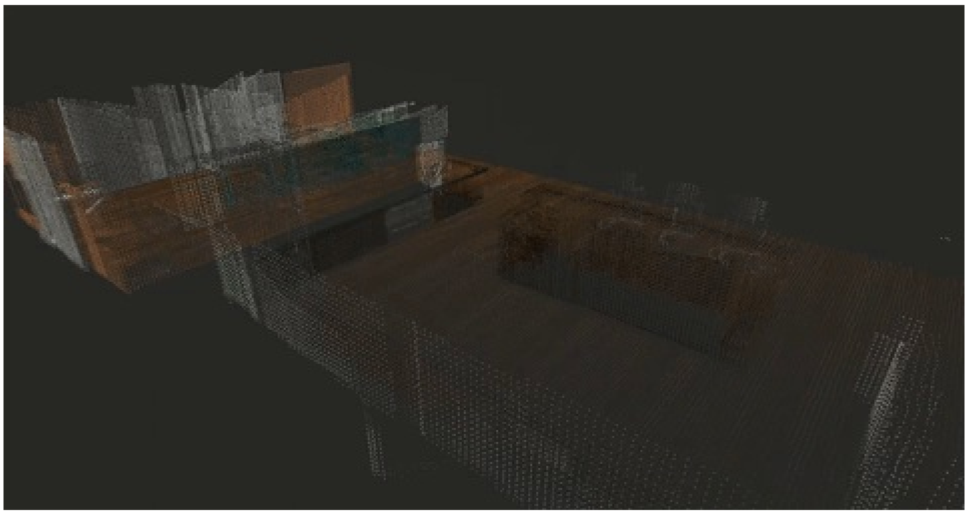
Map My World Robot

Abstract

The target of this project is to create a 2D occupancy grid and 3D octomap from a provided simulated environment. Furthermore, create a simulated environment (world) to be mapped as well.



The RTAB-Map framework is used to allow the robot to map the environment in 3D. Also, Gazebo and ROS are used for simulation.

Background

In robotic mapping and navigation, the simultaneous localization and mapping (SLAM) is the computational problem of constructing or updating a map of an unknown environment while simultaneously keeping track of an agent's location within it. We need a map to localize and we need to localize to make the map.

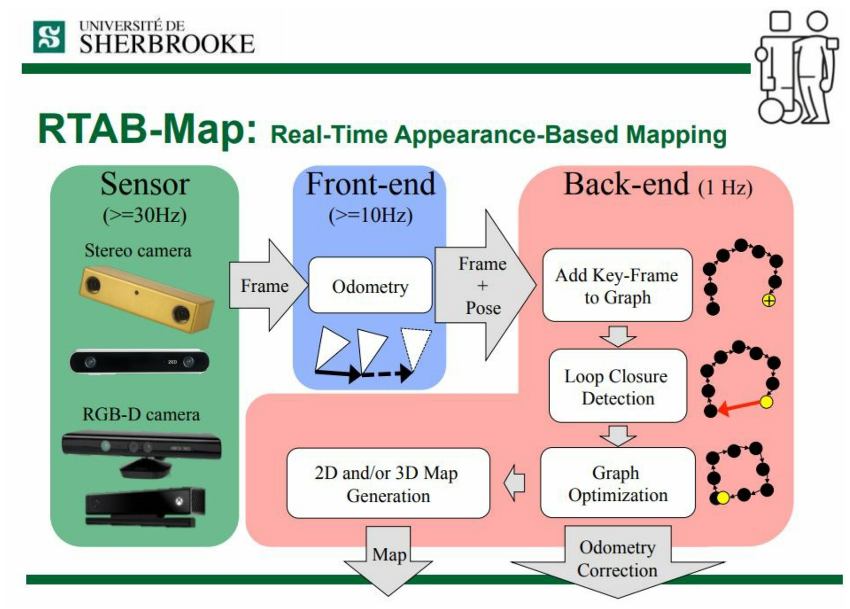


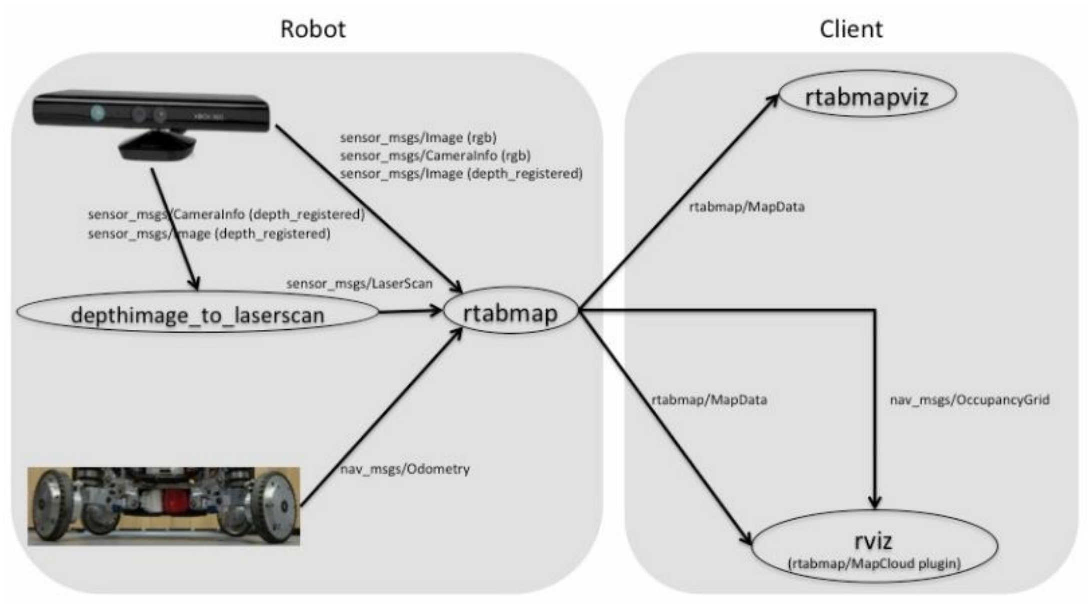
Fig. 2 - Workflow how RTAB-Map works.

