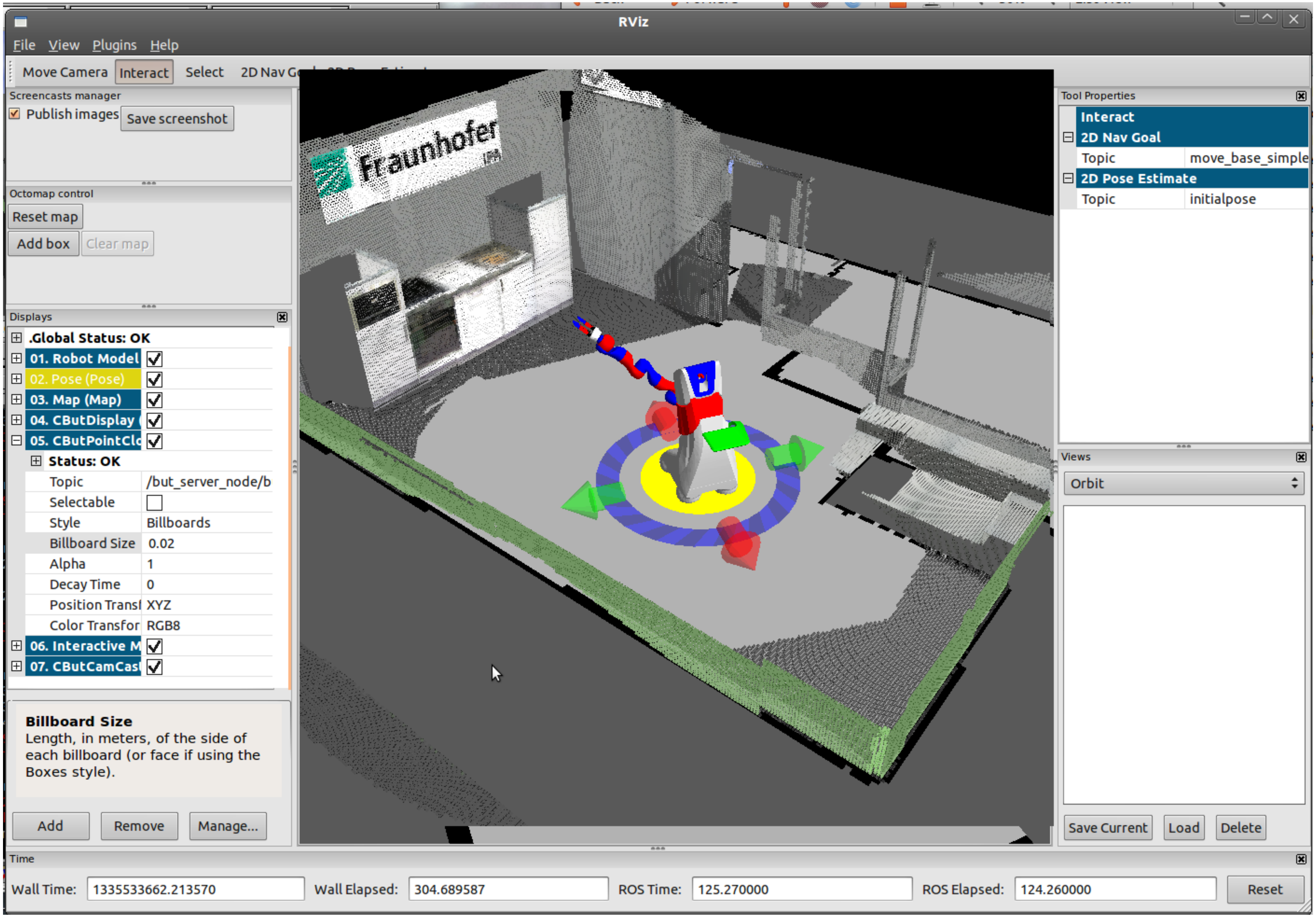
Octomap Sample Node, contd.

- Let’s see what’s behind…

(SRS) Environment Model

- Contains basic functionality of standard octomap_server (octomap_mapping stack)
  - Configuration and basic control services
  - Point cloud input
  - Output: Point cloud, collision map and collision objects, marker array, octomap
- Additional features
  - Plugin components architecture
  - Octomap filtering (noise removing, fast updates)
  - Differential frames publication
  - On demand locking
  - Interactive changes in data (octomap, collision map)

(SRS) Environment Model



☑ Publish images

.Global Status: OK	
01. Robot Model	☑
02. Pose (Pose)	☑
03. Map (Map)	☑
04. CButDisplay (	☑
05. CButPointClo	☑
Status: OK	
Topic	/but_server_node/b
Selectable	□
Style	Billboards
Billboard Size	0.02
Alpha	1
Decay Time	0
Position Transl	XYZ
Color Transfor	RGB8
06. Interactive M	☑
07. CButCamCast	☑

Billboard Size
Length, in meters, of the side of each billboard (or face if using the Boxes style).

