Int'l Journal of Computing, Communications & Instrumentation Engg. (IJCCIE) Vol. 2, Issue 2 (2015) ISSN 2349-1469 EISSN 2349-1477

Localization and Navigation of Autonomous Indoor Mobile Robots

Osama Hamzeh, and Ashraf Elnagar

***Abstract*—**Tele-robotic localization systems vary in implementation, but the cost of building such solutions is high. Therefore, utilizing such solutions in complex areas becomes a very difficult choice. We propose to use a low-cost localization and navigation solution that consists of a low cost Kinect sensor along with a normal laptop to control a small mobile robot. Our proposed solution involves remotely controlled mobile robot for navigating a pre-built MAP of an unknown environment. Experimental results confirm the success of the prototype design and implementation.

***Index Terms*—**Localization; Turtlebot 2; Kinect; ROS; Path Planning, Collision Avoidance.

I. Introduction

The main idea of sending the robot in a place were no human can enter, so that it can traverse the location, send the images and avoid obstacles by a human controlled interface and utilizing the depth information collected by the Kinect as discussed in different implementation ([1 - 4]). For example in a search and rescue operation this can be humans

Tele robotics is the area of robotics concerned with the control of robots from a distance, mainly using wireless connections (like Wi-Fi, Bluetooth, the 3G/4G Networks, and similar), "tethered" connections, or the Internet. This will allow applications in the fields of search and rescue, military, explorations and so on.

The main idea of sending the robot in a place were no human can enter, so that it can traverse the location, send the images and avoid obstacles by a human controlled interface and utilizing the depth information collected by the Kinect as discussed in different implementation ([1 - 4]). For example in a search and rescue operation this can be humans trapped in certain locations of a building, a rescue team can be sent later on to the exact location to rescue the casualties.

Localizing within indoor environments is very important for task-driven indoor mobile robots. Lots of approaches to localization have been researched, including using laser scanners, vision and wireless strength, [8]. Wireless strength indoor localization is a good approach but requires spreading lots of wireless probes around the localized localization.

Other works researched probabilistic localization techniques for humanoid robots by utilizing 3D representation of arbitrary complex environments. They also had to deal with all the challenges the might occur during the humanoid robot navigation. Such approach does not relay much on Odometry information and does not require a flat world assumption, [9].

Another solution which does not require any additional infrastructure in the environment and provides 3D positioning and orientation data was introduced, [10]. This was inspired by camera-projector calibration techniques, this solution uses a camera to locate and track a grid pattern projected onto surfaces in the camera's field of view to determine its distance and orientation to multiple fixed large planes in a space, this solution relays on image processing which requires high CPU utilization.

The main objective for this paper is to build a cost effective solution that can localize and navigate within a given MAP using low cost equipment while maintaining the collected information and to provide a navigation solution in both static and dynamic environments.

This paper is organized as follows. Section 2 describes the components compromising the solution. Section 3 details the simultaneous localization and navigation processes work. Section 4 provides details on the experiments results. Finally, we conclude the work in Section 6.

II. Components of the Solution

This solution was proposed to provide an affordable working indoor localization solution that can operate in homes, offices and especially in environments where it is dangerous to send people. While there are other expensive solutions that might does the same work, our solution becomes very handy when the risk of losing the robot is very high.

A. Robot Operating System (ROS)

This is an open source implementation for a public operated system in Robotics. This creates an ecosystem where different components called (nodes) are interconnected through a computer operated network, this network is used to pass messages between the different nodes, these messages can be Odometry information, control commands, video streams or images, or any kind of information that two nodes or more need to communicate. ROS, [5], also provides a wealth of plugin libraries to provide different parts of robotics functionality, this includes tele operating libraries, localization and navigation techniques, the plugin's source code is also available, as ROS is an open source implementation.

ROS requires a Linux distribution base operating system to operate, this provides the layer between ROS and the actual laptop hardware, ROS includes hardware abstractions and low level device controls in order to control the connected devices whether they are robots or sensors connected to the actual laptop itself.

