2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
Madrid, Spain, October 1-5, 2018

Human-in-the-loop Augmented Mapping

Abbas Sidaoui, Imad H. Elhajj, and Daniel Asmar

Abstract— **In this paper we develop a real-time human augmented mapping system. This approach replaces the traditional offline post processing of maps by a user-friendly system allowing for online editing capabilities. A wide number of applications that acquire accurate mapping of the environment could benefit from such a solution. The proposed framework consists of two main parts: 2D map building using LIDAR, encoders, and IMU; and a user interface for human map augmentation. The first part is built over Gmapping ROS package, while the second is developed in Unity software. Real-world experiments validated the ability of our system to correct for sensor noise and various mapping errors, thus increasing the accuracy of the obtained maps without additional computational costs.**

I. Introduction

Accurate mapping is an essential need for autonomous mobile robots, due to its vital role in navigation, path planning, exploration, etc. In the past decades, significant progress has been achieved in mapping technologies, as autonomous mobile robots are being widely integrated in different domains. In addition to its important role in autonomous navigation, accurate map construction has a great impact in construction management tasks, where it can replace the traditional method of comparing between computer-aided design (CAD) and laser tape measures performed by a human. The latter method is time consuming and is subject to the fact that a human might always suppose that a space is a collection of perpendicular straight lines, which is not always the case [1]. In Simultaneous Localization and Mapping (SLAM) [2, 3, 4], an autonomous agent produces a map of an environment, while localizing itself inside the map. SLAM maps could be built in two-dimensional space using 2D sensors such as LIDARs and laser range scanners, or in 3D space using 3D scanners, RGB-D sensors, etc. [25].

Traditionally, 2D maps are built in two stages, where the first step consists of using sensory data such as LIDAR, IMU, and encoders to build an acceptable map; the second step is to post-process these maps for more accurate and abstract representation of the environment. Moravec and Elfes [5] introduced one of the most common 2D map representation techniques, known as the Occupancy Grid Maps (OGMs). This technique is widely used due to its low computational cost, reduced complexity, and ease of implementation [6]. Occupancy grid maps are constructed through fusion of sensory range data into a 2D grid, where each cell contains a probability of being occupied by an obstacle. Reduced complexity can be achieved thanks to two assumptions: the first is independency of grid cells, and the second is conditional independency of sensory measurements [7]. Although these assumptions would increase mapping efficiency, they normally lead to “over- or under-confident estimate of the cells’ occupancy probabilities”[7]. A widely used approach to learning grid maps is the Adaptive Rao-Blackwellized particle filters [2]. Other map representation techniques include feature-based mapping [24], elevation maps [8], and OctoMap [9].

Numerous methods have been proposed to increase accuracy of maps: Ataer-Cansizoglu et al. [10] introduced Pinpoint SLAM, a combined 2D and 3D SLAM system that uses an RGB-D sensor. The key idea is the extraction of both 2D and 3D measurements from the frames to perform 3D-to-3D, 2D-to-3D, and 2D-to-2D point correspondences in order to produce more accurate maps [10]. This approach proved to have better accuracy than traditional SLAM techniques, but it still relies on offline post processing to achieve higher accuracy. Fusion of 2D and 3D measurements was also addressed in the work of Ratter and Sammut [11]; they combined data from a laser range finder and RGB-D camera to propose a robust algorithm for map construction. Their map fusion algorithm runs slow on a computer without a powerful GPU. In [12], the authors propose to solve 2D SLAM by fusing data from a laser range finder and omnidirectional camera. Dense RGB-D-Inertial system was proposed in [13] to build a fully dense robust 3D map, even when the robot moves with aggressive maneuvers. A major limitation of this approach is its inability to work properly on low-cost computer systems without a GPU.

Improving map accuracy in 2D SLAM was also addressed in the literature. A major challenge when using a Lidar to perform 2D mapping is detecting obstacles lower or higher than the detection plane of the sensor. Even though Merging data from a Kinect sensor and LIDAR proved its ability to detect portions of these low obstacles when used in Hector SLAM, it failed when applying it in Gmapping [14]. Moreover, this approach couldn’t prevent obtaining skewed areas in the map. In the work of Nihei et al. [25], 2D scans are extracted on different heights of a 3D scan, then the map is updated with the scan of the maximum likelihood probability. Despite the fact that this method is more accurate than traditional 2D mapping in dynamic environments, its computational cost is still high.