Interact	
2D Nav Goal	
Topic	move_base_simple
2D Pose Estimate	
Topic	initialpose

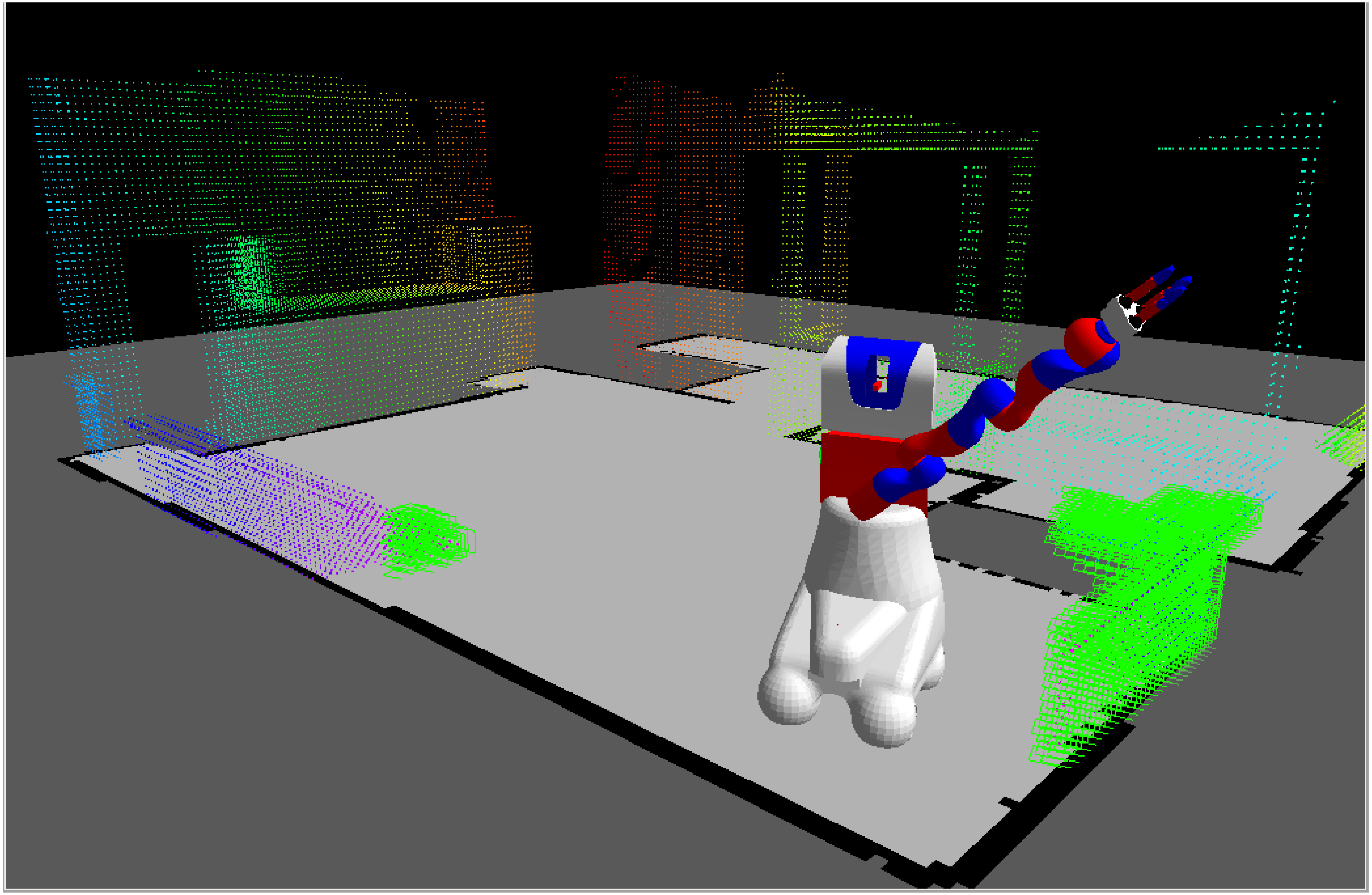
Orbit

Wall Time: 1335533662.213570 Wall Elapsed: 304.689587 ROS Time: 125.270000 ROS Elapsed: 124.260000