RTAB-Map (Real-Time Appearance-Based Mapping) is a RGB-D Graph-Based SLAM approach based on an incremental appearance-based loop closure detector. The loop closure detector uses a bag-of-words approach to determinate how likely a new image comes from a previous location or a new location. When a loop closure hypothesis is accepted, a new constraint is added to the map's graph, then a graph optimizer minimizes the errors in the map. A memory management approach is used to limit the number of locations used for loop closure detection and graph optimization, so that real-time constraints on large-scale environnements are always respected. In this project a robot is equiped with Kinect and RTAB-Map to generate 6DoF RGB-D mapping.

Introduction

This project uses rtabmap_ros that is a wrapper of RTAB-Map to provide the mapping functionalities used by the robot to map the environment. This package can be used to generate a 3D point clouds of the environment and/or to create a 2D occupancy grid map for navigation.

For image capture the project uses a Kinect camera (RGB-D) and, to make the Kinect publish in the scan topic, the project uses the following configuration.



Note that the configuration provides a method of mapping the depth point cloud into a usable scan topic that can be managed by RTAB-Map.

Selecting the Robot

The target of this step is to chose a robot type, build the urdf model, load it into Gazebo and run the robot simulation to map the environment using the rtabmap-ros package. So, why not get a real robot to use as reference in the sumulation? Then we can apply the lessons learned in the real model. That sounds good. So, the figure below show the model selected.

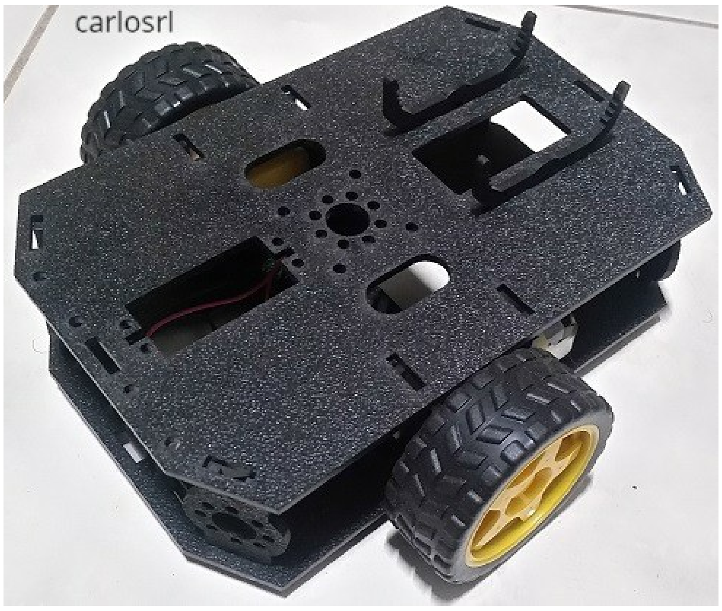


Fig. 4 - Robot select as reference.

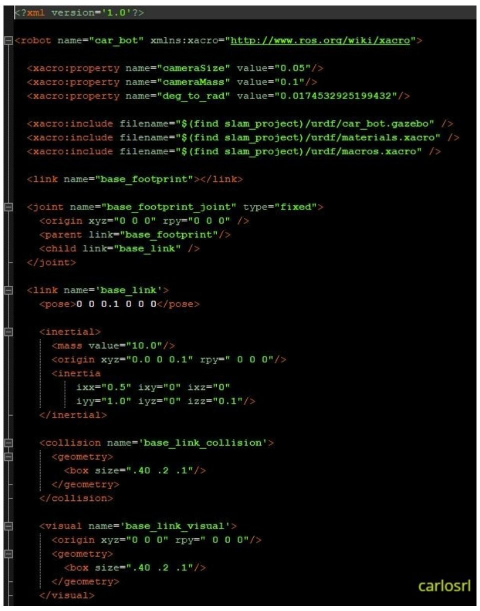
It is a two differential whelled robot using two step motors. So, adding more hardware components we can use it for mapping.

Building the Robot

The urdf model must have:

- chassis
- right wheel
- left wheel
- caster
- RGB-D camera
- RGB camera
- laser scan

Based on the list above, two files were created: car_bot.xacro and car_bot.gazebo. The car_bot.xacro contain the urdf model definition and car_bot.gazebo contain the urdf sensor definition. The Fig. 5 shows part of the car_bot.xacro.



```xml
<?xml version='1.0'?>

<robot name="car_bot" xmlns:xacro="http://www.ros.org/wiki/xacro">

  <xacro:property name="cameraSize" value="0.05"/>
  <xacro:property name="cameraMass" value="0.1"/>
  <xacro:property name="deg_to_rad" value="0.0174532925199432"/>

  <xacro:include filename="$(find slam_project)/urdf/car_bot.gazebo" />
  <xacro:include filename="$(find slam_project)/urdf/materials.xacro" />
  <xacro:include filename="$(find slam_project)/urdf/macros.xacro" />

  <link name="base_footprint"></link>

  <joint name="base_footprint_joint" type="fixed">
    <origin xyz="0 0 0" rpy="0 0 0" />
    <parent link="base_footprint"/>
    <child link="base_link" />
  </joint>

  <link name='base_link'>
    <pose>0 0 0.1 0 0 0</pose>

    <inertial>
      <mass value="10.0"/>
      <origin xyz="0.0 0 0.1" rpy=" 0 0 0"/>
      <inertia
          ixx="0.5" ixy="0" ixz="0"
          iyy="1.0" iyz="0" izz="0.1"/>
    </inertial>

    <collision name='base_link_collision'>
      <geometry>
        <box size=".40 .2 .1"/>
      </geometry>
    </collision>

    <visual name='base_link_visual'>
      <origin xyz="0 0 0" rpy=" 0 0 0"/>
      <geometry>
        <box size=".40 .2 .1"/>
      </geometry>
    </visual>
```

Fig. 5 - Part of robot URDF model.

To see the final urdf model we can take a look the picture of frames in Fig. 6 that shows the robot model has a camera and a rgbd frames.