As ROS communicates with difference nodes and sends

Osama Hamzeh is a graduate student with the Department of Computer Science, University of Sharjah, P O Box 27272, Sharjah, UAE

Ashraf Elnagar is a professor with the Department of Computer Science, University of Sharjah, P O Box 27272, Sharjah, UAE

control commands between them, time becomes a very important factor that requires to be synchronized between them, the use a Network Time Protocol (NTP) becomes vital.

Some areas of robotics research, such as indoor robot navigation, have matured to the point where "out of the box" algorithms can work reasonably well. ROS leverages the algorithms implemented in projects to provide a navigation system.

Although each node can be run from the command line, repeatedly typing the commands to launch the processes could get tedious. To allow for "packaged" functionality such as a navigation system, ROS provides tools that reads XML description of a graph and instantiates the graph on the cluster, optionally on specific hosts.

A single Ctrl-C will gracefully close all five processes. This functionality can also significantly aid sharing and reuse of large demonstrations of integrative robotics research, as the set-up and teardown of large distributed systems can be easily replicated.

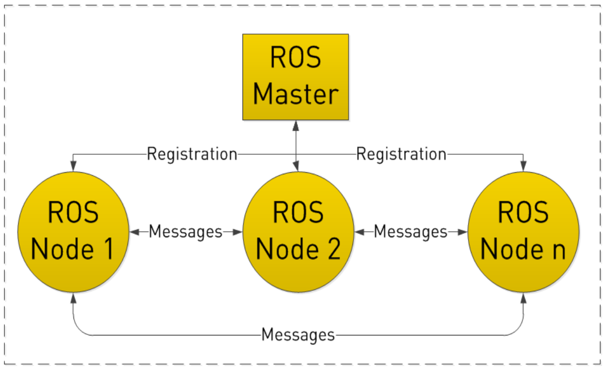


Fig 1. The ROS Ecosystem.

However, a more powerful concept is a visualization program that uses a plugin architecture: this is done in the RVIZ, [6], program, which is distributed with ROS.
Visualization panels can be dynamically instantiated to view a large variety of datatypes, such as images, point clouds, geometric primitives (such as object recognition results), render robot poses and trajectories, etc. Plugins can be easily written to display more types of data.

Fig. 2. The Turtlebot II Robot.

ROS have evolved through time since the release of ROS 1.0 in 2010, now the latest distribution is called ROS Jade Turtle, but for the sake of stability we decided to use ROS Indigo since most of the libraries are supporting the robot that we chose for this paper which is the Turtlebot II.

B. Turtlebot II

The Turtlebot II, [7], shown in Fig. 2 was built on the success of the original Turtlebot, it is an open robotics platform that consists of a motorized wheeled base that is the Yujin Kuboki hexa-base, on top of it is the Microsoft Kinect sensor that provides depth information. Video input and an audiometer (which is not utilized).

Some of the main features for this robot other than it is low cost, is that it includes Wheel encoders, Integrated gyroscope, Big batteries, Bump sensors, Cliff sensors and Wheel drop sensors. The speed of the robot can reach up to 50 cm/s.

The robot includes a lithium battery, which provides the electrical power for the robot itself, and the connected Kinect sensor.

C. Client PC

The client PC is installed with Ubuntu 14.04 and is connected to the Robot and the Kinect Sensor, the Kinect is mounted on top of the Turtlebot II; this PC collects all the sensor's information such as:

1) Depth information received from the Microsoft Kinect sensor.
2) Images and video stream received from the Kinect.
3) Odometry information received from the Turtlebot II.