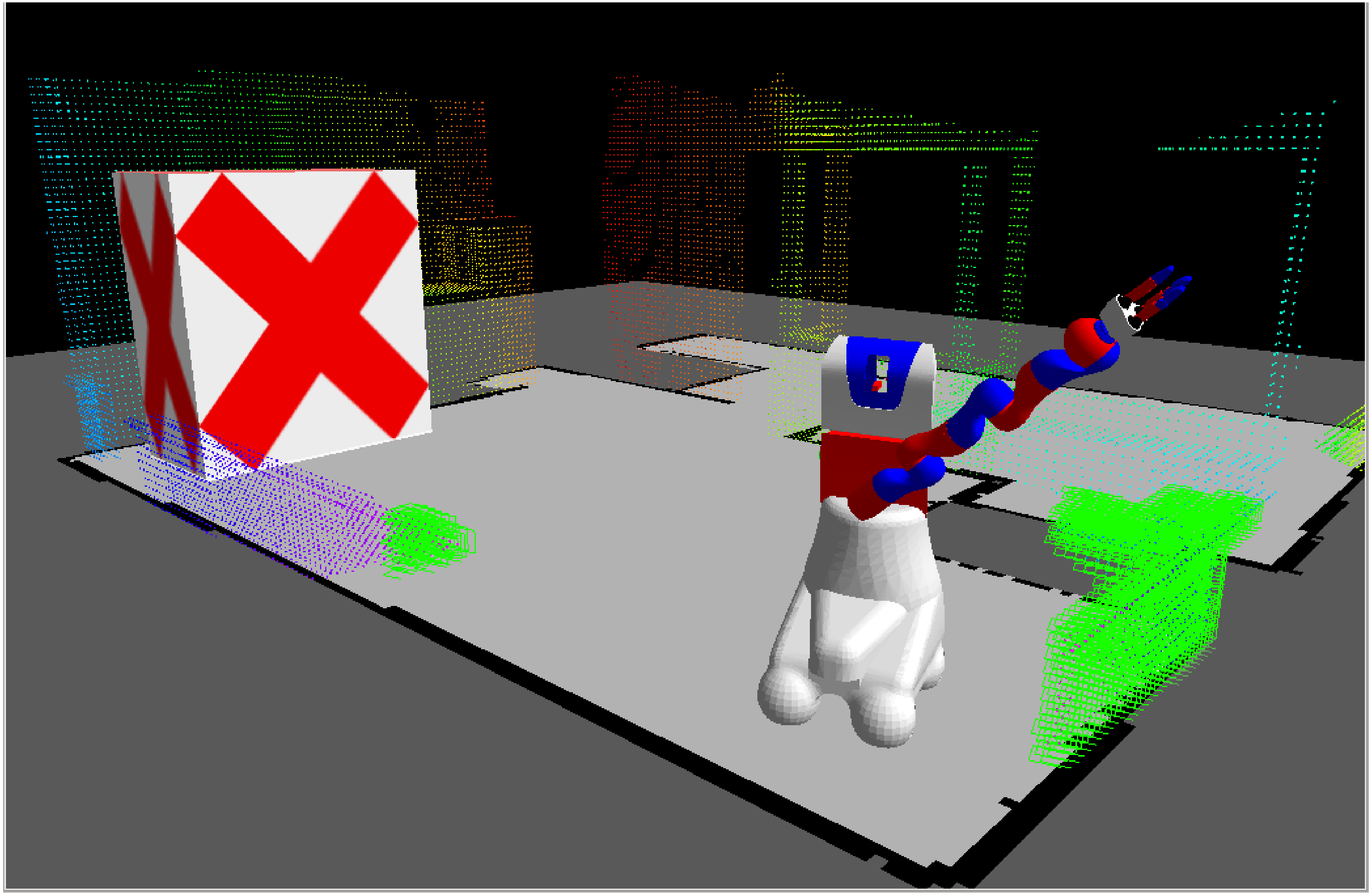
(SRS) Env. Model – Octomap Filtering

- Noised input data
- Isolated cells removing
  - Find all cell without neighbours and set them as free
- Incorrectly detected data removing
  - Search along a ray and set all cells as free
- Obsolete data removing
  - Timestamps
  - Older cells withering up