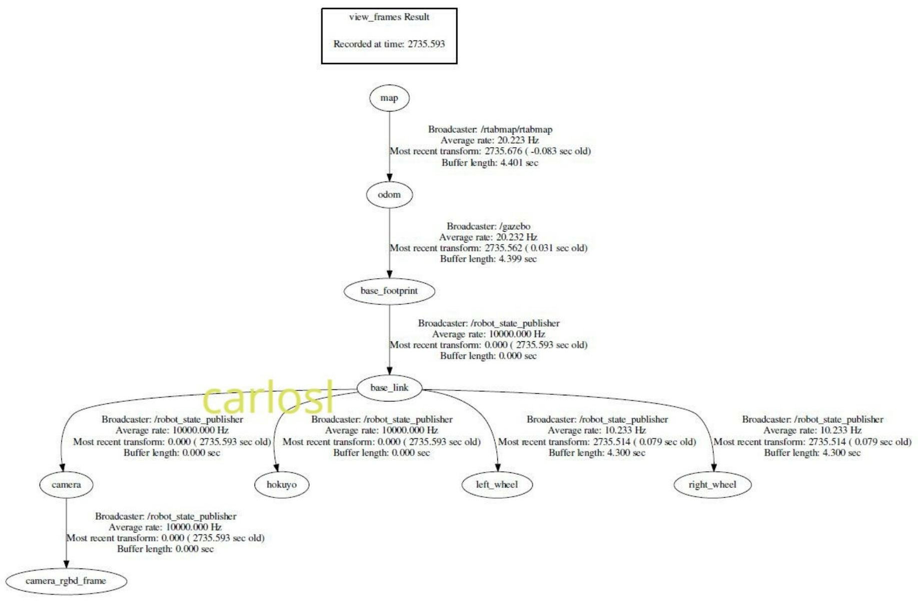

view_frames Result
Recorded at time: 2735.593
map
Broadcaster: /rtabmap/rtabmap
Average rate: 20.223 Hz
Most recent transform: 2735.676 (-0.083 sec old)
Buffer length: 4.401 sec
odom
Broadcaster: /gazebo
Average rate: 20.232 Hz
Most recent transform: 2735.562 (0.031 sec old)
Buffer length: 4.399 sec
base_footprint
Broadcaster: /robot_state_publisher
Average rate: 10000.000 Hz
Most recent transform: 0.000 (2735.593 sec old)
Buffer length: 0.000 sec
base_link
Broadcaster: /robot_state_publisher
Average rate: 10000.000 Hz
Most recent transform: 0.000 (2735.593 sec old)
Buffer length: 0.000 sec
Broadcaster: /robot_state_publisher
Average rate: 10000.000 Hz
Most recent transform: 0.000 (2735.593 sec old)
Buffer length: 0.000 sec
Broadcaster: /robot_state_publisher
Average rate: 10.233 Hz
Most recent transform: 2735.514 (0.079 sec old)
Buffer length: 4.300 sec
Broadcaster: /robot_state_publisher
Average rate: 10.233 Hz
Most recent transform: 2735.514 (0.079 sec old)
Buffer length: 4.300 sec
camera
hokuyo
left_wheel
right_wheel
Broadcaster: /robot_state_publisher
Average rate: 10000.000 Hz
Most recent transform: 0.000 (2735.593 sec old)
Buffer length: 0.000 sec
camera_rgbd_frame

Fig. 6 - Robot frames.

Using xacro elements and macros in xacro file, we can contruct the entire urdf model and load it into Gazebo tool. The result is in Fig. 7.

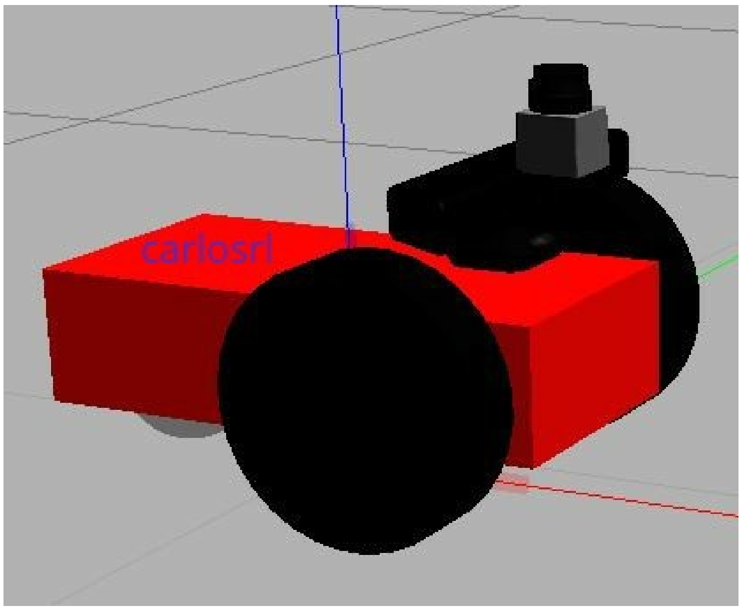


Fig. 7 - The robot model spawned in Gazebo tool.

World: the provided world

As the project title states, the project target is to map the world. So we have two worlds to be managed: the provided world and a new one that must be createded. The Fig. 8 shows the provided kitchen dining world.

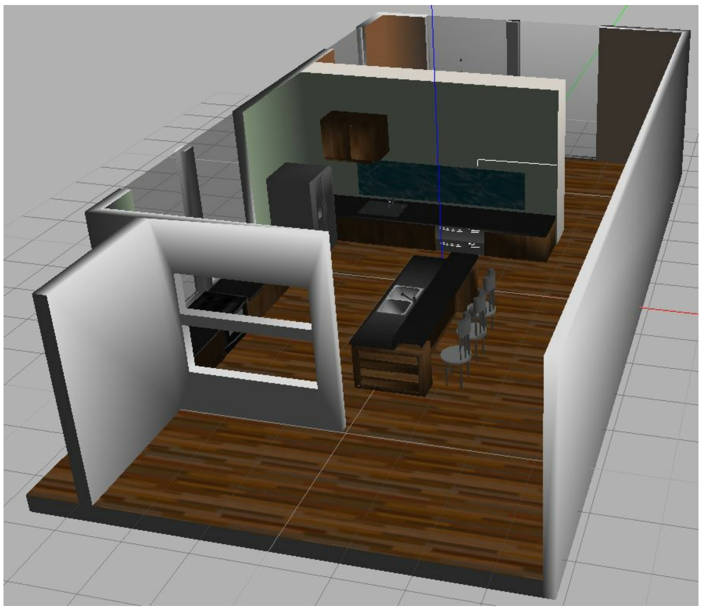


Fig. 8 - The kitchen dining world

Note that the environment has a lot of features which will demand a heavy hardware resource in the mapping process.

