Mapping, Localization and Navigation Improvements by Using Manipulated Floor Plan and ROS-Based Mobile Robot Parameters

Petri Oksa, Pekka Loula and Erkki Castrén
Telecommunications Research Center
Tampere University of Technology (TUT), Pori Department, Pohjoisranta 11A, FI-28100 Pori FINLAND
petri.oksa@tut.fi

Abstract— **Indoor environment mapping is one of the obvious benefit that can be assigned for service mobile robots. With an extensive set of parameters provided by Robot Operating System (ROS), the robotic system user can set core tasks to improve the quality and usability of the received environmental grid map. This has an evident impact for the robot teleoperating, map scale, and significantly, in avoiding discontinuities and inaccuracy objects of the generated map—that is to say the more accurate the map is—the more straightforward the autonomous navigation is. The use of an original floor plan instead of map obtained by the robot is one worthwhile option. Floor plans can be manipulated by using a generic image processing software in that case where the received map has unidentifiable areas due to the robot's 3D sensor unable to detect. This paper presents ROS-based indoor environment mapping, localization and autonomous navigation factors in the Open Cloud Robotic Platform (OpenCRP) ecosystem, a cloud robotics project upon an open-source basis, experimented with a TurtleBot II mobile robot. Moreover, a method for manipulating the floor plan is presented in this work.**

Keywords— mapping; localization; autonomous navigation; mobile robot; ROS

I. INTRODUCTION

Outside of manufacturing and wholesale warehouses, robots are hard to find. They are through its history considered mainly as static and monotonous job performers. Out-of-this-box thinking, current mobile robots could do many more pre-programmed tasks, such as lawn mowing, vacuum-, window- and swimming pool cleaning. In particular, mobile robots are ideal for exploring environments and monitoring situations that are dangerous, strenuous or too boring to humans. They are expected to perform long range, long term and complex missions. These robots are typically remote controlled (teleoperated) and require one or more human operators [1], [2], [3].

Cloud robotics projects such as DAvinCi [4] and RoboEarth [5] have paved the way for global system accessibility, scalability and parallelism advantages of cloud communication environment. One of the main purposes of these projects is to remain robots lightweight and offload heavy computation into the cloud. In addition, common to these projects is to network cooperative ROS[1] robots, provide secured customizable computing in large environments using the cloud as a medium for establishing a network between robots sensors and mobile devices.

OpenCRP[2] is an open-source cloud robotics ecosystem based on a service-oriented PaaS architecture using Ubuntu Linux operating system. Its multi-robot API allows any ROS robot to share their collected data via the cloud. The ecosystem's cloud environment is implemented through Apache Hadoop software framework and configured as a HDFS-cluster (Hadoop Distribution File System). HDFS-cluster operates as data storage for large data sets and could be managed from remote computers with terminal commands. Figure 1 shows a use case architecture of the ecosystem for the experiments.

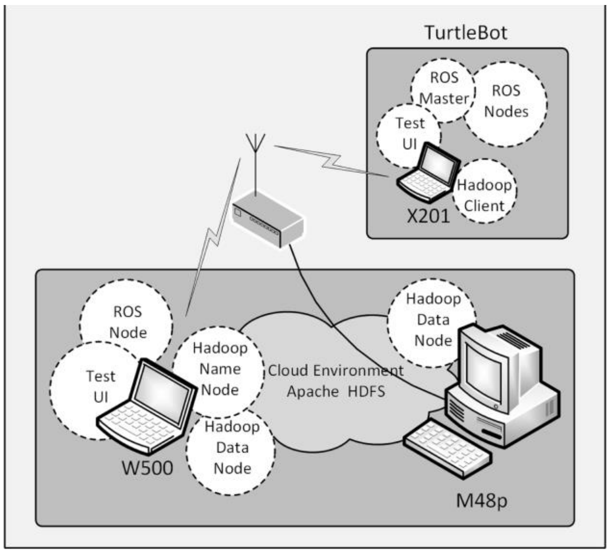

Fig. 1. OpenCRP use case architecture

There are several mobile robots that can be used with ROS. One of the most popular robot is TurtleBot II, a developmental kit for researchers, hobbyists and software developers see Figure 2 and Table 1 for technical specification. A wheeled TurtleBot II

[1] Please refer to http://www.ros.org for a complete description of ROS
[2] www.tut.fi/opencrp

integrates in several simple and low-cost state-of-the-art hardware devices such as Microsoft's Kinect Xbox 360 sensor, Yujin Robot's Kobuki, and iRobot's Create. It is a moving platform intended for indoor environments and with 3D perception capabilities by means of the Kinect and Asus Xtion Pro Live [6]. TurtleBot II is the evolution of the TurtleBot platform, the cheapest robot in the market with ROS architecture. TurtleBot II (Kobuki base) is a new version of TurtleBot platform and it has features over its predecessor: odometric measurement precision, open protocol, higher speed and mobility, bigger diameter wheels and capacity to overcome obstacles up to 12 mm [13].