Many researchers have investigated the use of human robot interaction (HRI) in assisting robots at the task of mapping to

A. Sidaoui and I. H. Elhajj are with the Maroun Semaan Faculty of Engineering and Architecture, Electrical and Computer Engineering Department, American University of Beirut, 1107 2020, Riad El Solh, Beirut, Lebanon; email: ams108@mail.aub.edu ie05@aub.edu.lb.

D. Asmar is with the Maroun Semaan Faculty of Engineering and Architecture, Mechanical Engineering Department, American University of Beirut, 1107 2020, Riad El Solh, Beirut, Lebanon; email: da20@aub.edu.lb

get more accurate and meaningful results. The term Human Augmented Mapping (HAM) was used for the first time in the work of Topp and christensen [19]. They suggested a system used for service robots where the robot builds a map by following the operator through an indoor environment. The operator can add semantic information to the map by providing labels for some specific locations. The concept of an operator guiding the robot and adding semantic information to the map was also applied in [20], [21]. In [22], the authors proposed an interactive mapping method where the user can correct misalignments between two 3D scans of the same area by applying a virtual force. This method is not user-friendly and does not implement any localization technique; thus it does not build complete maps. Jai et al. [23], presented an interactive GUI for mobile robots where the user can view and edit laser range finder LRF data. However, the resulting data needs to be post processed to produce useful maps.

In this paper, we present a system to edit maps in real-time by including a human in the loop. The proposed framework sends the occupancy grid map to a human through a user interface, then augments the human's edits onto the original map. Our proposed system is implemented using ROS and Unity software. The system was able to correct incomplete obstacle boundaries, delete noisy measurements and unwanted objects (*e.g.*, chairs), augment obstacles that are lower/higher than the LIDAR's detection plane, and correct errors occurring from a jumping pose. Our system can also correct any skewness in a wall, and add obstacles that might be invisible to a LIDAR such as a glass wall. It is believed that integrating this system in existing mapping algorithms would increase a map's accuracy, and would allow the use of low-cost sensors that might not be very accurate; this benefit would come at very little additional computational costs.

The remainder of the paper is structured as follows: Section II presents the methodology behind our work. Section III describes our hardware and software platforms, in addition to detailed implementation of the proposed system. Experiments are presented in Section IV to validate our proposed system, and Section V states our conclusions and the plan for future work.

II. Methodology

This section introduces the methodology used for grid map augmentation. As illustrated in Fig. 1, the method is centralized about a "Merger" unit responsible for merging the initial map produced by the "Master" from one side, and the edited map produced by the user through the User Interface "UI" from the other side. At each time step, the Merger receives two occupancy grids, one automated and one edited by a human, and produces two new merged and augmented grid maps according to an algorithm detailed in this section.

A. At initial time step (t_0)

Grid maps are initialized at this time step. The Master node receives sensory input, such as LIDAR scans and odometry data, to produce an Occupancy grid in the form of 1-D Array G_{t_0} of size *n*: $G_{t_0} = \{g_1, g_2, \cdots, g_i, \cdots, g_n\}$, where every cell g_i contains an occupancy probability P_o between 0 and 100, and -1 value for *undiscovered* cells. If a specific cell has an

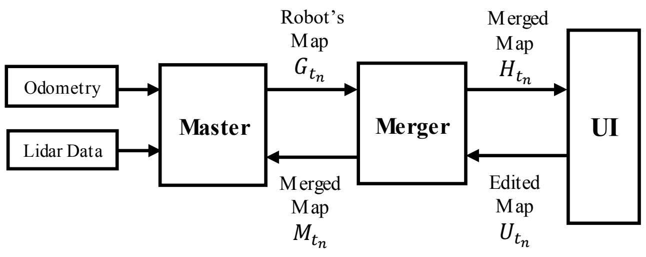

Figure 1. High-Level Methodology representation.

occupancy probability of $P_o = 100$, it means that the probability of this cell being occupied by an obstacle is *P*=1. Whereas $P_o = 0$ implies that there is zero probability for this cell to be occupied. In this work, any cell with $0 \leq P_o < 60$ is considered *free* and any cell with $P_o \geq 60$ is considered *occupied*. Since no human augmented map is received yet, G_{t_0} is copied and produced as:

- Edited map: $U_{t_0} = \{u_1, u_2, \cdots, u_i, \cdots, u_n\}$.
- Merged map to be sent to Master: $M_{t_0} = \{m_1, m_2, \cdots, m_i, \cdots, m_n\} = G_{t_0}$.
- Merged map to be sent to Human (through UI): $H_{t_0} = \{h_1, h_2, \cdots, h_i, \cdots, h_n\} = G_{t_0}$.