World: Creating a new world

To create a new world we can use the `Build Editor` option in Gazebo tool. In the editor we can use existing objects and models or create new objects using the features provided in the editor tool. So, using walls, tables and others objects, we created a new world as showed in Fig. 9.

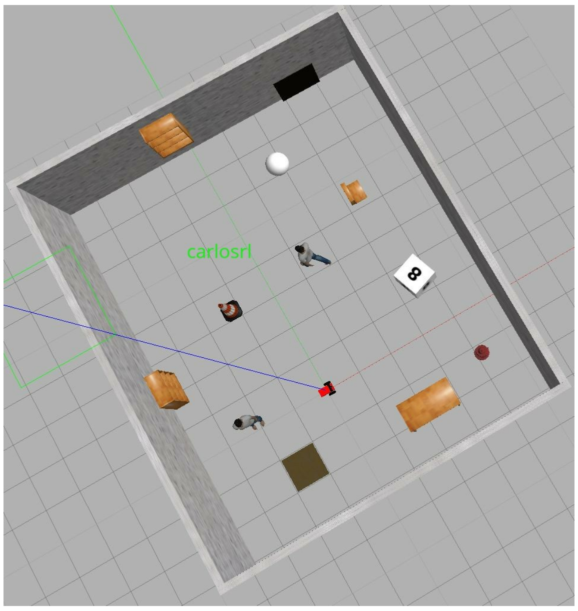


Fig. 9 - The new created world

Creating the launch files

At this point we have the robot model and the world environments. To manage the mapping process we need load the robot model, the world, launch the Gazebo simulation, launch the rtabmap-ros package and a teleop script to teleoperate the robot in the environment to allow it gather as much features, from the world, as possible.

To make the things easier, we can automate these steps creating launch files that will contain a set of commands that are executed using the `roslaunch` tool.

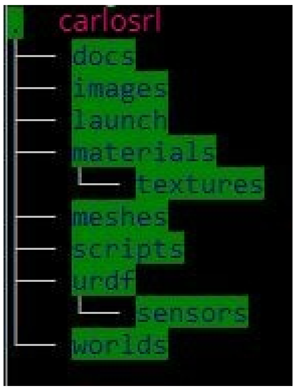
For illustration, the following snippet code is the world.launch content.

```
<?xml version="1.0"?>
<launch>
  <arg name="world" default="empty"/>
  <arg name="paused" default="false"/>
  <arg name="use_sim_time" default="true"/>
  <arg name="gui" default="true"/>
  <arg name="headless" default="false"/>
  <arg name="debug" default="false"/>

<!-- We resume the logic in empty_world.launch, changing only the name of the
world to be launched -->
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <!-- <arg name="world_name" value="$(find
slam_project)/worlds/kitchen_dining.world"/> -->
    <arg name="world_name" value="$(find
slam_project)/worlds/user_created.world"/>
    <arg name="paused" value="$(arg paused)"/>
    <arg name="use_sim_time" value="$(arg use_sim_time)"/>
    <arg name="gui" value="$(arg gui)"/>
    <arg name="headless" value="$(arg headless)"/>
    <arg name="debug" value="$(arg debug)"/>
  </include>

  <!-- spawn a robot in gazebo world -->
  <include file="$(find slam_project)/launch/car_bot_description.launch"/>

  <node name="car_bot_spawner" pkg="gazebo_ros" type="spawn_model"
respawn="false" output="screen"
   args="-urdf -param robot_description -model car_bot"/>
</launch>
```

The launch file has the xml format as showed above. Note that world.launch file contain two wolds that are commented/uncommented depending on which world must be launched.

In addition to world.launch, we have the mapping.launch, used to load the rtabmap-ros package and the rtabmapviz or rviz, depending on the selection in the command line and the rviz.launch, used to load the rviz tool. Instead of rviz.launch, a different solution could be add the rviz command inside the mapping.launch and select it in command line.

The final directory structure is showed in Fig. 10.



```
.  carlosrl
├── docs
├── images
├── launch
├── materials
│   └── textures
├── meshes
├── scripts
├── urdf
│   └── sensors
└── worlds
```

Fig. 10 - Directory structure

Robot, map the world

Putting all together, we can now start the mapping process.

Start launching the world and the robot model using the command

```
roslaunch slam_project world.launch
```

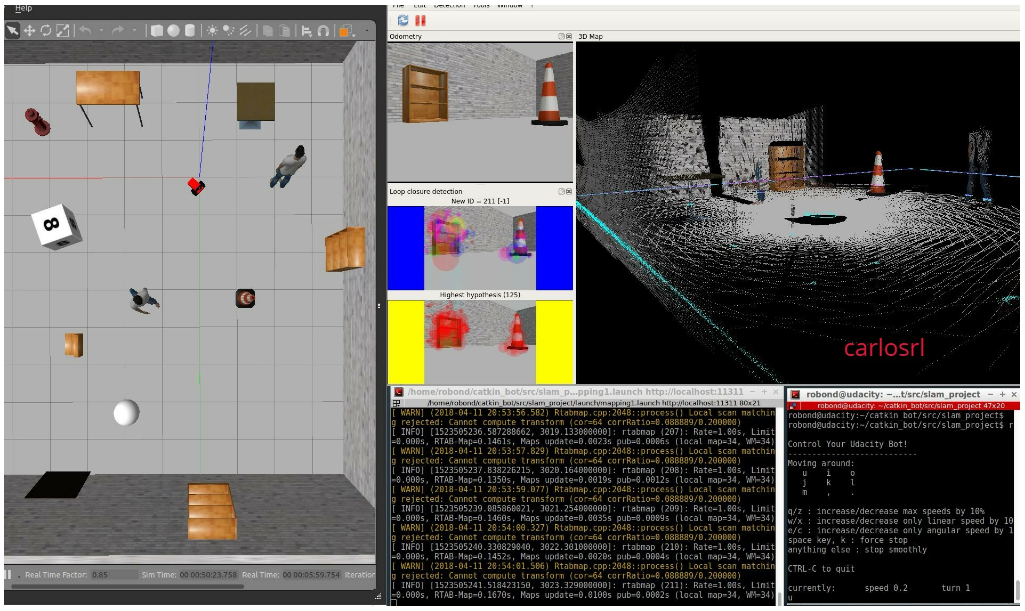
Now launch the rtabmap with rtabmaprviz tool, using the command

```
roslaunch slam_project mapping.launch
```