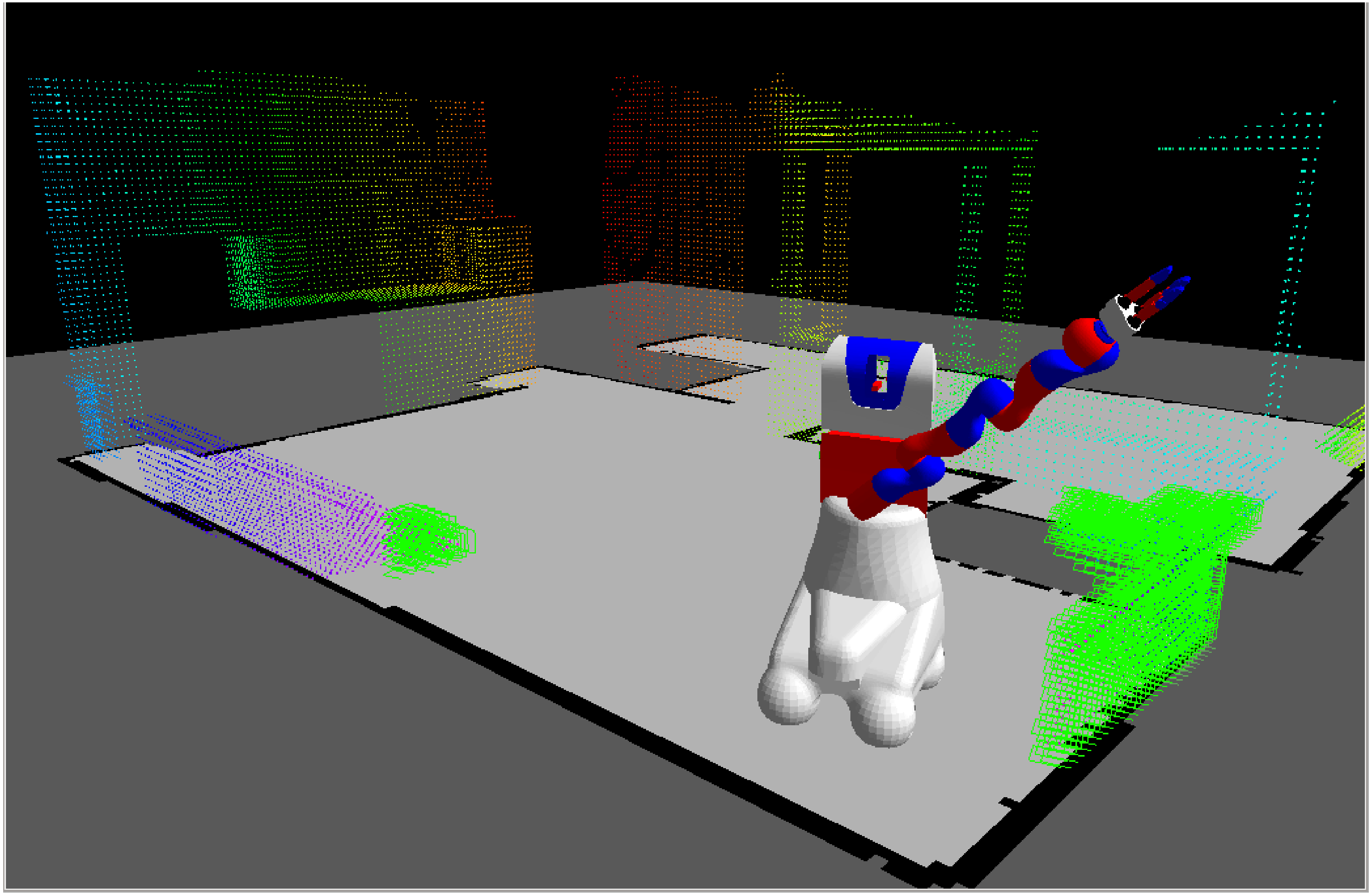
(SRS) Env. Model – Interactive Changes in Data