B. At $t_n \geq t_0$

The Master node updates merged map $M_{t_{n-1}}$ with the newly received sensory data to produce the most recent robot's map G_{t_n} and send it to the Merger node. Merger node now compares cell values of G_{t_n} and $H_{t_{n-1}}$ to produce a merged map H_{t_n} applying the conditions in Table I. This step is added to allow the Merger to recognize newly discovered *free/occupied* cells, and to replace *undiscovered* statuses with *free* statuses in H_{t_n} for better representation in the graphical user interface. UI receives the merged map H_{t_n}, waits for the user edits, then produce an edited grid map U_{t_n}. The user is allowed to modify any cell's status as follows:

- From *free* to *occupied:* by changing its value from $h_i = 0$ or $h_i = -100$ to $u_i = 100$
- From *occupied* to *free*: by changing its value from $h_i = 100$ to $u_i = -100$. This value is applied such that the Merger unit can distinguish between cells that were initially set free by the robot ($u_i = 0$) and cells forced to be free by the user ($u_i = -100$).

TABLE I. Rules Governing Merging Process

Input		**Output**	
g_{i_t}	u_{i_t} / $h_{i_{t-1}}$	m_{i_t}	h_{i_t}
-1	0	-1	0
	100	100	100
	-100	-1	0
$\geq 0\ OR\ < 60$	0	0	0
	100	100	100
	-100	0	-100
≥ 60	0	100	100
	100	100	100
	-100	0	-100

Cells not modified by the user are set to be $u_i = h_i$. After applying the edits, the new edited grid map $U_{t_n} = \{u_1, u_2, \cdots, u_i, \cdots, u_n\}$ is sent from UI to Merger unit. To produce M_{t_n}, the merging unit compares each cell of G_{t_n} and U_{t_n} in order to decide on output cells according to the rules presented in Table 1. In case U_{t_n} is not sent from UI, U_{t_n} is set to be equal to $U_{t_{n-1}}$. The pipeline for the entire method is shown in Fig. 2.

III. SYSTEM IMPLEMENTATION

A. *Hardware and Software Platforms Used*

The software platform consists of two parts: the first uses Robot Operating System (ROS) [15], installed on Ubuntu 14.04LTS operating system to perform mapping; and the second uses Unity3D installed on Windows 10 OS to develop the user interface. Fig. 3 shows the Husky A200 Explorer Package from Clearpath Robotics that is used as a hardware platform. It includes LMS-511 LIDAR, Point Grey Flea3 camera, high precision quadrature encoders, and a Mini-ITX computer system.

B. *Software Architecture and Implementation*

The software architecture used to implement the proposed method consists of three main parts, as shown in Fig. 4.
The first part implemented the mapping procedure and the second merges maps acquired from the robot with those edited by the user, to produce a new merged map. The third part is the user interface UI, a graphical user interface that allows the operator to acquire maps from the robot during the run, add or remove obstacles (occupied areas) and send back the map to the robot. The first two parts are implemented using ROS packages, while the third one is built in Unity3D software.