Finally, launch the teleop script to control the robot using the command

```
rosrun slam_project teleop
```

Mapping the created environment can be seen in Fig. 11

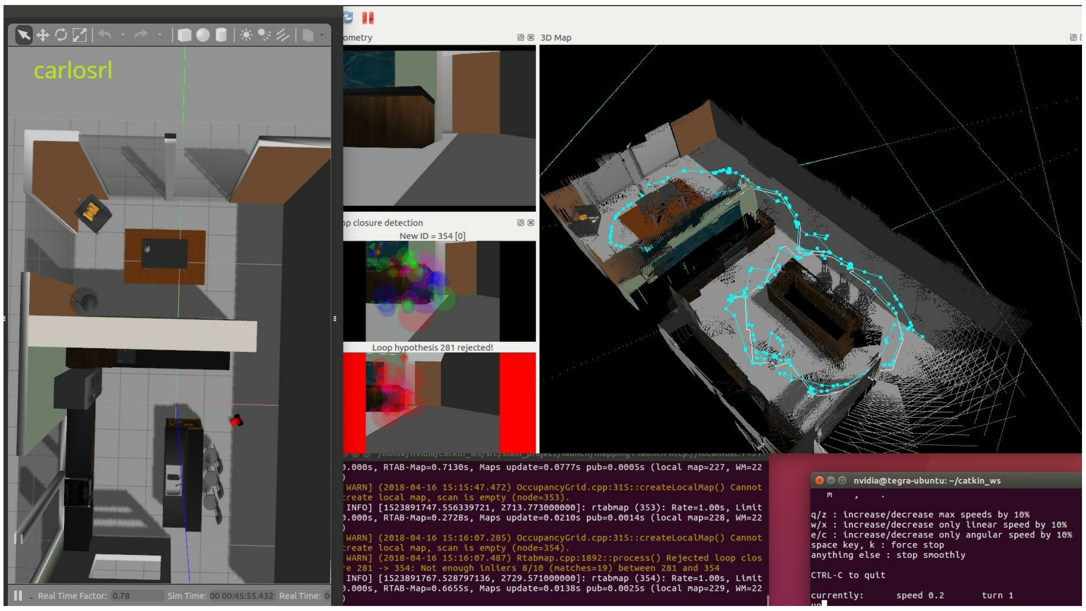


```
/home/robond/catkin_bot/src/slam_project/launch/mapping1.launch http://localhost:11311 80x21
[ WARN] (2018-04-11 20:53:56.582) Rtabmap.cpp:2048::process() Local scan matchin
g rejected: Cannot compute transform (cor=64 corrRatio=0.088889/0.200000)
[ INFO] [1523505236.587288662, 3019.133000000]: rtabmap (207): Rate=1.00s, Limit
=0.000s, RTAB-Map=0.1461s, Maps update=0.0023s pub=0.0006s (local map=34, WM=34)
[ WARN] (2018-04-11 20:53:57.829) Rtabmap.cpp:2048::process() Local scan matchin
g rejected: Cannot compute transform (cor=64 corrRatio=0.088889/0.200000)
[ INFO] [1523505237.838226215, 3020.164000000]: rtabmap (208): Rate=1.00s, Limit
=0.000s, RTAB-Map=0.1350s, Maps update=0.0019s pub=0.0012s (local map=34, WM=34)
[ WARN] (2018-04-11 20:53:59.077) Rtabmap.cpp:2048::process() Local scan matchin
g rejected: Cannot compute transform (cor=64 corrRatio=0.088889/0.200000)
[ INFO] [1523505239.085860021, 3021.254000000]: rtabmap (209): Rate=1.00s, Limit
=0.000s, RTAB-Map=0.1460s, Maps update=0.0035s pub=0.0009s (local map=34, WM=34)
[ WARN] (2018-04-11 20:54:00.327) Rtabmap.cpp:2048::process() Local scan matchin
g rejected: Cannot compute transform (cor=64 corrRatio=0.088889/0.200000)
[ INFO] [1523505240.330829040, 3022.301000000]: rtabmap (210): Rate=1.00s, Limit
=0.000s, RTAB-Map=0.1452s, Maps update=0.0020s pub=0.0004s (local map=34, WM=34)
[ WARN] (2018-04-11 20:54:01.506) Rtabmap.cpp:2048::process() Local scan matchin
g rejected: Cannot compute transform (cor=64 corrRatio=0.088889/0.200000)
[ INFO] [1523505241.518423150, 3023.329000000]: rtabmap (211): Rate=1.00s, Limit
=0.000s, RTAB-Map=0.1670s, Maps update=0.0100s pub=0.0002s (local map=34, WM=34)
```

```
robond@udacity: ~/catkin_bot/src/slam_project 47x20
robond@udacity:~/catkin_bot/src/slam_project$
robond@udacity:~/catkin_bot/src/slam_project$ r

Control Your Udacity Bot!
---------------------------
Moving around:
   u    i    o
   j    k    l
   m    ,    .

q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10
e/c : increase/decrease only angular speed by 1
space key, k : force stop
anything else : stop smoothly

CTRL-C to quit

currently:      speed 0.2       turn 1
u
```

Fig. 11 - Robot mapping the user created world.

