Implementation of the SLAM Algorithm in a Simulated Environment

Pinto, Jorge

Abstract—Simultaneous Localization and Mapping algorithms have been of interests in the latest years due to the increasing number of applications for autonomous mobile robots. The Graph-SLAM algorithm approach RTAB-Map was developed for generating 3D maps of unknown environments, and it can be applied to almost any scenario. In this case, two designs (benchmark vs. custom) were configured in ROS and further analyzed via the RTAB-Map package. Differences in performance were encountered due to the inherent set of features therein contained. Future work could evaluate the in application of different particle filter methods for localization purposes versus that of the RTAB-Map package.

Index Terms—SLAM, Grid Occupancy Algorithm, RtabMap.

1 Introduction

The development of mobile robots intended for transportation, search or rescue operations envisions the development of precise mapping algorithms. As an example, consider the placement of a vacuum cleaning robot in a closed environment; one could argue that uploading a predefined map is a quick solution for the robot to find its way, but the argument fails once the surrounding becomes variable in time and uploading a new map every now and then becomes not efficient. At such condition, a mapping routine would be required to run continuously in order to determine whether a path/place has been previously followed/visited. Moreover, robot's pose uncertainty adds another variable to the problem as it is not possible to know with enough precision its standing point at every time step.

In this sense, an increased interest in Simultaneous Localization and Mapping (SLAM) algorithms has been observed as the installation of external reference systems (e.g. GPS) would impact the robot's own development costs. The current project therefore intends to show the maps generated from applying the SLAM scheme on two simulated environments: a first kitchen-like instance with inherent features, and a second customized environment with extra characteristics. One single robot will be used in both case studies so similar conditions are kept between the simulations. Related work regarding different SLAM applications/flavors can be found in the suggested literature [1]–[4].

2 Formulation

As mentioned by [5], the SLAM problem consists of determining a robot's trajectory while such moves around plus the simultaneous mapping of the environment from the features within. Two methods have been developed therefore:

1) On-line methods, or filtering approaches, which estimates the robot's state from its current position and the map itself, and where such estimation is augmented by including new measurements whenever they are available.
2) Full methods, or smoothing approaches, where the whole trajectory of the robot is determined from the full set of taken measurements.

As such measurements carry an inherent quantity of noise, the SLAM problem into that of a probabilistic nature. To better address its formulation, assume that a mobile robot moving through an environment takes observations from a set of unknown landmarks via an installed range sensor (see Fig.1). As Durrant-White explains [1], the following quantities are taken at a time k:

- X_k: State vector with robot's position and orientation.
- u_k: Control vector applied at *k-1* to reach X_k.
- m_i: location of the ith landmark.
- z_{ik}: observation of the ith landmark at time k taken from the robot.

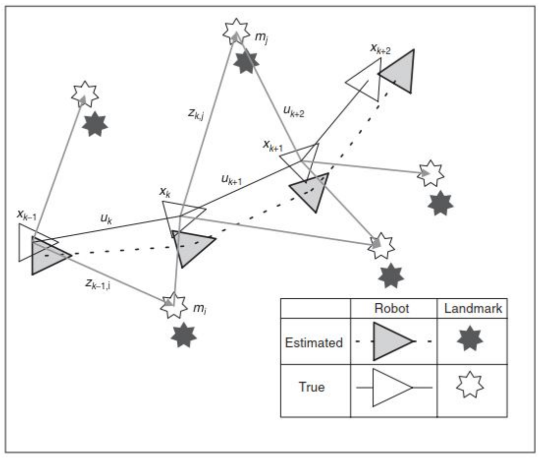


Fig. 1: Graph-Based SLAM formulation as per [1]

The last is called the graph-SLAM problem, where nodes (i.e. robot's poses) are connected by edges (i.e. sensor measurements) that constrain the perceived landmarks, and issue thus reduces to find a configuration of nodes matching as best as possible the set of constraints proposed; that is, an error minimization problem.

That been said, in order to solve the full SLAM problem, it is required to estimate ”the posterior probability of the robot’s trajectory $X_{1:T}$ and the map m of the enviroment given all the measurements plus an initial position X_0” [5]:

$$p = (X_{1:T}, m|Z_{1:T}, u_{1:T}, X_0) \tag{1}$$

Where X_0 is an arbitrary initial position on the map. For this case study, the focus would rather be on a Graph-Slam based approach named Real-Time Appearance-Based Mapping (RTAB-Map) [6]. The implementation is intended for 3D maps’ SLAM as the algorithm bases itself in the analysis of images (gathered from vision sensors) in order to localize the vehicle and map the environment. As a matter of fact, loop-closure processes that occur in real-time, are those who will determine if the robots has previously been in a specific location or not. Consequently, as the map expands by adding new locations, optimization techniques like ”bag of words” [7] and memory management are already included in the algorithm. To prove its strength two simulations were constructed and the incoming sections will not only describe the scenarios, but also all of the considerations taken into account beforehand.