ROS is installed on this PC, here we describe the application/handlers running on the client PC itself:

1) Turtlebot handler (hardware driver): this is the main driver to connect the components of the Robot through a USB cable.
2) OpenNI handler: this the driver that connects the Microsoft Kinect to the PC to collect its depth information, Video and Audio streams.
3) Depth_to_laserscan application: this is application converts the Kinect's depth information into a horizontal 'line of site' laser scan data, which is used later in the main application to detect and build the grid map.
4) Map server: This application is used to collect the built grid map from the main application and sends a copy of the map to the workstation for viewing/saving.
5) The Adaptive Monte Carlo Localization (AMCL) application: this is the main ROS application that processes all the sensor's information (Laser scan, Odometry) to Localize and Navigate.

D. Workstation PC

This PC is installed with Ubuntu 14.04 and ROS indigo. This PC is a remote machine that controls the Client PC, views the maps, saves the grid maps and view the video stream, all of this is done through the below components:

1) ROSCORE application: this is the main ROS application which should run on the ROS MASTER Node, all other ROS commands and nodes will communicate to the ROS master node to get a list of all the available nodes in the ROS ecosystem, so it acts as a database listing all the nodes and how to reach them.
2) Tele-operating application: this is an application that connects to the ROS ecosystem and sends control commands to the Robot to coordinate its movement, this can be done through the keyboard or a joystick connected to the workstation PC.

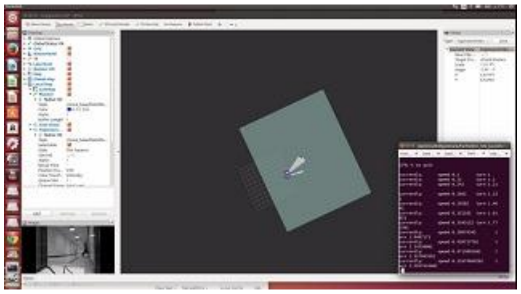


Fig 3. The RVIZ Visualizer

3) RVIZ: this is a 3D visualizer for displaying sensor data and state information from ROS. Using RVIZ, you can visualize the current configuration on a virtual model of the robot. You can also display live representations of sensor values coming over ROS Topics including camera data, infrared distance measurements, sonar data, and more, check Fig 3.
4) Map_saver: this is an application that saves the current built grid map to a local file on the workstation.

E. The Network:

Any TCP/IP network will work for ROS as long as the nodes have access to the main ROS master node, which is the workstation PC with Ubuntu 14.04 and has ROS indigo installed and the ROSCORE application running,

Name resolution is also essential for the ecosystem to communicate as the IP address is not enough to communicate and any communication is done through the host name.

For the sake of mobility the client PC should be connected to either a Wireless Network Infrastructure or 3G/4G network that is routable to the main workstation, while the workstation PC can have any type of connection as long as it has the routes to reach the Client PC.

Bandwidth should be taken into consideration as the real-time video streaming from the Kinect takes around 4 Mb of traffic, failing to provide such bandwidth will cripple the solution and might lead to a disconnected robot, running the solution without a video stream is also possible and does requires almost 200 Kb to operate.

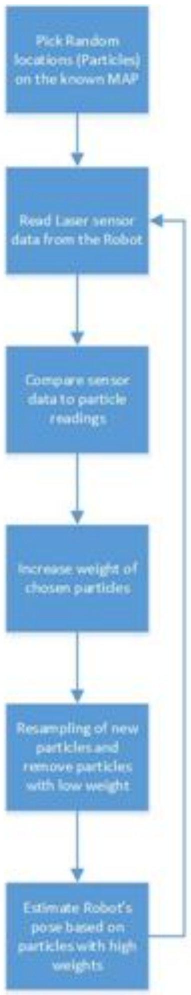
Latency is another important factor, as the robot is tele-operated, failing to receive control commands at the right time might lead to the robot hitting obstacles as the control commands might reach the Client PC late, latencies up to 200 ms are found to be acceptable.

III. Navigation stack

The ROS navigation stack implementation [11] is based on continues localization and path planning, the localization is done through the Adaptive Monte Carlo Localization (AMCL) algorithm [12]. The Algorithm itself relies on the randomization approach to overcome the complexity of the problem, as it will choose arbitrary locations on the local MAP (called particles), these particles are then saved in a list that includes their X and Y coordinates and their distances from all the nearby boundaries and obstacles, the angle of the distances is also stored in the same list.