Mapping the provided environment can be seen in Fig. 12



```
).000s, RTAB-Map=0.7130s, Maps update=0.0777s pub=0.0005s (local map=227, WM=22
)
 WARN] (2018-04-16 15:15:47.472) OccupancyGrid.cpp:315::createLocalMap() Cannot
create local map, scan is empty (node=353).
 INFO] [1523891747.556339721, 2713.773000000]: rtabmap (353): Rate=1.00s, Limit
).000s, RTAB-Map=0.2728s, Maps update=0.0210s pub=0.0014s (local map=228, WM=22
)
 WARN] (2018-04-16 15:16:07.285) OccupancyGrid.cpp:315::createLocalMap() Cannot
create local map, scan is empty (node=354).
 WARN] (2018-04-16 15:16:07.487) Rtabmap.cpp:1892::process() Rejected loop clos
re 281 -> 354: Not enough inliers 8/10 (matches=19) between 281 and 354
 INFO] [1523891767.528797136, 2729.571000000]: rtabmap (354): Rate=1.00s, Limit
).000s, RTAB-Map=0.6655s, Maps update=0.0138s pub=0.0025s (local map=229, WM=22
)
```

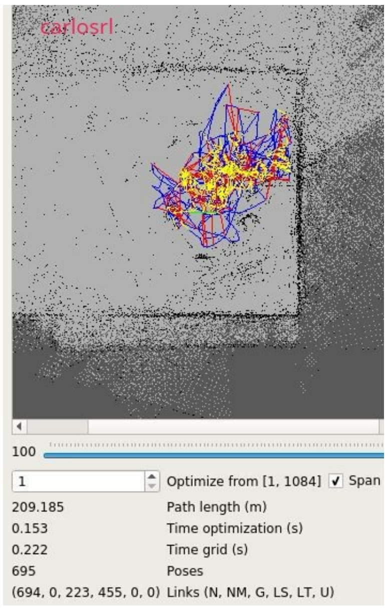
```
nvidia@tegra-ubuntu: ~/catkin_ws
   m    ,    .

q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%
space key, k : force stop
anything else : stop smoothly

CTRL-C to quit

currently:      speed 0.2       turn 1
uo
```

Fig. 12 - Robot mapping the provided world.

Results

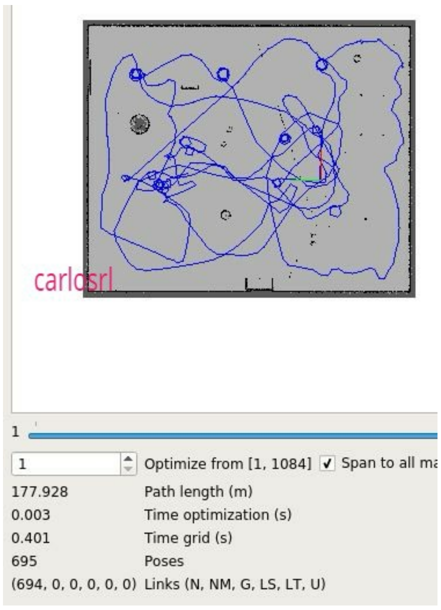
While the robot is moving in the environment, in RTAB-Map we can see in realtime many features being collected like loop closure, feature mathing and so on.

In RTAB-Map loop closure is detected using a bag-of-words approach, that is commonly used in vision-based mapping. In RTAB-Map the default method for extracting features from an image is called Speeded Up Robust Features of SURF.



```
100
1                  Optimize from [1, 1084]  ✓ Span
209.185            Path length (m)
0.153              Time optimization (s)
0.222              Time grid (s)
695                Poses
(694, 0, 223, 455, 0, 0) Links (N, NM, G, LS, LT, U)
```

Fig. 13 - 2D image with no optimization for user created env.



```
1
1                  Optimize from [1, 1084]  ✓ Span to all ma
177.928            Path length (m)
0.003              Time optimization (s)
0.401              Time grid (s)
695                Poses
(694, 0, 0, 0, 0, 0) Links (N, NM, G, LS, LT, U)
```

Fig. 14 - 2D image with optimization for user created env.

Note that global and local closure loop values are zeroed when optimization is applied.

For the provided environment, where we have more features, the 2D map is more clean as we can see in Fig. 15 and 16. Besides, we see few difference between optmized and no optimized 2D graphic.

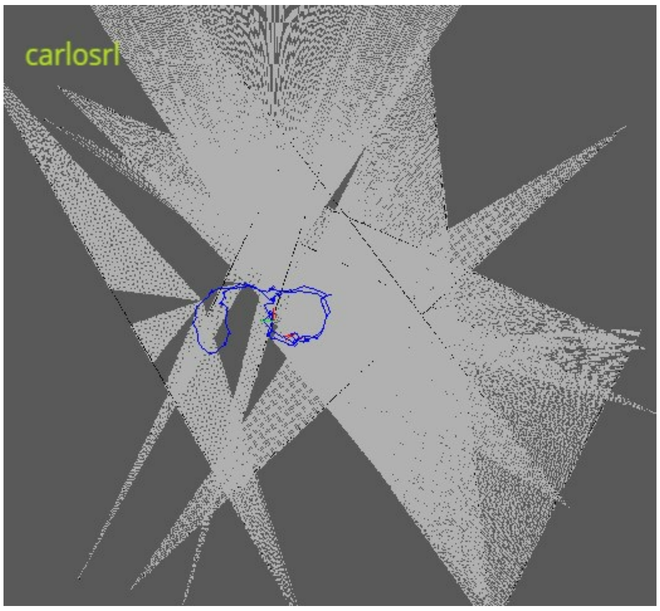


Fig. 15. - 2D image with no optimization for provided env.

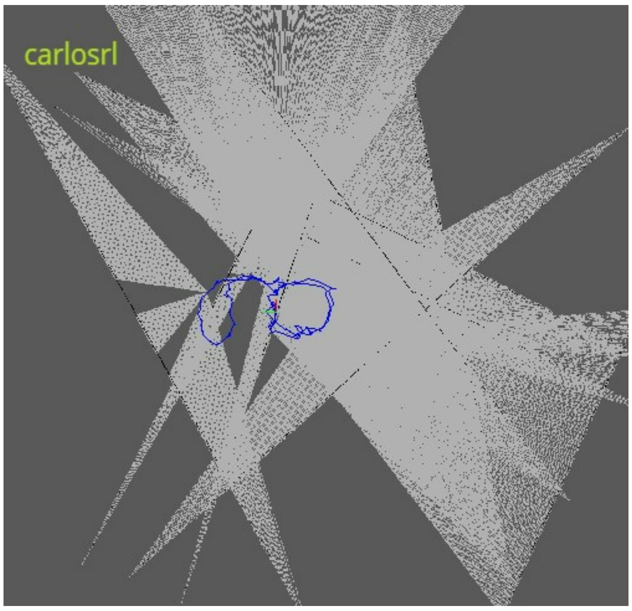


Fig. 16. - 2D image with optimization for provided env.