3 Simulations

The arrangements shown in Fig.4a and 4c were used to implement RTAB-Map. A robot model conceived in a previous experiment was also configured to run under Gazebo®and Rviz®simulations for both cases; in this sense, three gazebo plug-ins were included in a definition file before implementation for robot’s sight, surroundings and range detection system, and motion. Moreover, all the necessary ROS kinetic packages were installed in Linux®as well as other ingredients.

3.1 Packages

The package *rtabmap_ros* was included in addition to others already existing (i.e. move_base) in order to enable the tools required for 3D mapping of environments. The moving action was taken via the *Teleop* node so input commands could be introduced via the keyboard.

3.2 Topics

In general, channels for position information transmission was used (i.e., 2D LaserScans plus published odometry). For this case study to succeed however, the original RGB camera of the robot was modified to an RBG-D camera instead in order to enable the processing of depth data from the images. Three topics stand out thereof:

- **rgb/image**: For RGB/Mono image data.
- **rgb/camera_info**: Intended to transmit RGB camera metadata.
- **depth/image**: For registered depth image transmission

On the other hand, ROs original plugin was also modified to that of*libgazebo_ros_openni_kinect.so*. It is important to point out that the new camera had the parameter **camera_rgbd_frame** included to define from which transform would the plugin get the required odometry data for processing. In this sense, a couple of consecutive coordinate frame rotations were required to get the PointCloud data correctly positioned before generating the 3D map. The transform tree shown in Fig.2 accounts for such set of operations.

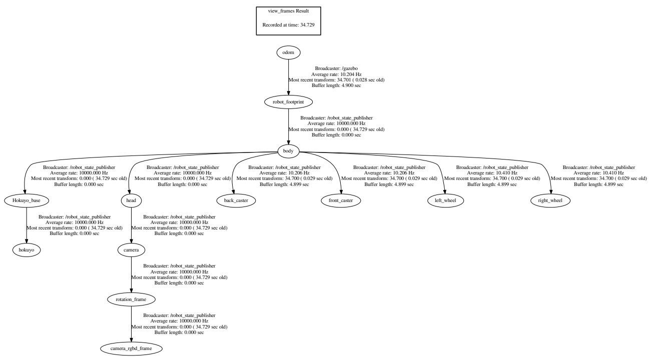

view_frames Result
Recorded at time: 34.729
odom
Broadcaster: /gazebo
Average rate: 10.204 Hz
Most recent transform: 34.701 (0.028 sec old)
Buffer length: 4.900 sec
robot_footprint
Broadcaster: /robot_state_publisher
Average rate: 10000.000 Hz
Most recent transform: 0.000 (34.729 sec old)
Buffer length: 0.000 sec
body
Broadcaster: /robot_state_publisher
Average rate: 10000.000 Hz
Most recent transform: 0.000 (34.729 sec old)
Buffer length: 0.000 sec
Broadcaster: /robot_state_publisher
Average rate: 10000.000 Hz
Most recent transform: 0.000 (34.729 sec old)
Buffer length: 0.000 sec
Broadcaster: /robot_state_publisher
Average rate: 10.206 Hz
Most recent transform: 34.700 (0.029 sec old)
Buffer length: 4.899 sec
Broadcaster: /robot_state_publisher
Average rate: 10.206 Hz
Most recent transform: 34.700 (0.029 sec old)
Buffer length: 4.899 sec
Broadcaster: /robot_state_publisher
Average rate: 10.410 Hz
Most recent transform: 34.700 (0.029 sec old)
Buffer length: 4.899 sec
Broadcaster: /robot_state_publisher
Average rate: 10.410 Hz
Most recent transform: 34.700 (0.029 sec old)
Buffer length: 4.899 sec
Hokuyo_base
head
back_caster
front_caster
left_wheel
right_wheel
Broadcaster: /robot_state_publisher
Average rate: 10000.000 Hz
Most recent transform: 0.000 (34.729 sec old)
Buffer length: 0.000 sec
Broadcaster: /robot_state_publisher
Average rate: 10000.000 Hz
Most recent transform: 0.000 (34.729 sec old)
Buffer length: 0.000 sec
hokuyo
camera
Broadcaster: /robot_state_publisher
Average rate: 10000.000 Hz
Most recent transform: 0.000 (34.729 sec old)
Buffer length: 0.000 sec
rotation_frame
Broadcaster: /robot_state_publisher
Average rate: 10000.000 Hz
Most recent transform: 0.000 (34.729 sec old)
Buffer length: 0.000 sec
camera_rgbd_frame