(SRS) Env. Model – Interactive Changes in Data

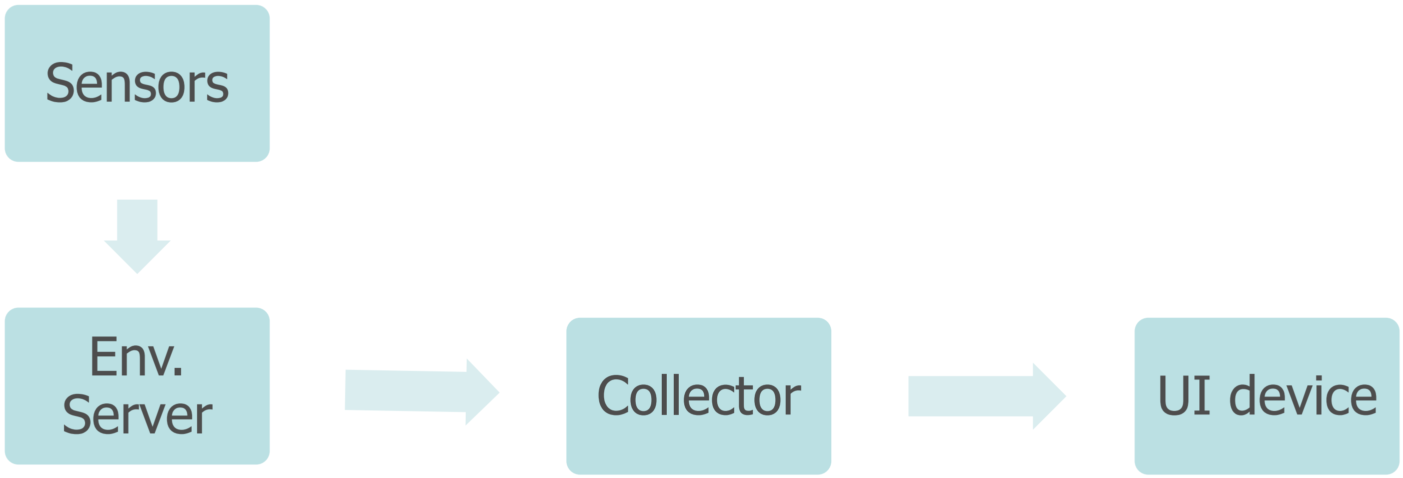


(SRS) Env. Model – Interactive Changes in Data



(SRS) Env. Model – Differential Frames Publication

- Minimizes data flow for visualization purposes
  - E.g. teleoperated robots
- Only changed (≈currently visible) part of octomap is published
- Additional node is needed – „collector“



Coding Pitfalls…

- Eigen’s memory alignment
  - Aligned allocator required when creating STL containers and classes containing Eigen types…

```
//std::vector<pcl::PointCloud<pcl::PointXYZ> > planecloud(model-
>planes.size());

typedef pcl::PointCloud<pcl::PointXYZ> PointCloud;
typedef std::vector<PointCloud,
                    Eigen::aligned_allocator<PointCloud> >
                    PlaneCloud;
PlaneCloud planecloud(model->planes.size());
```

- http://eigen.tuxfamily.org/dox/TopicUnalignedArrayAssert.html

Coding Pitfalls…

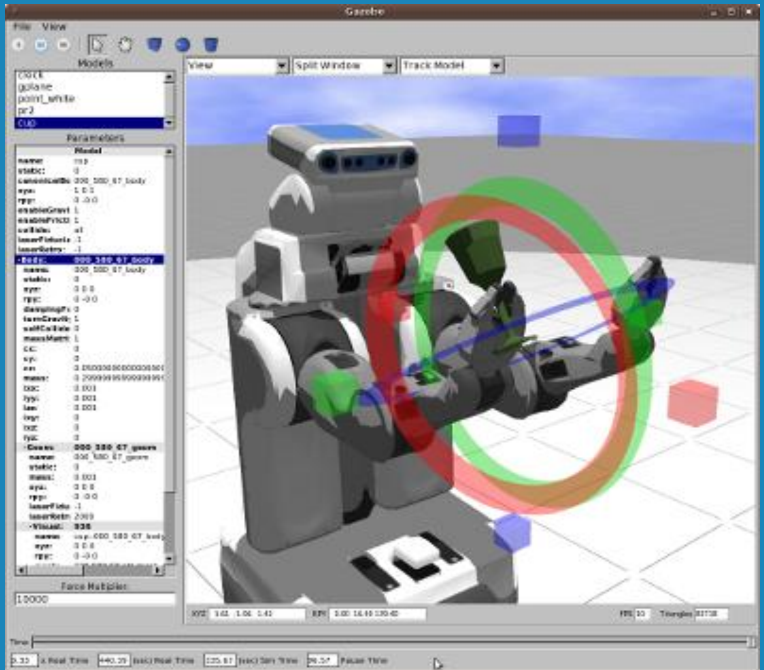
- Debugging nodes executed from a launch file
  - launch-prefix attribute of the <node>

```
launch-prefix="gdb -ex run --args“
launch-prefix="valgrind“

<node pkg=“my_pkg" type=“my_node" name=“my_node“
    launch-prefix="gdb -ex run --args“>
    …
</node>
```

- http://www.ros.org/wiki/roslaunch/Tutorials/Roslaunch%20Nodes%20in%20Valgrind%20or%20GDB

ROS rulez!


Gazebo
File View
Models
clock
gplane
point_white
pr2
cup
Parameters
View
Split Window
Track Model
Force Multiplier
10000