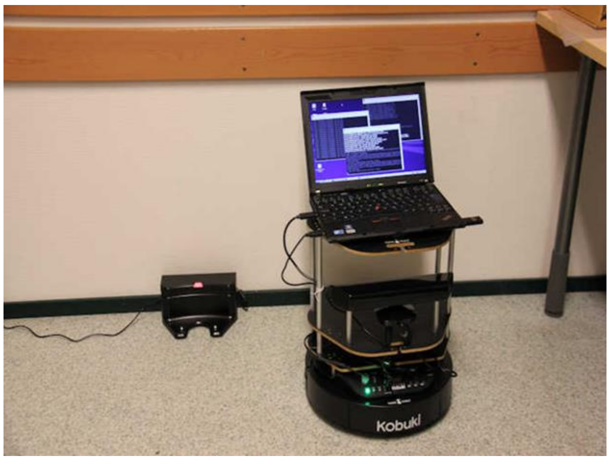
Fig. 2. TurtleBot II robot

Few works address mapping question at the mission level. Authors in [14] present a ROS-Based control system with a Pioneer 3-DX robot for indoor mapping, localization and autonomous navigation. Some factors associated with indoor environments that can affect mapping, localization and automatic navigation, are also presented. Authors investigate in [15] the suitability of the Xbox Kinect sensor for navigation and simultaneous localization and mapping. A prototype using the Kinect to capture 3D point cloud data of the external environment is also presented. The data is used in a 3D SLAM to create 3D models of the environment and localize the robot in the environment. By projecting the 3D point cloud into a 2D plane, the Kinect sensor data for a 2D SLAM algorithm is used and compared the performance of Kinect-based 2D and 3D SLAM algorithm with traditional solutions.

Multi-sensor navigation of intelligent ROS-based wheelchair is presented in [16] for elderly and disabled people. The system enables capabilities of autonomous navigation, inter-operation by multiple modality, interaction and collaboration with the wheelchair. An autonomous navigation for the wheelchair is implemented enabling user point a goal and arrive the goal automatically in a static environment.

In this paper, improvements of mapping, localization and autonomous navigation is presented by using TurtleBot II mobile robot. For experiments, a part of the floor of the building is used for mapping experiments and factors to enhance the received 2D grid map are presented. The rest of the paper is organized as follows. Section II show up issues occurred in a dynamic environmental mapping and autonomous navigation with a received grid map. In Section III, experimental results are given to demonstrate the feasibility and development by the ROS-based (ROS Indigo platform) use case configurations. Finally, a conclusion and summary of the whole work of this paper are discussed in Section IV.

II. Robot System

A. *ROS Parameters*

ROS Parameter Server is a shared, multivariable dictionary that is accessible via network APIs. Nodes use this server to store and retrieve parameters at runtime. The Parameter Server is implemented using XML-RPC and runs inside the ROS Master, which means that its API is accessible via normal XMLRPC libraries. The Parameter Server is used to store data that is accessible by all the nodes. ROS has a tool to manage the Parameter Server called *rosparam* [6].

The robot will move through the map using two types of navigation—global and local. The global navigation is used to create paths for a goal in the map or a far-off distance. The local navigation is used to create paths in the nearby distances and avoid obstacles, for example, a square window of 4 x 4 meters around the robot. The *costmaps* have parameters to configure the behaviors, and they have common parameters as well, which are configured in a shared file. Configuration basically consists of three files where different parameters can be set up [6], [7]. These files are as follows:

– costmap_common_params.yaml

– global_costmap_params.yaml

– local_costmap_params.yaml

Parameters can be accessed via command line, code or launch file [8]. They are named using the normal ROS naming convention. This means that ROS parameters have a hierarchy that matches the namespaces used for topics and nodes. This hierarchy is meant to protect parameter names from colliding. The hierarchical scheme also allows parameters to be accessed individually or as a tree [7]. The rosparam command-line tool enables user to query and set parameters on the Parameter Server using *YAML* (YAML Ain't Markup Language) syntax [9].

YAML-encoded file contains an experimental library for using YAML with the Parameter Server. This library is intended for internal use only. rosparam can be invoked within a *roslaunch* file [10].

B. *OpenSLAM*

Simultaneous Localization and mapping (SLAM) is an algorithm for robot to autonomously explore and map its environment with its sensors while localizing itself at the same time [15]. The preliminary challenge in SLAM is how mobile robots can autonomously learn maps of their environments. The basic difficulty is that the robot must know exactly where it is, so that it can update the right part of the map: relying on dead-reckoning alone (i.e. integrating the motor commands) is unreliable because of noise in the actuators (slippage and drift). One solution is *GMapping* from *OpenSLAM*, Rao-Blackwellized particle filter (RBPF) to learn grid maps from laser range data. This approach uses a particle filter in which each particle carries an individual map of the environment. A key question is how to reduce the number of particles. The RBPF provides adaptive techniques to reduce the number of particles in a Rao-Blackwellized particle filter for learning grid maps. In addition, an approach to compute an accurate proposal distribution taking into account not only the movement of the robot but also the most recent observation, is presented. This drastically decreases the uncertainty about the robot’s pose in the prediction step of the filter. Furthermore, RBPF applies an approach to selectively carry out re-sampling operations, which reduces the problem of particle depletion [11], [12].