Fig. 2: Model independently created

3.3 Robot Description

As mentioned, a robot designed in a previous experience will be used for this study. Figure below shows a general arrangement of the implemented robot, comprised of the next elements:

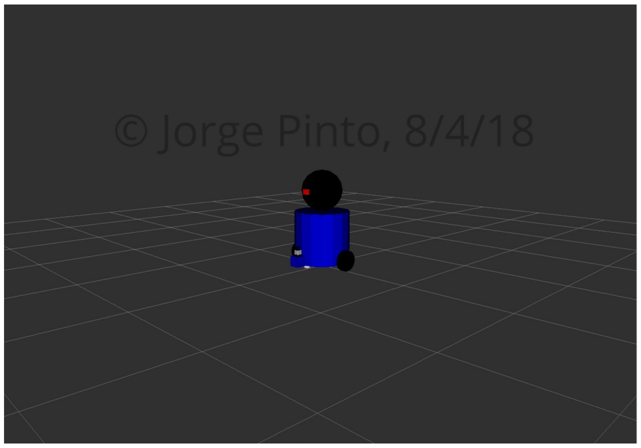


Fig. 3: Model independently created

1) Base: Cylindrical, of radius 0.27 and length 0.5 meters, and mass 10 kgs.
2) Head: Spherical of radius 0.2 meters and 5 kgs of mass.
3) Actuators: Two cylindrical wheels of radius 0.1 and length 0.05 meters, and mass 5kgs each, located at each side of the base.
4) Camera: Located in the front of the head (the nose of the robot), as a 0.05x0.05x0.05 meters box.
5) Hokuyo-sensor: Placed on the bottom-front side of the body, on a 0.1x0.1x0.1 meters box with mass 0.1 kgs.

6) Caster Wheels: Located in both front and back side of the base to avoid continuous tilting over the vertical axis of the robot. Each conceived as a sphere of radius 0.05 meters.

Similarly, these elements were hinged to the base by links defined in the respective .xacro file. The two frame rotations mentioned above were implemented in this file as two links consecutively hinged but initially rotated $-pi/2$ radians over the Z axis, with a second rotation of $pi/2$ radians over the Y axis.

3.4 Worlds to Map

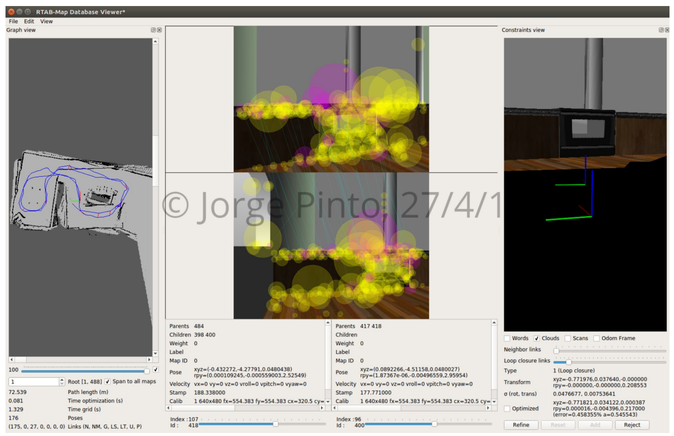
The first environment/world for the robot to map resembles the one portrayed in Fig.4a as a closed kitchen-like area plus a living room side with regular appliances and objects, whereas a customized one can be seen in Fig.4c displaying an open playground with different solids around the main structures.

In both cases, the presence of enough features was required for RTAB-Map to properly detect close loops whenever the robot made a pass. As a matter of fact, creating a map with repeated consecutive features may report false positives at a location where no real loop closure exists. Note also that the robot started at the same exact position (i.e., origin coordinates) in the two scenarios without really caring the position of the surrounding features. Finally, no goal to reach had to be established for the robot to navigate; the mapping itself was performed in real-time while providing keyboard inputs to the vehicle.