When the Robot laser scans are received and transformed into distances and angels, they are compared with the data collected and saved earlier in the randomized particles, the

particles with the closes readings to the Robot will be given a higher weights and will be chosen as possible poses for the Robot itself, this process will continue as the Robot is localized to the MAP itself. At the next iteration of the procedure a resampling process will be done and only particles with higher weights will be included in the process of localization, this will make the number of possible particles less and less with time and the Robot pose will be guessed correctly. The AMCL algorithm is shown in fig. 4:


Pick Random locations (Particles) on the known MAP
Read Laser sensor data from the Robot
Compare sensor data to particle readings
Increase weight of chosen particles
Resampling of new particles and remove particles with low weight
Estimate Robot's pose based on particles with high weights
Fig 4. Workflow For The AMCL Algorithm

For the navigation process itself to start and before the

AMCL algorithm to run, we need to manually localize the Robot in the RVIZ interface, this is done by setting an approximate location and orientation for the Robot, once that is done we will have to set the Goal location (Fig 5) and then the main navigation will start.

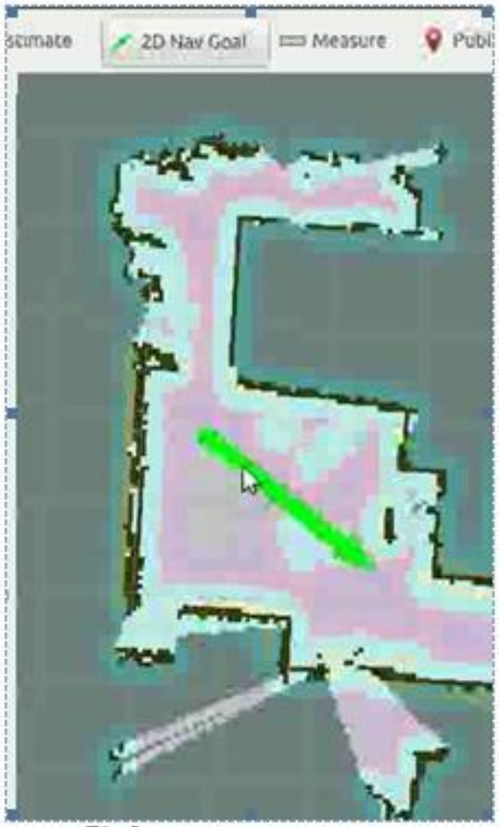


Fig 5. Setting The Goal In RVIZ

The global planner [13] process starts once we select the Goal, the whole MAP is organized as a binary image of zeros and ones, this MAP should include open spaces, unknown areas, obstacles and the Robot itself is represented as a dot on the MAP. To make things easier for the navigation process there is an extra space that is added to each obstacle (object and wall) which inflates the area, this area is easily calculated as our Turtlebot Robot is circular in shape, this area will be considered part of the obstacles and the Robot should not touch it, the global planner process will utilize the Dijkstra's famous algorithm [14] to calculate the path from the current location to the Goal.

After the path is calculated, the global planner will translate the path into commands that controls the Robot, these are acceleration and direction commands sent as ROS messages within the ROS ecosystem.

While the Robot is utilizing the planned path and moving, the AMCL process and the sensors readings from the Robot are also running and updating the MAP and this is where simultaneous localization and mapping (SLAM) is happening [15] , so if a new obstacle is encountered then the global planner will check if this new obstacle is in the path that was calculated, if it is in the path then it will pause the Robot and recalculate a new path around this new obstacle from the current position to the Goal, if a new path cannot be generated then the Robot will a command to make a full 360 degrees turn in his place to explore possible changes in the environment and it will then try to recalculate a possible path, if no possible path is found then the Robot will stop and report that no possible path is found.