C. *Creating a map with TurtleBot and ROS*

ROS has a tool that builds a map using the odometry and a laser sensor. This tool is a ROS map-server (slam_gmapping), a tool enabling to build a map using the robot’s odometry and 3D sensor data. *amcl* (Adaptive Monte Carlo Localization) is a probabilistic localization system for a robot moving in 2D. It implements the adaptive Monte Carlo localization approach, which uses a particle filter to track the pose of a robot against a known map [6].

To navigate the environment, the gmapping algorithm is used. The gmapping package provides laser-based SLAM to create a 2D occupancy grid map (like a building floor plan) from laser and pose data by TurtleBot. TurtleBot provides odometry data, which is needed for slam_gmapping. The slam_gmapping node will attempt to transform each incoming scan into the odom (odometry) *tf* frame. tf is a package that lets the user keep track of multiple coordinate frames over time. tf maintains the relationship between coordinate frames in a tree structure buffered in time, and lets the user transform points, vectors, etc. between any two coordinate frames at any desired point in time [17].

D. *Mapping Issues*

In OpenCRP, the 2D SLAM algorithm is performed using an ROS implementation of GMapping from OpenSLAM [18]. One of the biggest drawbacks for a proper mapping is indoor surroundings where the corridors are long or where there are not enough identifiable objects for scanning operation. If the corridors are long with no clearly identifiable objects such as chairs, vertical pillars, or tables, the scanned map is inapplicable almost in every case. Another issue is the case where objects in the environment are moved to different place, for instance someone have moved the chair, causing the robot lose its location on the map. In this case, the robot performs two rotate recoveries to find its location on the map until it aborts and stops the ROS master because the path plan cannot be created.

Aforementioned drawbacks result unwanted wall line discontinuation, diagonals or many gray regions in the generated map. Grey regions indicate that there is no data available for these regions. This is one major issue due to amcl mapping algorithm causing the experiment complicated.

The robot is moved in the desired environment via joystick. Joystick is connected to the on-board laptop running ROS with an USB dongle. Controlling the velocity by joystick is quite difficult, therefore it is much convenient to modify the myjoystick.launch file to attain an optimal robot control. Figure 3a shows discontinuation of the generated grid map at the velocity of 1.5 rad/sec (rotational) and 0.5 m/s (translational) highlighted in red circle. In Figure 3b, the velocity is set down to 1.0 rad/s and 0.2 m/s. The velocity can be altered by changing two parameter values in *myjoystick.launch* file:

```
<param  name="scale_angular"  value="1.5"
/>
<param  name="scale_linear"  value="0.5"
/>
```

As can be seen in the Figure 3, slowing down the velocity has a visible difference for the map quality. The mapped environment structure is a closed loop, where the robot starts moving from a point and comes back at that point through different way. To overcome the mapping issues presented here, a solution is described in Sectlion III. A laptop was mounted onto the TurtleBot robot. TurtleBot has either Kinect or Xtion sensor used in the experiments. Both sensors are tested in experiments in order to find the optimal parameters for mapping. Relevant mapping functions for ROS-based robot are then clarified. A good reminder for the mapping is to notice both ROS-parameters and min/max parameter values changeable through the 3D sensor.

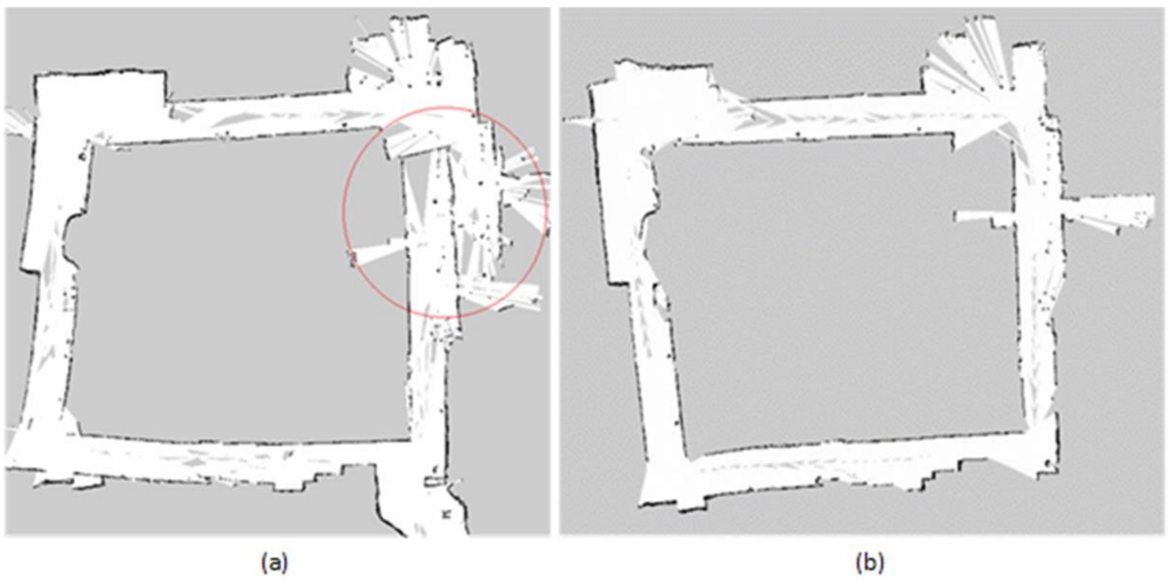

Fig. 3. Discontinuation of the generated grid map at the velocity of (a) 1.5 rad/s and 0.5 m/s, and (b) 1.0 rad/s and 0.2 m/s