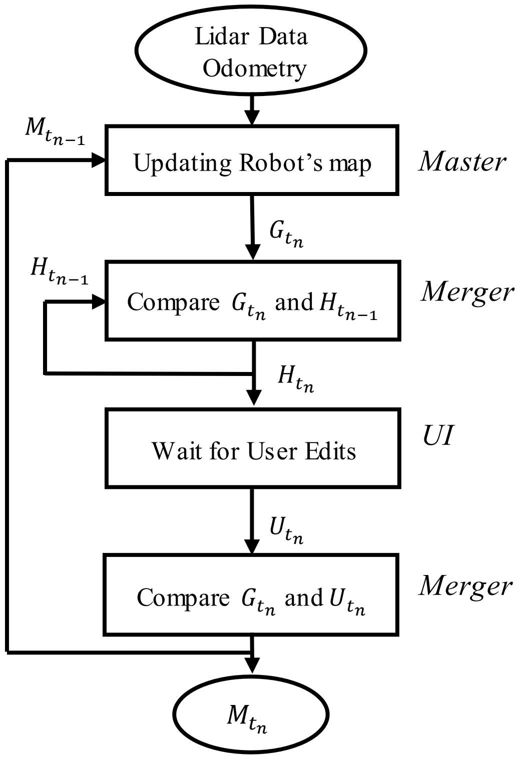

Figure 2. Proposed method's pipeline

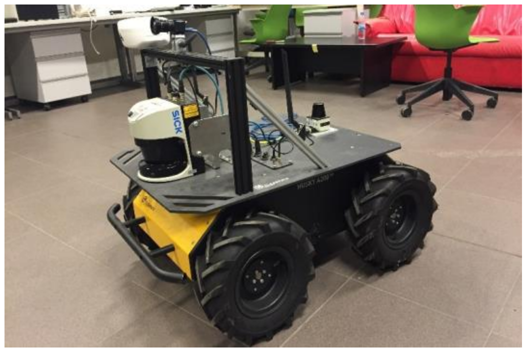

Figure 3. Husky A200 Explorer Package.

1) *2D Mapping*: The Mapping pipeline adopted in this work uses wheel odometry for pose estimation and scans from 2D Lidar to build the occupancy grid. The 2D mapping framework builds on top of Gmapping, a ROS package that implements OpenSLAM's Gmapping [16]. Gmapping is an adaptive technique that applies Rao-Blackwellized particle filter to build grid maps from raw laser range data, odometry, and scan matching-process [2]. Rao-Blackwellized particle filter applied in this work uses a set of 30 particles to estimate the accurate map. Each particle uses odometry measurements observations $z_{1:t}$, and the previous grid map $M_{t_{n-1}}$ to build its own map m_t. These particles are filtered using the Sampling Importance Resampling (SIR) filter [2]. Our 2D mapping package publishes the map with highest probability $G_{t_n} = m_t$ through OccupancyGrid.msg on */map* topic every 0.5 seconds. OccupancyGrid is a compact ROS message that contains:

- "header": information about the map's sequence number, time stamp on which the map is published, and frame_id – "map" in our case.
- "info": most important information presented here are resolution of map in meters/cell, and width and height of the 2D-grid map.
- "data": contains the 1D-array of the grid map G_{t_n}.

2) *Merger*: This part consists of a ROS node named "Edited_Map_Publisher" that subscribes to */map* topic from the 2D Mapping package and */Edited_map* topic from the UI part. It merges the received maps as stated in Section III and publishes OccupancyGrid.msg containing M_{t_n}on */map* topic and H_{t_n} on */map_to_Unity* topic.

3) *User Interface (UI)*: The UI is built in Unity software version 5.4.2f2 on top of unityROS source code [17]. UI receives and sends ROS messages through *Rosbridge_websocket* using ROSBridgeLib, a Unity library that allows communication with ROS using RosBridge Protocol [18] . RosBridge Protocol allows clients to publish or subscribe to any ROS topic through sending JSON messages. Our UI subscribes to */map_to_unity* topic and

publishes on */Edited_map* topic. Fig. 5 explains the UI used in editing the maps:

- Acquiring map: when "Acquire map" button is clicked, maps are acquired through */map_to_unity* as ROS messages and converted to JSON messages using *Rosbridge_websocket*. *Editing system* receives the 1-D array and converts it into a 2D-array using the width and height information presented in the OccupancyGrid message.
- Graphical Representation: the initial tile grid size in GUI is set to be 10meters×10meters, with resolution equal to that of the acquired map. Starting from the lower left corner of the tile grid, a black tile is placed in every cell with *occupied* status. User can zoom in and out using zoom buttons, and navigate in all directions using Arrow keys.
- Editing: the user can place a tile in any *free* cell by clicking on the tile icon, then choosing the cells to place tiles in. User can press the "Esc" button then click on *occupied* cells to delete tiles from them thus converting their statuses to *free*.
- Sending map: when "Send map" button is pressed, the 2D-array is updated with the user's edits and converted to a 1D-array of size *n=width×height* in Editing *System*. This array is then converted to ROS message and sent to *Merger* through */Edited_map*