IV. Experimental results

To check the performance of the proposed solution, we ran the solution against multiple environments to see the output in different scenarios, we discuss the multiple scenarios below:

A. *Scenario I: House environment*

A Learned MAP for a normal apartment consisting of three bedrooms, kitchen and a living room was used in this experiment. The Robot was localized at first manually and was requested reach a Goal in a different room, the test was done first in a static environment and the Robot was able to reach the location and the orientation requested, this is illustrated in Fig 6. Another test was done by adding obstacles to the same MAP that were not available in the original MAP, again the results were the same and the Robot was able to maneuver successfully to its given Goal around the new obstacles, this is illustrated in Fig 7.

Another final experiment was done to simulate a dynamic environment, a small kid was asked to intercept the Robot while it was utilizing the calculated path to its Goal, and the results were quite amazing as the Robot was able to maneuver the kids jumping in front of the Robot.

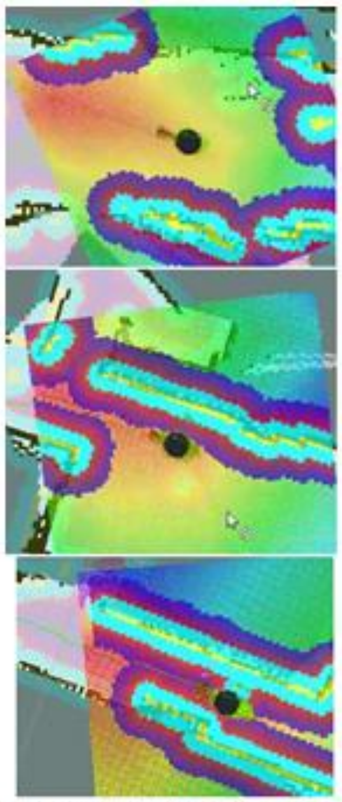


Fig 6. Navigation With Static MAP

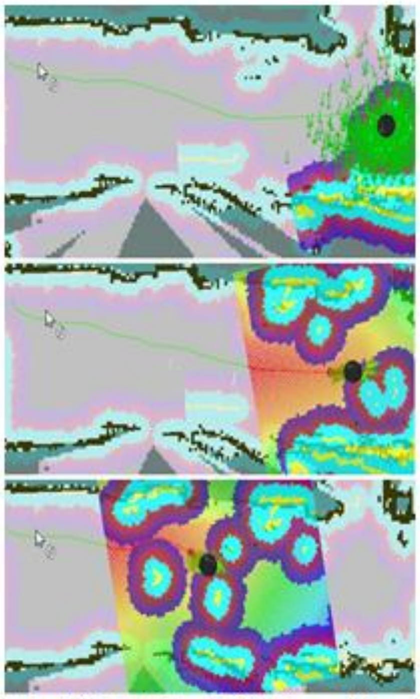


Fig 7. Navigation With Obstacles; The First Image Shows The Path Without The Obstacles While The Other Two Images Below Show The Detection And Avoidance Of Obstacles.

B. Scenario II: Office environement

The second scenario was done in the Computer Science Department building, which is a typical office environment, the three navigation scenarios of static environment of a known MAP, extra obstacles added to the known MAP and a final experiment by adding dynamic changes while the Robot is moving, the results are shown in Fig 8.

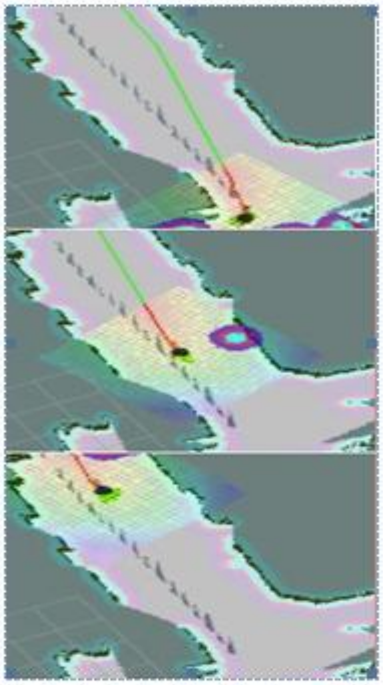


Fig 8. Navigation In An Office Environment. This Is An Example Of Dynamic Environment.