III. EXPERIMENTAL RESULTS

To validate the presented system and to identify the systems functionality and performance, an experimental use case has been implemented. The experiment entails TurtleBot II mobile robot, MS Kinect and Xtion Pro Live sensors, and laptop mounted onto the TurtleBot. The goal is to improve the map using certain ROS parameters and to make assessment on where the system needs to be improved upon.

A. 3D Sensors

Both of 3D sensors used in experiments, Asus Xtion Pro Live and Microsoft Kinect for Xbox, use laser point cloud for distance measuring. The point cloud is projected into the environment, which is going to be mapped. The pattern is then received by the IR CMOS sensor. As a default setting, a 10-point slice is used and nearest vertical measuring points is projected into a same plane. This produces an emulated laser scanning line.

In OpenCRP, the most significant features are minimum- and maximum measurement ranges. If the minimum range is high, object in nearby will not be observed. Whereas the maximum range is small, localization suffers as the AMCL-algorithm gets less sensor data, which can be compared to a base map. Both ranges were measured by placing the robot perpendicularly towards the measuring surface. *rviz* was used to visualize the laser scanning.

A minimum measurement range was taken from the leading edge of sensor measured to a three different wall surfaces. Then the smallest distance where the laser scanning line is still continuous in the middle of the view, were searched, because the measurement result disappears first when the distance begin to decrease.

Distances were measured by the measuring tape and results are rounded down to a nearest centimeter, because the determining of a continuous line from a rippling laser scan line was not so unambiguous. Table 2 shows the minimum measurement ranges for different materials. Mat surfaced brick wall gave the best results for both sensors. The results show that dim surfaces have better features than glossy. To get unambiguous reason for this behavior is difficult to analyze, because the exact operating principle is not released.

TABLE I. TURTLEBOT II TECHNICAL SPECIfiCATION

Max. translational velocity	65 cm/s
Max. rotational velocity	3.14 rad/s
Payload	5 kg (hard floor), 4 kg (carpet)
Cliff	will not drive off a cliff with a depth >5cm
Threshold climbing	climbs thresholds of 12mm or lower
Rug climbing	climbs rugs of 12mm or lower
Expected operating time	3/7 hours (small/large battery)
Expected charging time	1.5/2.6 hours (small/large battery)
Docking	can perform docking within a 2m x 5m area in front of the docking station
PC connection	USB or via RX/TX pins on the parallel port
Motor overload detection	disables power to motors on detecting high current
Odometry	25718.16 ticks/revolution, 11.7 ticks/mm
Gyro	factory calibrated, 1 axis (100 deg/s)
Bumpers	left, center, right
Cliff sensors	left, center, right
Wheel drop sensor	left, right
Power connectors	5V/1A, 12V/1.5A, 12V/5A
Docking recharging connector	19V/2.1A- expansion pins: 3.3V/1A, 5V/1A, 4 x analog in, 4 x digital in, 4 x digital out
Audio	several programmable beep

	sequences
Programmable LED	2 x two-coloured LED
State LED	1 x two-coloured LED [blinking-charging, Green-high level, Orange-low level]
Buttons	3 x touch buttons
Battery	14.8V lithium-Ion 2200 mAh (small) 4400 mAh (large)
Firmware upgradeable	via USB
Sensor Data Rate	50 Hz
Recharging Adapter	Input: 100-240V AC, 50/60 Hz, 1.5A max; Output: 19V DC, 3.16A
Netbook recharging connector (only enabled when robot is recharging)	19V/2.1A DC
Docking IR Receiver	left, centre, right

TABLE II. MINIMUM MEASUREMENT DISTANCES FOR DIFFERENT MATERIALS

Wall material	Kinect Xbox 360/min. range (cm)	Xtion Pro Live/min. range (cm)
White fiberglass wallpaper	54.5	55.5
Grey sheet metal locker	60.5	63.5
White door	65.0	77.5
Red brick wall	50.0 (from the surface of the brick)	49.0 (from the surface of the brick)

The minimum measuring range using Kinect was 0.50–0.65 meters, which is better than the manufacturer have informed, 0.80 meters [19]. It is specified in [20], that the OpenKinect software limits the minimum measuring range to 0.50 meters, which was also the minimum value achieved in the measurements.

For Xtion Pro, the manufacturer gives the minimum measurement range of 0.8 meters, same as Kinect [21]. When measuring the range to white door, which was the most challenging material, we got the 0.77 meters range, being quite close to what is stated in the technical specification. However, depending on the surface, shorter ranges is also possible to measure, as can be noticed in the red brick wall result of 0.49 meters.