4 Results

For both environments, the mapping task was achieved by running the rtabmap_ros package right after loading them in Gazebo®and Rviz ®. Four scripts had to be ran in total: *my_droid_kitchen.launch, mapping.launch, rviz.launch* and *teleop.launch* for the provided world, with the difference of running *my_droid_world.launch* in a separate instance for the custom one.

The results herein portrayed were gathered by running the above scripts on a *Jetson TX2* platform. This allowed the proper generation of the shown points clouds that would otherwise be unachievable via a regular Linux-based virtual machine.



```
100
1            Root [1, 488] ✓ Span to all maps
72.539             Path length (m)
0.081              Time optimization (s)
1.329              Time grid (s)
176                Poses
(175, 0, 27, 0, 0, 0, 0) Links (N, NM, G, LS, LT, U, P)

Parents   484
Children  398 400
Weight    0
Label
Map ID    0
Pose      xyz=(-0.432272,-4.27791,0.0480438)
          rpy=(0.000109245,-0.000559003,2.52549)
Velocity  vx=0 vy=0 vz=0 vroll=0 vpitch=0 vyaw=0
Stamp     188.338000
Calib     1 640x480 fx=554.383 fy=554.383 cx=320.5 cy=
Index :107
Id :    418

Parents   417 418
Children
Weight    0
Label
Map ID    0
Pose      xyz=(0.0892266,-4.51158,0.0480027)
          rpy=(1.87367e-06,-0.00496559,2.95954)
Velocity  vx=0 vy=0 vz=0 vroll=0 vpitch=0 vyaw=0
Stamp     177.771000
Calib     1 640x480 fx=554.383 fy=554.383 cx=320.5 cy=
Index :96
Id :    400

Type            1 (Loop closure)
Transform       xyz=-0.771976,0.037640,-0.000000
                rpy=-0.000000,-0.000000,0.208553
σ (rot, trans)  0.0476677, 0.00753641
Optimized       xyz=-0.771821,0.034122,0.000387
                rpy=0.000016,-0.004396,0.217000
                (error=0.458355% a=0.545543)
```

Fig. 5: Rtabmap-databaseViewer results for the Provided World.

Moreover, the shown results confirmed the fact that a specific bag of words was generated by the *rtabmap_ros* package for each of the studied environments.

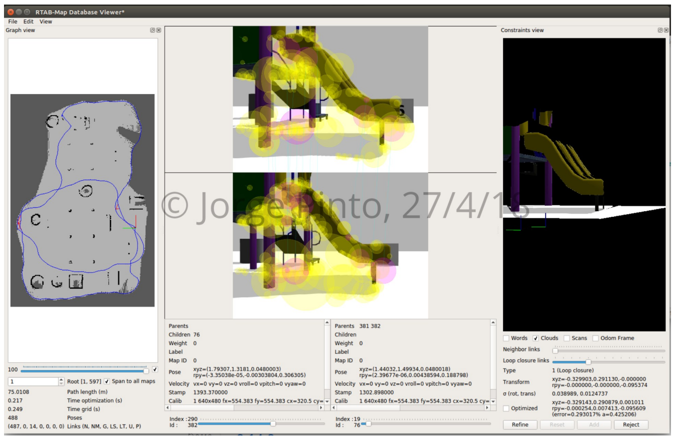
4.1 Provided Environment

The point cloud shown in 4b refers to the map generated for the benchmark world. As observed, several features can be identifiable at a first glance like the kitchen arrangement, chairs, and spatial distribution of most of the items therein contained.

On the other hand, Fig. 5 provides via the *Rtabmap-databaseViewer* application an analysis of how the loop closure process was performed and how many of them could be registered during the execution of the script. In this scenario, *Rtabmap-databaseViewer* registered 27 global loop closures detected from a total of 175 neighbors.

4.2 Customized Environment

Likewise, the point cloud portrayed in Fig.6 displays the general arrangement of features detected by the robot when moving around the playground: slides, solid textures, associated color, etc. In addition, for this world *Rtabmap-databaseViewer* registered a total of 487 neighbors and 14 global loop closures instead.