V. Conclusion

The proposed solution provides a low cost alternative to existing solutions that navigates through a known MAP for indoor environments. Using the solution to navigate unsafe areas where we cannot send human personals becomes feasible especially when the total cost of losing the solution can be neglected compared to the cost of other expensive robots and because this solution adapts to sudden changes in the environment. Kinect's depth information quality is acceptable for navigating indoor global maps and avoiding obstacles in dynamic environments.

As the new CPUs coming out are becoming smaller, more powerful, less power demanding and less expensive, we noticed the Raspberry PI board which can act as a normal PC, ROS already supports Raspberry PI but the current CPU speed is not suitable for our solution. Similarly, drones are becoming very popular and their cost are making them very attractive, combing a drone with the Raspberry PI and the Kinect would provide a perfect solution that can overlook most of the physical environmental difficulties that the Turtlebot cannot coup with, especially flat ground which is required for it to operate.

References

[1] Erickson H.; LaValle M., Navigation among visually connected sets of partially distinguishable landmarks, Submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science in the Graduate College of the University of Illinois at Urbana-Champaign, 2012

[2] Ganapathi V.; Plagemann C.; Koller D., Real Time Motion Capture Using a Single Time-Of-Flight Camera , published in Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference Pages 755-762

[3] Khandelwal P., A low cost ground truth detection system for RoboCup using the Kinect. Proceedings of the RoboCup International 2011

[4] http://mirror2image.wordpress.com/2010/11/30/how-kinect-works-stereo-triangulation/ (accessed June 2015)

[5] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng, ROS: an open-source robot operating system, in ICRA Workshop on Open Source Software, 2009.

[6] http://wiki.ros.org/rviz (accessed July 2015)

[7] http://kobuki.yujinrobot.com/home-en/about/reference-platforms/turtlebot-2/ (accessed June 2015)

[8] Joydeep Biswas and Manuela Veloso, WiFi Localization and Navigation for Autonomous Indoor Mobile Robots, in International Conference on Robotics and Automation, 2010. http://dx.doi.org/10.1109/robot.2010.5509842

[9] Armin Hornungm, Kai M. Wurm Maren Bennewitz, Humanoid Robot Localization in Complex Indoor Environments, Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2010.

[10] Moritz Köhler, Shwetak N. Patel, Jay W. Summet, Erich P. Stuntebeck2, and Gregory D. Abowd, TrackSense: Infrastructure Free Precise Indoor Positioning Using Projected Patterns, 5th International Conference, PERVASIVE 2007, Toronto, Canada, May 13-16, 2007. Proceedings

[11] http://wiki.ros.org/amcl (accessed August 2015)

[12] Dieter Fox , Wolfram Burgard , Frank Dellaert , Sebastian Thrun , Monte Carlo Localization: Efficient Position Estimation for Mobile Robots, IN PROC. OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE.

[13] http://wiki.ros.org/global_planner (accesses August 2015)

[14] https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm (accessed August 2015)

[15] Jose Cano, Eduardo Molinos, Vijay Nagarajan, Sethu Vijayakumar, Dynamic process migration in heterogeneous ROS-based environments, In Proc. 17th IEEE International Conf. on Advanced Robotics (ICAR 2015), Istanbul, Turkey (2015)
http://dx.doi.org/10.1109/ICAR.2015.7251505

[16] Zhengyou Zhang, Microsoft Kinect Sensor and Its Effect, IEEE MultiMedia, vol.19, no. 2, pp. 4-10, April-June 2012, doi:10.1109/MMUL.2012.24
http://dx.doi.org/10.1109/MMUL.2012.24

[17] https://digitalerr0r.wordpress.com/2011/06/21/kinect-fundamentals-3-getting-data-from-the-depth-sensor/, (accessed June 2015)