Maximum measurement range was measured only on two different surface materials, because such a long measurement sites for other surface materials was not available. In measurements, the robot was placed on a wheeled table and distances were carried out by moving the table. For each surfaces three limiting distances were measured:

(1) laserline is static as a whole (only small gaps is in view as the distance is gradually increasing in both sensors)

(2) static laserline parts are still separable; sensor data is available

(3) measurement results disappear on a whole width of the measurement point (occasional glimmers were ignored)

Distances were measured with Bosch PLR 15 digital range rangefinder, measuring accuracy of ±3.0mm [22]. Results in Table 3 are given the accuracy of 10 cm, because the specific interpretation is much more challenging when comparing them to minimum range measurements. White door measurement was proved to be the most difficult for both sensors. In Kinect, a varying behavior was detected in such a way, that the static parts of laserline disappeared in some distances and coming back when the distance increases. This kind of spot was in the measurement of white fiberglass wallpaper at 8.6 meters range and 7.0 meters for white door. In Xtion Pro sensor, such as this kind of zones was not detected, but the behavior changes consistently with relation to the distance.

Kinect's maximum measuring distance is 4.0 meters according to the manufacturer [19]. As the measurement for white door show, the laserline is still static at 5.7 meters distance, being obviously better than manufacturer have specified. For Xtion Pro the manufacturer give the maximum measuring distance of 3.5 meters [21]. Like Kinect, Xtion Pro's maximum measuring distance is better than manufacturer have specified as can be seen from results; laserline is still stable at the distance of 5.8 meters.

TABLE III. MAXIMUM MEASUREMENT DISTANCES FOR DIFFERENT MATERIALS

Wall material	Kinect Xbox 360/min. range (cm)	Xtion Pro Live/min. range (cm)
White fiberglass wallpaper	6.1, 9.3, 9.7	7.1, 9.0, 10,1
White door	White fiberglass wallpaper	White fiberglass wallpaper

B. *Mapping Remarks and Solution*

A primary solution for successful mapping became obvious after mapping the same environment several times and comparing the results with each other. The best results can be achieved when driving the robot in a slow speed, turning it 360 degrees after few meters scanning and rotating it at the corner of the wall, so the laser sensor can observe both walls. By using this method, the robot observes both walls at the same time and it obviously improves the corner shape and wall line straightness, see Figure 4 for the mapping result. It is important that the robot get distance data all the time when it is moving forward. However, this is not comprehensive method because of the mapping algorithm fixes the corners while walls lengths being incorrect due to the dimensional error.

Especially remarkable discontinuation error is caused by a long corridor with no reference points when the algorithm try to fix data received from odometry in a way that the corridors become shorter than they really are. When driving the robot forward along the corridor, the distance from sidewalls do not change which interferes the mapping. In a straight corridor, it is advisable to drive the robot slantingly one after another wall so that the changing distance data is available all the time.

By using the methods above it is possible to get proper mapping results, but they will not ensure a relevant map in every environment scanning. Mapping has such feature that if the same data is available, the generated map is not similar after each scan. The variation of map shape is understandable when taking into account that the algorithm is based on a probability function. This issue is discussed in more detail in [23].

When mapping the environment by the robot, obstacles on the map will be removed only, if measurement points can be received from behind another obstacle. This means that the oncoming obstacle in a long corridor can cause a long row of obstacles if data behind its measurement point is not possible to get.

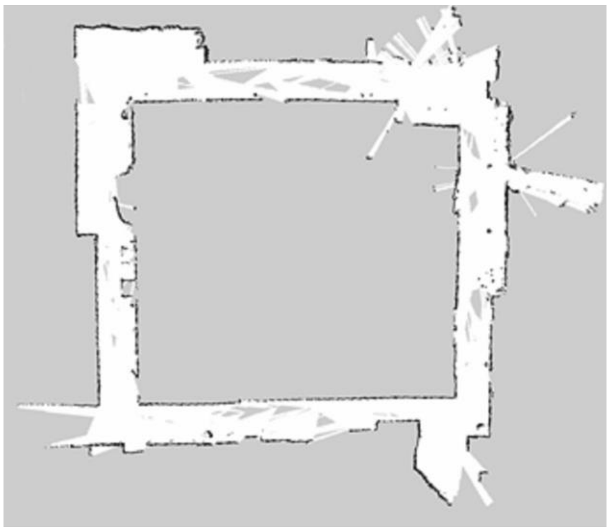

Fig. 4. Rotating at the corners of walls

Parameters for marking the obstacles and removing them are in turtlebot_navigation package's gmapping.launch file:

```
$ maxUrange //(default 6.0m) the
  maximum range of the sensor
$ maxRange //(default 8.0m) the
  maximum usable range of the laser
```

In a mapping procedure, obstacles within the range *maxUrange* are marked onto the generated map. Regions in further, but within a range of *maxRange* parameter, will be cleared and then appeared as a free space in a map between the area of robot and maxUrange. Reasonable explanation for the area between maxUrange and maxRange parameter can be found. Obstacle locations on the generated map should be accurate enough and therefore only regions the most accurate laser sensor measuring range will be used. When the data is available outside of the accurate measurement region, sufficient obstacle location and distance is unobtainable, but with the existence of usable measurement region can be concluded that the area is free from obstacles in measurement direction. Thus, also the inaccurate measurement data is available for the improvement of a grid map.