Now we have the 3D images for both worlds. It is clear that provided world has more features, as we can see reflected when we compare both worlds images.

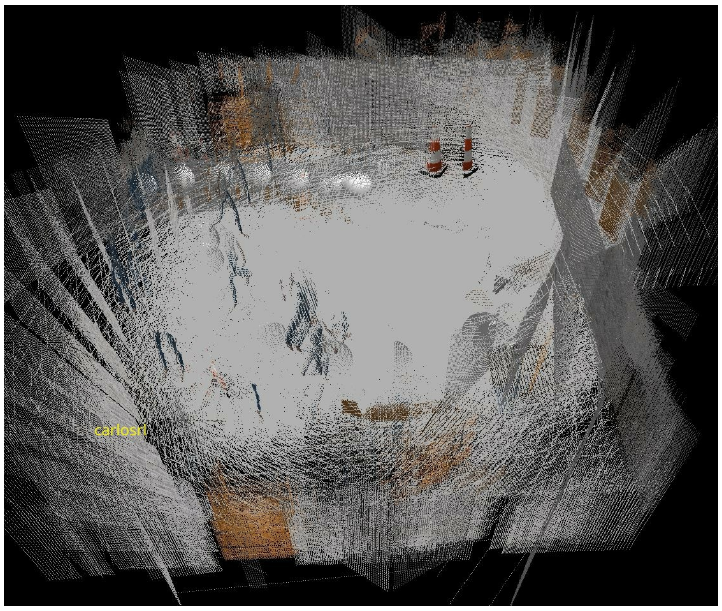


Fig. 17 - 3D image with no optimization for created env.

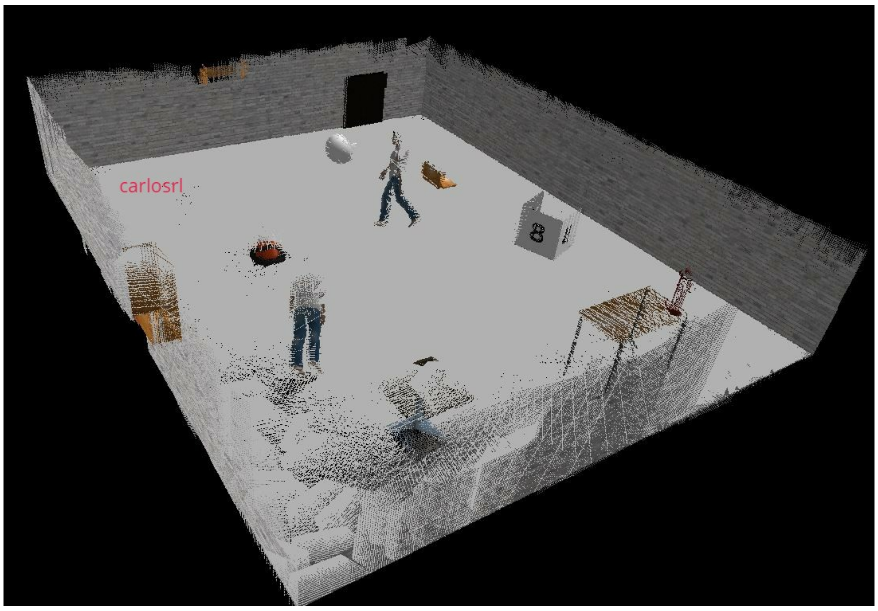


Fig. 18 - 3D image with optimization for created env.

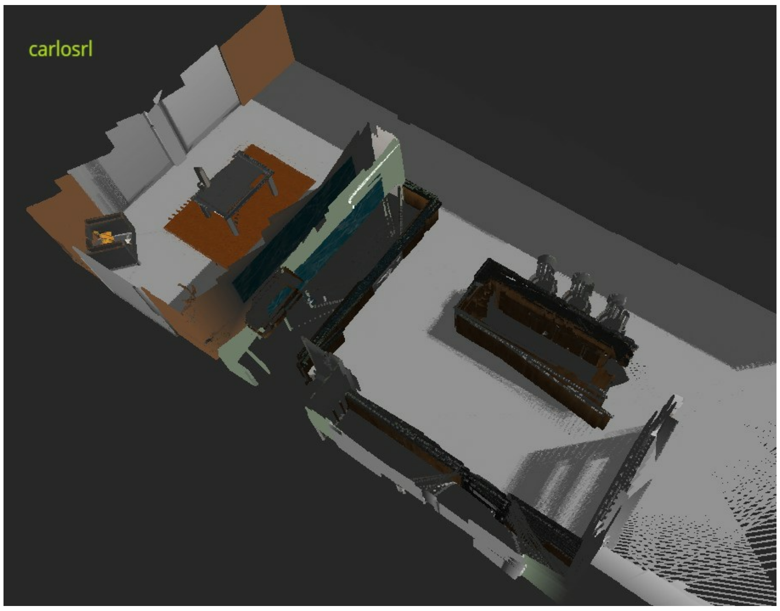


Fig. 19 - 3D image with no optimization for provided env.