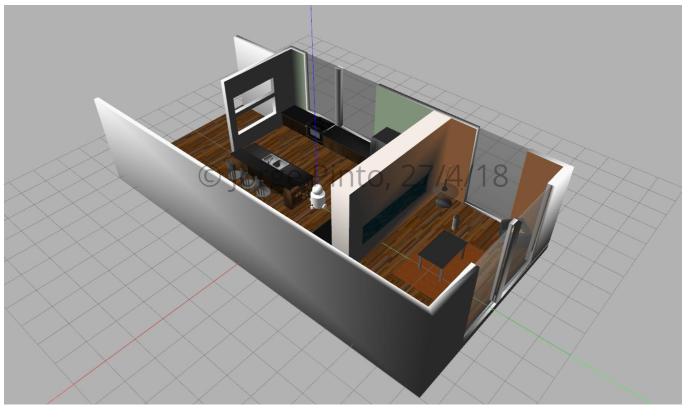
```
100
1            Root [1, 597] ✓ Span to all maps
75.0108            Path length (m)
0.217              Time optimization (s)
0.249              Time grid (s)
488                Poses
(487, 0, 14, 0, 0, 0, 0) Links (N, NM, G, LS, LT, U, P)

Parents
Children  76
Weight    0
Label
Map ID    0
Pose      xyz=(1.79307,1.3181,0.0480003)
          rpy=(-3.35038e-05,-0.00303804,0.306305)
Velocity  vx=0 vy=0 vz=0 vroll=0 vpitch=0 vyaw=0
Stamp     1393.370000
Calib     1 640x480 fx=554.383 fy=554.383 cx=320.5 cy=
Index :290
Id :    382

Parents   381 382
Children
Weight    0
Label
Map ID    0
Pose      xyz=(1.44032,1.49934,0.0480018)
          rpy=(2.39677e-06,0.00438594,0.188798)
Velocity  vx=0 vy=0 vz=0 vroll=0 vpitch=0 vyaw=0
Stamp     1302.898000
Calib     1 640x480 fx=554.383 fy=554.383 cx=320.5 cy=
Index :19
Id :    76

Type            1 (Loop closure)
Transform       xyz=-0.329903,0.291130,-0.000000
                rpy=-0.000000,-0.000000,-0.095374
σ (rot, trans)  0.038989, 0.0124737
Optimized       xyz=-0.329143,0.290879,0.001011
                rpy=-0.000254,0.007413,-0.095609
                (error=0.293017% a=0.425206)
```

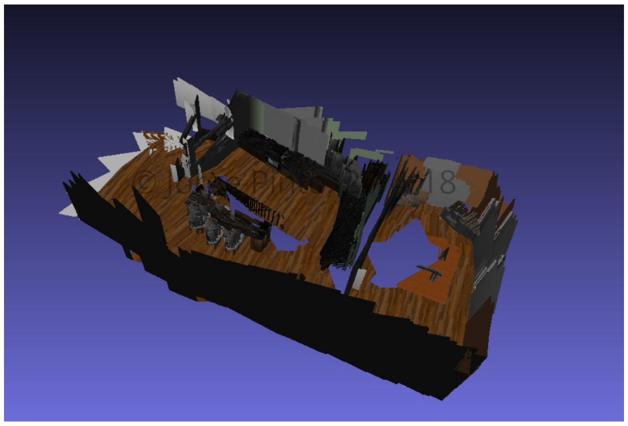
Fig. 6: Rtabmap-databaseViewer results for the Custom World.

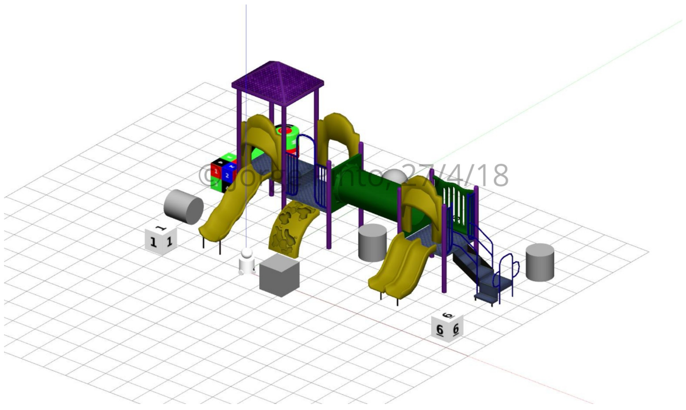
5 Discussion

In general, both worlds had features inherent to them. The kitchen plus living room arrangement had chairs, a fridge, a corner lamp, etc; these were the places where most of the loop closures were registered and this makes a difference in terms of how complex the worlds were and at which points what motivated the package to create the neighboring links. On the other hand, although feature rich in terms of architecture, the custom world did not account for much global loop closures although neighboring links were high in comparison.