C. Floor Plan Manipulation

One worthwhile option for preferable navigation is the use of a floor plan. For the experiments, a PDF-file from the building's floor plan was used as a map to be manipulated. All the mappings were performed in the second floor of the building. This mapped environment consists of corridors around the telecommunications lab. All other surrounding areas were separated off from the map except the lab and corridors as illustrated in Figure 5.

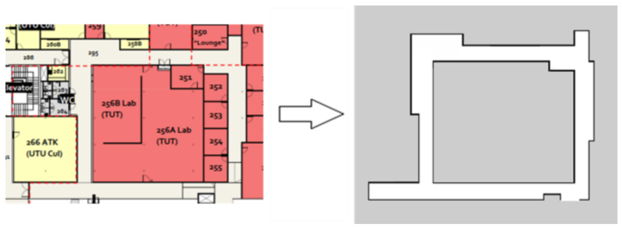

Fig. 5. Manipulated floor plan

A resulted PGM file format on the right hand side is a base map file that TurtleBot uses for navigate the environment. For manipulation of the generated 2D grid map, GIMP (GNU Image Manipulation Program) is used [24]. The map must be in scale and therefore the map generated by TurtleBot was imported to its own layer. Then manipulated floor plan was scaled together with the robot-generated map. After that, the resulting PGM (Portable Greymap) map is then saved to the robot's laptop and a new YAML file is saved utilizing the YAML file used for scaling before. Lastly, a YAML file name should be changed and edit the image file similar as done with the floor plan. In our use case the modified YAML file is:

```
$ image: /home/user/maps/
  pohjapiirros.pgm
```

The manipulated floor plan was tested by driving the robot through the environment area. The test proved that the floor plan was in same scale as the robot-generated map. One option to scaling is to map a part of the environment with the robot and use this map for scaling the manipulated floor plan. YAML file consist of a resolution parameter, which can be used to define distance in meters per pixel. This gives the possibility to scale by using pixels of the map, being useful for instance in the case where the length of building corridor is known. The GIMP, for example, can then be used to measure distances in pixels. That means it is not necessary to perform the environmental mapping with robot at all.

D. Navigation

In ROS navigation, several map layers are used at the same time. In addition to a static map layer, the dynamic global costmap is maintained, in which the robot-collected obstacle information is inserted, those which are not updated into the local costmap. Paths are calculated in a larger scale according to the global costmap and during the robot teleoperation local cost-map is used, by which the local path is calculated in nearby the robot along dynamically changing environment. In practice, by means of the information got from a local costmap, the obstacles in the robot's way are avoided.

For obstacle avoidance, costmaps have parameters turtlebot_navigation package's *costmap_common_params.yaml* folder:

```
$ obstacle_range //(default 2.5)
  maximum range for obstacle insert
$ raytrace_range //(default 3.0) range
  for costmap clearing
```

The obstacle_range parameter determines the maximum range sensor reading that will result in an obstacle being put into the costmap. The *raytrace_range* parameter is the maximum range in meters at which to raytrace out obstacles from the map using sensor data. Setting it to 3.0 meters as above means that the robot will attempt to clear out space in front of it up to 3.0 meters away given a sensor reading [25], [26].

The environment that is going to be mapped can consist of complicated structures like narrow passes, edges or zigzag ways, for example. One such a structure is an automatic sliding door. In our experiments, we had a long corridor with an automatic sliding door at the end of it. Its response distance was different when coming in to the corridor than going out of it. The door opened nicely when the robot approaches it in one direction but from coming opposite direction robot stops when detecting it before the door opens.

Solution for the problem was virtual wall added into the 2D grid map. That changes the robot's path in such a way that the door have enough time to open. As visualized in Figure 6, the robot is forced to go around the virtual wall so that the door's opening mechanism can detect the robot in time. Obstacle clearing from behind the received sensor reading is not considered when adding the virtual wall, because the static map is not updated with this sensor data. This was experimented by driving the robot beside to virtual wall measuring direction pointed through it towards the real wall. Measuring results from the real wall was visualized by rviz's local costmap view, where the measurements are clearly displayed. This test proved that it is not unable to produce a path through the virtual wall.

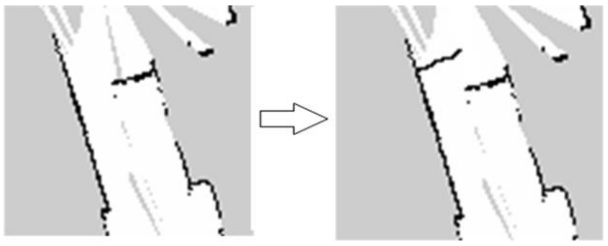
Fig. 6. Modified grid map

IV. Conclusion and Discussion

The presented work describes a ROS-based system for improving mapping, localization and navigation. Parameters for controlling the TurtleBot II robot in indoor environment are experimented. Experiments give practical results on how the mapping can be improved.