IV. Experiments

In order to evaluate the proposed system performance, we conducted experiments, taking into account different scenarios that affect the precision of maps created by the traditional mapping procedure. Experiments were carried out in the Vision and Robotics Lab (VRL) at the American University of Beirut using the Clearpath Husky A200 Explorer package. All ROS packages run on a Lenovo M93P 10A8-A0PW00 mini station with Intel Core-i7 4770 at 3.4GHz, Intel Integrated VGA with HD Graphics 4600, and 16GB memory. Our Unity developed UI runs on a Lenovo P70 mobile workstation with Intel Xeon CPU E3-1505M v5 at 2.81GHz, NVIDIA Quadro M5000M graphics card, and 32 GB memory. The resolution of grid cells is set to be 0.05m×0.05m with an initial grid size of 10m×10m on both ROS and UI. The robot is tele-operated using a joystick and the rate of map generation is set to be 0.5 seconds. Fig. 6 (a) shows our VRL testing environment that features some challenges such as obstacles lower and higher than the Lidar's scanning plane (Fig6 (b)). The operator was in the same area being mapped, and had direct view on the robot and the room, in addition to a live video stream from the Flea3 camera. In all experiments, the user controlled the

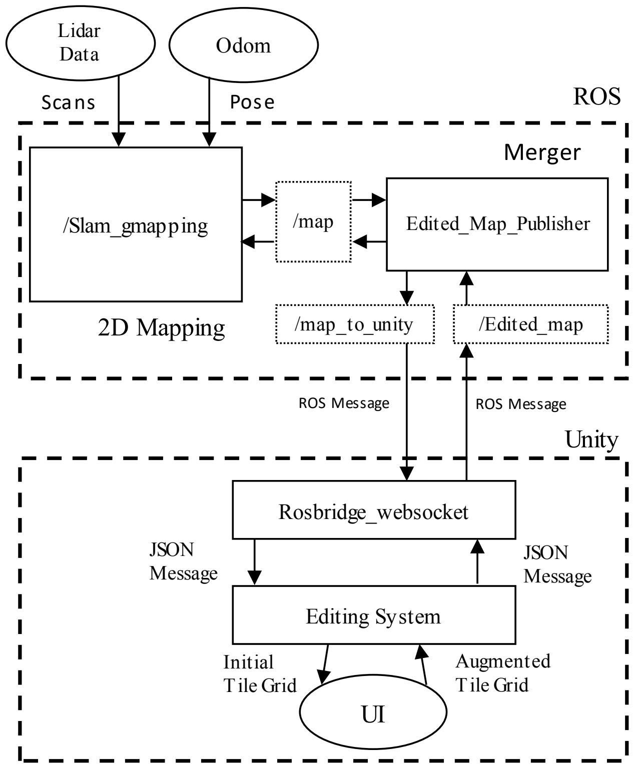

Figure 4. Detailed system architecture.