A single fact from comparing both experiences proves that the provided world required more robot passes to generate the map than that of the custom world (see blue paths on Figs5 and 6. This creates a luck of compromise that should be taken into account whenever mapping an environment: the more features it has, the more bag of


(a) Provided World.


(b) Point Cloud generated for the Provided World.


(c) Custom World.

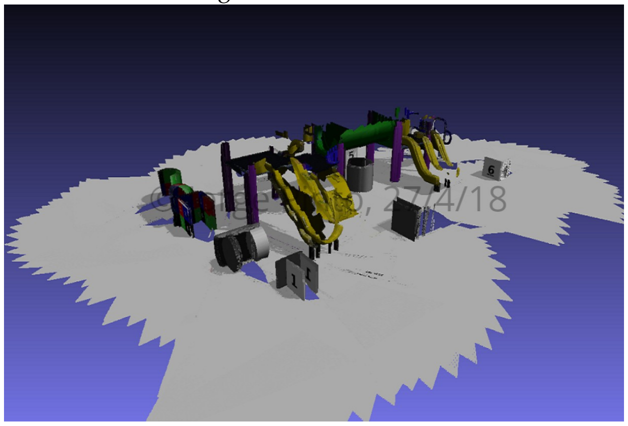

(d) Point Cloud generated for the Custom World.

Fig. 4: Resulting 3D maps generated by the RTAB-Map package.

words would be generated from the passes and thus the much heavier the database will eventually be.

In this sense, optimization of the gathered images will also affect the precision of the generated point clouds. By comparing Figs.4b and 4d it is clear that a more feature rich world will require a higher frequency of image feeding for the package to better perform smoothing actions. Actually all of the items contained in the playground where mapped without the need of too many robot passes, which in turn reduced the amount of images to smooth by the package itself.

At the end, if precise maps are required or not it would depend on the application the robot is designed for. For instance, mapping an empty parking lot would be easier than say a house full of appliances and objects due to the difference in the number of constraints the robot has to account for. In such case, robot's and map requirements have to be properly addressed before any experiment can be deployed in the field.

6 Conclusion / Future work

RTAB-Map algorithm was tested in two scenarios: first on a benchmark provided beforehand, and later on a custom design. In both cases, it was possible to generate a point cloud library that portrayed the majority of the features therein contained. Nonetheless, less loop-closures were detected for the customized scenario due to the lack of much recognizable features as compared to the benchmark. On the other hand, as less features were encountered in the custom design, the image gathering proved to be smoother while moving the robot as it can be inferred from comparing the overlapping set of images in Figs.4b and 4d. Moreover, the generated map was lighter in terms of occupied disk space for the custom than for the benchmark design, although this is directly related to the number of passes performed by the robot.

This project proved that it doesn't matter the type of environment, RTAB-Map is a good choice for 3D mapping generation. In addition, RTAB-Map can be used for localization purposes; in this sense, it would be interesting to compare its results with those of particle filters processes like Adaptive Monte Carlo Localization (AMCL) or Extended Kalman Filters (or EKF). Future work in this direction would give a better understanding of the applications where these rationales are more efficient.

References

[1] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part i," *IEEE robotics & automation magazine*, vol. 13, no. 2, pp. 99–110, 2006.

[2] G. Grisettiyz, C. Stachniss, and W. Burgard, "Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling," in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pp. 2432–2437, IEEE, 2005.

[3] G. Grisetti, G. D. Tipaldi, C. Stachniss, W. Burgard, and D. Nardi, “Fast and accurate slam with rao–blackwellized particle filters,” *Robotics and Autonomous Systems*, vol. 55, no. 1, pp. 30–38, 2007.

[4] G. Grisetti, C. Stachniss, and W. Burgard, “Improved techniques for grid mapping with rao-blackwellized particle filters,” *IEEE transactions on Robotics*, vol. 23, no. 1, pp. 34–46, 2007.

[5] G. Grisetti, R. Kummerle, C. Stachniss, and W. Burgard, “A tutorial on graph-based slam,” *IEEE Intelligent Transportation Systems Magazine*, vol. 2, no. 4, pp. 31–43, 2010.

[6] M. Labbé and F. Michaud, “Long-term online multi-session graph-based splam with memory management,” *Autonomous Robots*, Nov 2017.

[7] D. Filliat, “A visual bag of words method for interactive qualitative localization and mapping,” in *Robotics and Automation, 2007 IEEE International Conference on*, pp. 3921–3926, IEEE, 2007.