Minimum measuring range results were fairly similar for both sensors. The blind spot was the main goal for the sensor experiments. The length of a blind spot was rounded down to a same tolerance along with measurements $(\pm 0.25\mathrm{cm})$. Kinect's blind spot varied between $24.2{-}39.2 \approx 24.0{-}39.0\mathrm{cm}$ and Xtion Pro's between $22.8{-}51.3 \approx 23.0{-}51.5\mathrm{cm}$.

The blind spot substantially impairs the robot's ability to perform its tasks in a dynamic environment. The robot should have the ability to make observations immediately after its physical dimensions in order to avoid obstacles nearby. The results of minimum distance range cannot therefore consider adequate.

The difference in maximum measuring results between sensors was bigger than minimum measuring results. Maximum distances on a whole length of static laserline varied with Kinect between $5.7{-}6.1\mathrm{m}$ and with Xtion Pro between $5.8{-}7.1\mathrm{m}$. Default values of maxUrange and maxRange parameters for both sensors were more suitable to white fiberglass wallpaper than to results with white door, which was challenging to measure. maxUrange parameter default value is `6.0 meters`, which means that obstacles within the range are inserted onto the generated map, is close to results with white door for both sensors. Similarity can be seen when comparing results with maxRange default parameter value of `8.0 meters`. When measuring towards a white door, Kinect's maximum range was `7.9` and Xtion Pro's `7.8 meters`, in which case the laserline has still static parts. maxRange parameter indicates the maximum distance from which results enabling the clearance the area between the robot and maxRange. Default parameter values for TurtleBot are quite suitable according to the experiment results.

Kinect managed a little bit slightly between the area of maxUrange and maxRange having an area around at `7.0 meters`, where the static measurement was unable to get. Xtion Pro was found better in navigation than Kinect in a practical manner. Maps obtained with Kinect included more unknown gray regions, which inaccurate zones between `6.0` and `8.0 meters` will partly explain.

Obtaining a proper map is challenging with both sensors. A short maximum observation range and limited scanning beam lead to situation, where the robot have to turn around to scan the whole area again. In small rooms this problem not exist, but larger spaces and especially long closed paths will most likely cause difficulties.

When using TurtleBot equipped with Kinect- or Xtion Pro-like sensor, the following circumstances are good to notice:

– The velocity should be moderate; experiments showed that suitable rotational speed is `1.0 rad/s` and translational `0.2 m/s`

– Robot must have measuring data related to distance all the time when it is moving forward. This can be done by driving the robot slantingly one after another wall

– In corners the best way is to stop the robot about a `1.0 meters` distance from the corner and turn the robot 360 degrees so that sensor data is available from both walls at the same time

One worthwhile option for accurate mapping is floor plan manipulation, especially in larger operating areas. Manipulation of the floor plan, like adding new walls or clearing them from map, is easily editable by using image manipulation program. In particular, environments where moving obstacles exists, e.g. patient beds or chairs in hospital's corridor, the floor plan manipulation is very viable option to consider. In that way the robot treats all obstacles as a dynamic obstacles inserting them and clearing them from costmap always when the situation in environment changes during the mapping procedure.

Navigation goes to its purpose when there is enough room around the robot. TurtleBot can calculate the alternative route if the first one is obstructed. Avoiding dynamical obstacles coming to the robot's way come off fine as far as sensor data is available and the distance can be measured. If an obstacle appears inside the minimum distance range, the robot is usually in difficulties.

Very challenging situation is also at dynamical obstacles, which have already left from the area and are inserted on map located nearby the robot. Because the robot localize itself constantly when measuring the environment, also its calculated probable location will fluctuate a bit. When the calculated location is just on the inserted point on map, the robot will not recover from this situation and user have to physically step in to start the ROS master again.

For clearing the dynamical obstacles from costmap depend on significantly the range of sensor. If the sensor range is not enough for making observations from behind the trace left by the obstacle, the trace cannot be cleared.

As mentioned earlier, the floor plan manipulation is a worthwhile option. The robot will not do path planning in such area, which is separated from other area. This is useful method when the operating area delimitation is designed. The virtual wall can also be used for affecting to the robot's path and diverting it to go via a different path. In our experiment, this benefit was utilized by adding a virtual wall inside the response distance of an automatic sliding door sensor to get it opened when the robot approaches the door.

Both navigating and sensors performance are limiting factors. In OpenCRP ecosystem, the robot can be controlled safely in environment where it cannot end up to dynamically narrow passed places. In practice, this means that, for instance in congested corridors, the navigation is uncertain.

Acknowledgment

The authors would like to thank Tampere University of Technology (TUT) Pori Department staff. This work was supported by the European Regional Development Fund (ERDF).

References

[1] Reid, R.; Cann, A.; Meiklejohn, C.; Poli, L.; Boeing, A., Braunl, T. (2013). Co-operative multi robot navigation, exploration, mapping and object detection with ROS. In: IEEE Intelligent Vehicles Symposium (IV), June 23-26, 2013, pp. 1083-1088.

[2] Janssen, R.; van de Molengraft, R.; Bruyninckx, H.; Steinbuch, M. (2016). Cloud based centralized task control for human domain multi-robot operations. Intelligent Service Robotics, 9(1): 63-77.