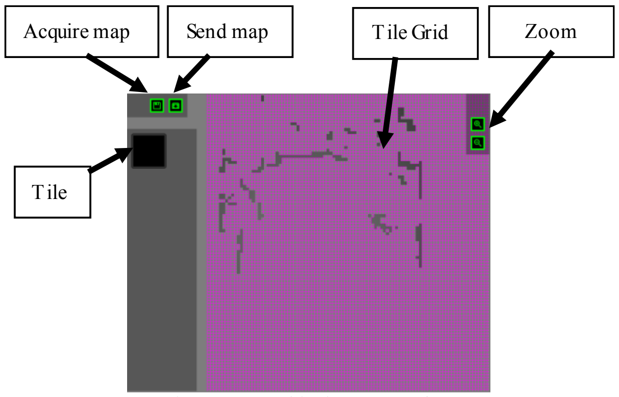

Figure 5. Graphical User Interface

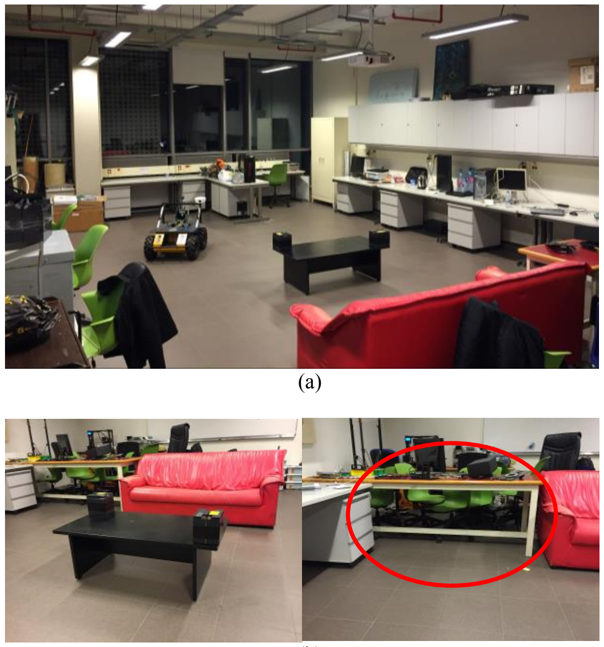

Figure 6. Vision and Robotics Lab testing environment. (a) a photo of the lab where experiment is conducted. (b) examples of low obstacles: black table in the middle, and high obstacles: the table surrounded by the red circle.

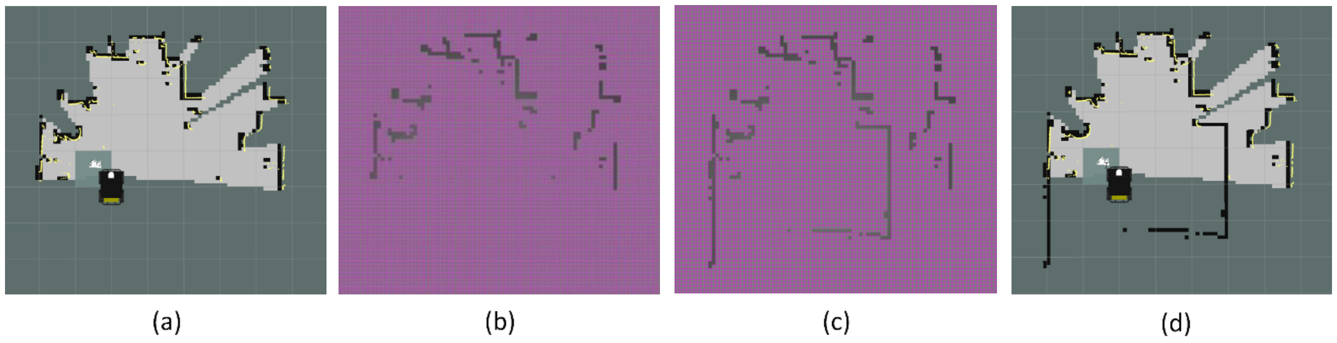

Figure 7. Example of map editing procedure. (a) the original map built by ROS packages visualized in rviz. (b) map graphically represented in our graphical user interface. (c) map after performing edits. (d) edited map as receives by the robot to continue mapping over it.

movement of the robot around the room to produce and edit the maps at different stages of the run. At every stage, the robot is posed in place before acquiring and editing the map in UI. For comparison reasons, we are publishing both the edited and original maps on the ROS side using the rviz visualization tool. An example of the map editing procedure is shown in Fig. 7. Fig.7(a) shows the initial map built by the Gmapping package. After requesting it by the user, it is displayed in Fig.7 (b) as a tile grid in the UI. Fig.7 (c) shows the tiles added by the user as an example of adding edges to the map. Finally, Fig. 7(d) shows the augmented map as received by Gmapping package and presented in rviz. The edges added by the human are visible in the final figure.

At the first stage of the run, the robot is moved to map the right half of the lab. Fig. 8 (a) shows errors in the map due to sensor's noise and failure in detecting the table in the middle of the room and the high table at the corner. The robot is posed in place, a map is acquired by the UI, and edits are made to correct for the errors. Fig. 8 (b) shows the map after editing. The robot is moved again to continue mapping over the edited map then it is posed in place again to add undetected obstacles and edges. The map before and after editing is shown in Fig. 9. After mapping the entire area, the robot's map is acquired by the UI, where a final edit is performed to correct for all noise, and delete unwanted obstacles, such as chairs. The final clean version of the map versus the unedited map is shown in Fig. 10. To show the importance of our system when slippage occurs, we rotated the robot by hand such that the encoders do not record any change in position. Fig. 11 (a) shows the resulting map where the robot superposed a rotated map of the walls, and other obstacles on top of the original map. This map is edited using our technique and the augmented map resulted is shown in Fig. 11 (b). As we can see, the error in mapping due to the simulated robot slippage was successfully removed.