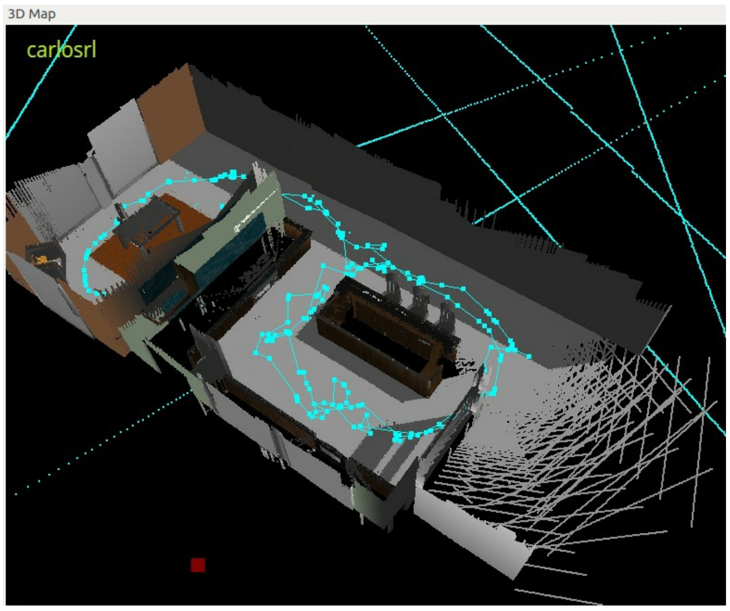
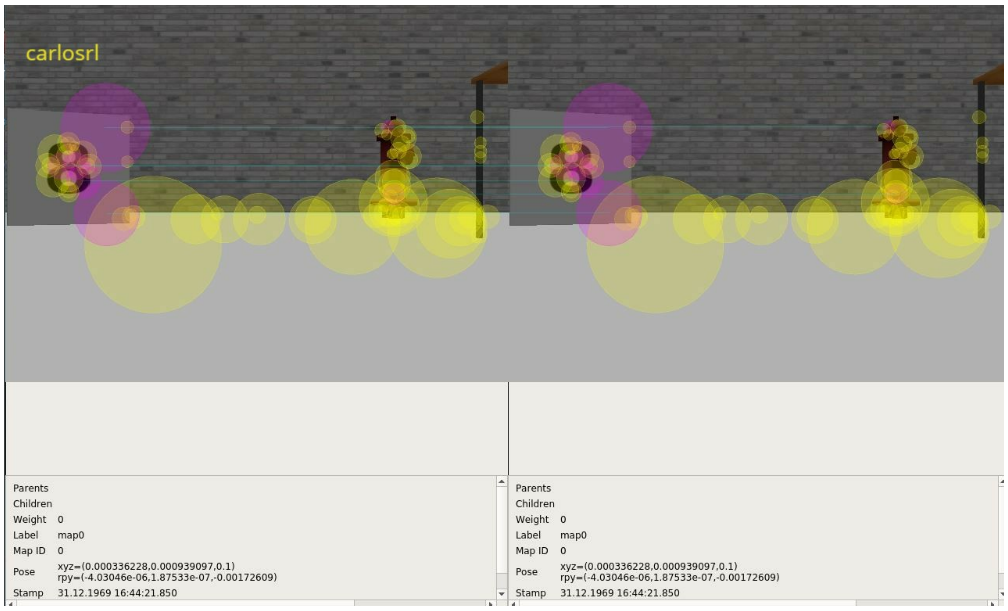


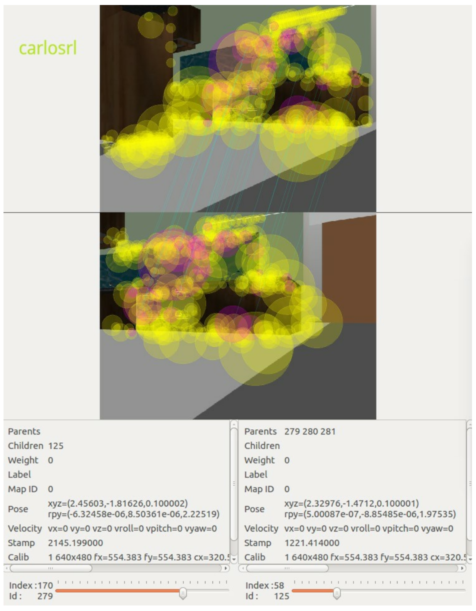
Fig. 20 - 3D image with optimization for provided env.

Then when we compare the mathing features, again the provided world has much more features than the created one.



```
Parents                                                   Parents
Children                                                  Children
Weight    0                                               Weight    0
Label     map0                                            Label     map0
Map ID    0                                               Map ID    0
Pose      xyz=(0.000336228,0.000939097,0.1)               Pose      xyz=(0.000336228,0.000939097,0.1)
          rpy=(-4.03046e-06,1.87533e-07,-0.00172609)                rpy=(-4.03046e-06,1.87533e-07,-0.00172609)
Stamp     31.12.1969 16:44:21.850                         Stamp     31.12.1969 16:44:21.850
```

Fig. 21 - The image shows mathing features



```
Parents                                                   Parents   279 280 281
Children  125                                             Children
Weight    0                                               Weight    0
Label                                                     Label
Map ID    0                                               Map ID    0
Pose      xyz=(2.45603,-1.81626,0.100002)                 Pose      xyz=(2.32976,-1.4712,0.100001)
          rpy=(-6.32458e-06,8.50361e-06,2.22519)                    rpy=(5.00087e-07,-8.85485e-06,1.97535)
Velocity  vx=0 vy=0 vz=0 vroll=0 vpitch=0 vyaw=0          Velocity  vx=0 vy=0 vz=0 vroll=0 vpitch=0 vyaw=0
Stamp     2145.199000                                     Stamp     1221.414000
Calib     1 640x480 fx=554.383 fy=554.383 cx=320.5        Calib     1 640x480 fx=554.383 fy=554.383 cx=320.5

Index :170                                                Index :58
Id :   279                                                Id :   125
```

Fig. 22 - The image shows mathing features

The figures 21 and 22 shows when mathing features are found in both worlds and how one is reacher in features than the other.

Discussion

Thanks to Jetson TX2 that has computational resources sufficent to process the provided world. At first I tried to use the VM environment but it was unable to process the provided world. Moving to TX2 was the solution to finish this project.

The Fig. 21 and 22 shows the Bag-of-Words in action. A feature is a very specific characteristic of the image like a patch with complex texture or a corner or edge. For each feature a unique descriptor is associated to it. While new features are being located, the dictionary is searched loocking for matches that are represented by the lines in the image.

We can see that as more features in the environment, more matching lines we will see in the graphic.

The optimization applied to 2D and 3D maps seems to be very effective once that we can clearly identify the objects in optimized maps which does not occur when we see maps not optimized. But here we can see that this process is less effective when the enrironment has a high number of features as we can see in Fig. 19 and 20 where there is no big differences.

The RTAB-Map showed to see a heavy package in relation to computational resouce.

Future work

As commented in the begining of this document, the idea is to apply this process to a reaal robot.

If we consider apply this project to a drone, due to high computational tasks, we need take care about the battery size that can be consumed quickly if not well dimensioned.