[3] Jaiem, L.; Druon, S.; Lapierre, L.; Crestani, D. (2016). A Step Toward Mobile Robots Autonomy: Energy Estimation Models. Springer Lecture Notes in Artificial Intelligence, June 21-July 1, 2016, pp. 177-188.

[4] Arumugam, R.; Enti, V. R.; Bingbing, L.; Xiaojun, W.; Baskaran, K.; Kong, F. F.; Senthil K. A.; Meng, K. D.; Kit, G. W. (2010). DAvinCi: A cloud computing framework for service robots. In: IEEE 2010 International Conference on Robotics and Automation (ICRA), May 3-8, 2010, pp. 3084-3089.

[5] Hunziker, D.; Gajamohan, M.; Waibel, M.; D'Andrea, R. (2013). Rapyuta: The roboearth cloud engine. In: IEEE 2013 International Conference on Robotics and Automation (ICRA), May 6-10, 2013, pp. 438-444.

[6] Martinez, A.; Fernández, E. (2013). Learning ROS for robotics programming. Packt Publishing Ltd., UK.

[7] ROS.org, Parameter Server, (2013). [Online]. Available: http://wiki.ros.org/Parameter%20Server [Accessed: 2 September 2018].

[8] O'Kane, J. M.; Kane, J. M. O. A gentle introduction to ROS. O'Kane, 2013.

[9] Ingy döt Net; Ben-Kiki, O.; Evans, C. YAML Ain't Markup Language (YAMLTM) Version 1.2. [Online]. Available: http://yaml.org/spec/1.2/spec.html [Accessed: Accessed: 2 September 2018].

[10] Conley, K. "rosparam", (2014). [Online]. Available: http://wiki.ros.org/rosparam [Accessed: 2 September 2018].

[11] Murphy, K. P. (1999). Bayesian Map Learning in Dynamic Environments. In NIPS, November 1999, pp. 1015-1021.

[12] Grisetti, G.; Stachniss, C.; Burgard, W. OpenSLAM.org. [Online]. Available: http://openslam.org/gmapping.html. [Accessed: 22 September 2017].

[13] Kobuki Turtlebot II. User's Manual, Robotnik Automation, S.L.L., Indigo v1.

[14] Zaman, S.; Slany, W.; Steinbauer, G. (2011). ROS-based mapping, localization and autonomous navigation using a Pioneer 3-DX robot and their relevant issues. In: IEEE 2011 Saudi International Electronics, Communications and Photonics Conference (SIECPC), April 2011, pp. 1-5.

[15] Oliver, A.; Kang, S.; Wünsche, B. C.; MacDonald, B. (2012). Using the Kinect as a navigation sensor for mobile robotics. In: ACM Proceedings of the 27th Conference on Image and Vision Computing New Zealand, November 2012, pp. 509-514.

[16] Li, R.; Oskoei, M. A.; McDonald-Maier, K. D.; Hu, H. (2013). ROS based multi-sensor navigation of intelligent wheelchair. In: IEEE 2013 Fourth International Conference on Emerging Security Technologies (EST), September 2013, pp. 83-88.

[17] ROS.org, tf, (2016). [Online]. Available: http://wiki.ros.org/tf [Accessed: 26 September 2018].

[18] Gerkey, B. Gmapping (2015), Available: http://wiki.ros.org/gmapping [Accessed: 26 September 2018].

[19] Microsoft, "Coordinate Spaces", (2016). [Online]. Available: https://msdn.microsoft.com/en-us/library/hh973078.aspx#Depth_Ranges [Accessed: 7 September 2018].

[20] Andersen, M. R.; Jensen, T.; Lisouski, P.; Mortensen, A. K.; Hansen, M. K.; Gregersen, T.; Ahrendt, P. (2012). Kinect depth sensor evaluation for computer vision applications. Technical Report Electronics and Computer Engineering, 1(6).

[21] ASUSTeK Computer Inc., "Xtion PRO LIVE". [Online]. Available: http://www.asus.com/3D-Sensor/Xtion_PRO_LIVE/specifications [Accessed: 26 September 2018].

[22] Robert Bosch GmbH, "Bosch PLR 15 Digital Laser Measure". [Online]. Available: http://www.bosch-plr15.com/gb/en/technical-data.html [Accessed: 26 September 2018].

[23] Gerkey, B. gmapping: slam_gmapping.cpp Source File. [Online]. Available: http://docs.ros.org/indigo/api/gmapping/html/slam_gmapping_8cpp_source.html [Accessed: 26 September 2018].

[24] Gimp.org, GIMP-GNU Image Manipulation Program, (2016). [Online]. Available: https://www.gimp.org [Accessed: 26 September 2018].

[25] ROS.org, navigation/Tutorials/RobotSetup-ROS Wiki, (2015). [Online]. Available: http://wiki.ros.org/navigation/Tutorials/RobotSetup#Costmap_Configuration_.28local_costmap.29_.26_.28global_costmap.29. [Accessed: 26 September 2018].

[26] Marder-Eppstein, E.; Lu, D. V.; Hershberger, D. "costmap_2d - ROS Wiki", (2015). [Online]. Available: http://wiki.ros.org/costmap_2d. [Accessed: 26 September 2018].