The entire mapping/editing procedure was performed in 5 minutes and 11 seconds, while the average mapping time for the same area without editing was 4 minutes and 6 seconds. Through these experiments, we were able to add missing edges

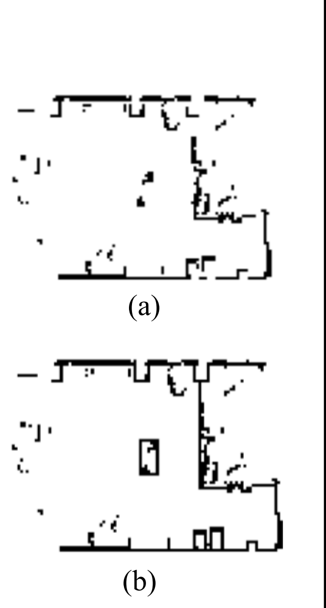

Figure 8. Maps of first stage. (a) map built by the robot throughout the first stage of the experiment. (b) map after editing

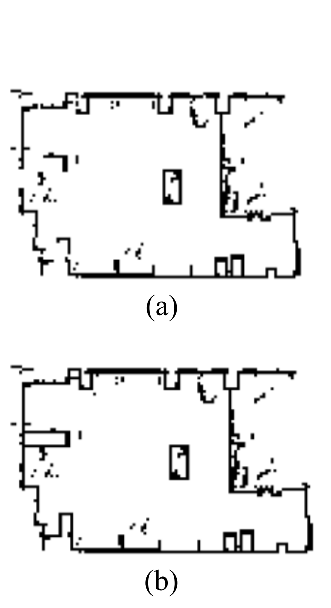

Figure 9. Maps of second stage. (a) map built by the robot throughout the second stage of the experiment. (b) map after editing

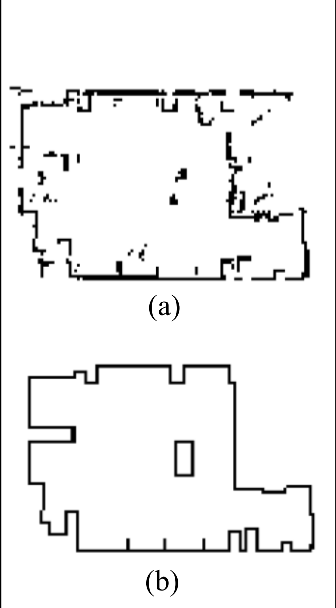

Figure 10. Final map. (a) the final map in case no edits were applied through different stages (b) Final map after editing

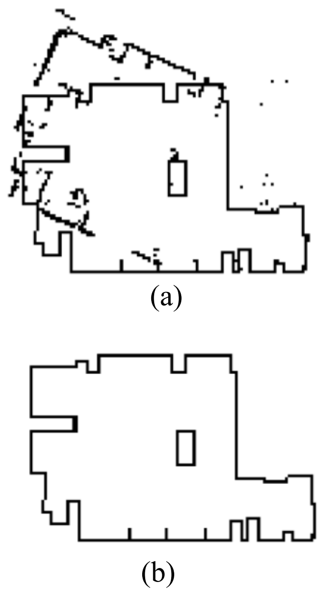

Figure 11. Map when slippage is introduced. (a) New map is superposed over the initial map. (b) map after editing

and boundaries of objects that are lower/higher than the LIDAR's detection plane, delete outliers and unwanted objects from the scene, clean all edges, and correct the map after a pose jump. The proposed framework can be easily implemented with any 2D grid-based mapping technique. Moreover, any user with average computer skills can use this system to perform edits to maps in real time.

V. Conclusion

In this paper, we presented a framework to augment maps by humans, in real-time. Our proposed system allows for onsite map editing capabilities instead of the traditional offline map editing. We developed our system using ROS packages and Unity software. The user interface allows for adding or removing obstacle boundaries from the original map and sending it back to the robot. We evaluated our proposed system in real-world experiments. The system showed both utility and efficiency in correcting maps for sensors' noise and different types of errors. Future work will take into account automating the maps sending/receiving process and correcting for the robot pose. Moreover, we will make use of Augmented Reality technology for easier user interface and more accurate map editing.

Acknowledgment

This research was funded and supported by the AUB University Research Board.

References

[1] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2D LIDAR SLAM," in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, 2016, pp. 1271-1278: IEEE.

[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*, 2005, pp. 2432-2437: IEEE.

[3] D. Hahnel, W. Burgard, D. Fox, and S. Thrun, "An efficient FastSLAM algorithm for generating maps of large-scale cyclic environments from raw laser range measurements," in *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, 2003, vol. 1, pp. 206-211: IEEE.

[4] G. Younes, D. Asmar, E. Shammas, and J. Zelek, "Keyframe-based monocular SLAM: design, survey, and future directions," *Robotics and Autonomous Systems,* vol. 98, pp. 67-88, 2017.

[5] H. Moravec and A. Elfes, "High resolution maps from wide angle sonar," in *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*, 1985, vol. 2, pp. 116-121: IEEE.

[6] C. E. Vido and F. Ramos, "From grids to continuous occupancy maps through area kernels," in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, 2016, pp. 1043-1048: IEEE.

[7] R. S. Merali and T. D. Barfoot, "Optimizing online occupancy grid mapping to capture the residual uncertainty," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, 2014, pp. 6070-6076: IEEE.

[8] J. Ryde and H. Hu, "3D mapping with multi-resolution occupied voxel lists," *Autonomous Robots,* vol. 28, no. 2, p. 169, 2010.

[9] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots,* vol. 34, no. 3, pp. 189-206, 2013.

[10] E. Ataer-Cansizoglu, Y. Taguchi, and S. Ramalingam, "Pinpoint SLAM: A hybrid of 2D and 3D simultaneous localization and mapping for RGB-D sensors," in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, 2016, pp. 1300-1307: IEEE.

[11] A. Ratter and C. Sammut, "Fused 2D/3D position tracking for robust SLAM on mobile robots," in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, 2015, pp. 1962-1969: IEEE.

[12] G. Gallegos and P. Rives, "Indoor SLAM based on composite sensor mixing laser scans and omnidirectional images," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, 2010, pp. 3519-3524: IEEE.

[13] T. Laidlow, M. Bloesch, W. Li, and S. Leutenegger, "Dense RGB-D-Inertial SLAM with Map Deformations," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 6741-6748.

[14] K. Kamarudin *et al.*, "Improving performance of 2D SLAM methods by complementing Kinect with laser scanner," in *Robotics and Intelligent Sensors (IRIS), 2015 IEEE International Symposium on*, 2015, pp. 278-283: IEEE.

[15] M. Quigley *et al.*, "ROS: an open-source Robot Operating System," in *ICRA workshop on open source software*, 2009, vol. 3, no. 3.2, p. 5: Kobe, Japan.

[16] Gmapping, ROS package. Accessed Oct,2017.[online] http://wiki.ros.org/gmapping

[17] M. Jenkin. (Oct, 2017). *unityros*. Available: https://github.com/michaeljenkin/unityros

[18] rosbridge, ROS package. Accessed Oct, 2017.[online], http://wiki.ros.org/rosbridge

[19] E. A. Topp and H. I. Christensen, "Tracking for following and passing persons," in *IROS*, 2005, pp. 2321-2327.

[20] S. Kim, H. Cheong, J.-H. Park, and S.-K. Park, "Human augmented mapping for indoor environments using a stereo camera," in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, 2009, pp. 5609-5614: IEEE.

[21] A. Diosi, G. Taylor, and L. Kleeman, "Interactive SLAM using laser and advanced sonar," in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, 2005, pp. 1103-1108: IEEE.

[22] P. Vieira and R. Ventura, "Interactive mapping in 3D using RGB-D data," in *Safety, Security, and Rescue Robotics (SSRR), 2012 IEEE International Symposium on*, 2012, pp. 1-6: IEEE.

[23] S. Jia, H. Yang, X. Li, and W. Fu, "SLAM for mobile robot based on interactive GUI," in *Mechatronics and Automation (ICMA), 2010 International Conference on*, 2010, pp. 1308-1313: IEEE.

[24] P. de la Puente, D. Rodriguez-Losada, A. Valero, and F. Matia, "3D feature based mapping towards mobile robots' enhanced performance in rescue missions," in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, 2009, pp. 1138-1143: IEEE.

[25] Y. Nihei *et al.*, "A method of localisation and multi-layered 2D mapping using selective update for particle filter," in *Robotics and Biomimetics (ROBIO), 2014 IEEE International Conference on*, 2014, pp. 1398-1403: IEEE.