Ginger: First Lessons Learned from Deployment of A Long-term Autonomy Restaurant waiter Robot

Ashutosh Chapagain , Sagar Shrestha

Abstract

Over the past 2-3 years, we at Paaila Technology have successfully deployed mobile robots in Kathmandu. Our approach is majorly based on the adoption of the open-source software development kit, Robot Operating System (ROS). ROS[1] navigation stack was taken as a starting point and major improvements were made on the navigation system. We present an overview of overall navigation system and designs for the construction and successful long-term deployment of mobile robots.

navigation, ros-melodic , amcl , kalman-filter , epipolar geome

1. Introduction

Robot technology has significantly advanced over recent years, and we have a milestone where we can apply these innovations to the real world applications[5]. To leverage the use of AI, we at Paaila Technology, along with Naulo restaurant which is one of the biggest restaurant chains of Nepal, have deployed waiter robots in two restaurant outlets. The robot recognizes people's faces, talks to them in Nepali/English and delivers food to their table.

However, while robots often perform flawless demos and win applause under controlled settings with operator's guidance, they frequently fail when working for an extensive length of time in restaurant environment without close supervision[3]. On top of that, as mentioned in [5], long term deployments adds more uncertainties and covers many corner cases in the technical components. As a result, long term autonomy is a significant challenge in service robots.

We present Ginger, a long-term autonomous mobile waiter robot deployed in an unsupervised real-world environment. In the first year of deployment, Ginger received a lot of attention from national and international media outlets. They have served customers 10 hours per day, every day of the week.

Ginger operates in the restaurant environment at Naulo Restaurant. They are initially placed at the entrance where it is charged and they detect passersby with face detection and interact with them using speech. Fig shows the map of the restaurant space. Ginger stands at the hallway right after the entrance to greet the customers.Customers order food through a tablet fitted in the respective tables. After the food is ready, Ginger will get a notification about the order and firstly fetches food from the kitchen and finally head towards the customer's table.

Figure 1: Ginger Robots

The paper is organized as follows, in the technical specification section we introduce the tools and the current software drivers in ROS Navigation stack and in the following section we purpose the alternatives that were helpful in our case to achieve long term autonomy.

2. Related Works

Valerie[6] roboceptionist is the one of first robot to Carnegie Mellon's social robots project. The robot combines useful functionality - giving directions, looking up weather forecasts, etc. - with

an interesting and compelling character. Their analysis indicated several design decisions that should facilitate more natural human-robot interactions. RHINO deployed at University of Bonn described in detail the major components of a software architecture of a successfully deployed mobile robot. The robot's task was to provide interactive tours to visitors of a museum with an emphasis on robust localization and navigation in a crowded environment[7].

The SPENCER project[8] investigates social interaction in a crowded environment with multiple humans, which is critical in an unsupervised open setting. Kanda et al. deployed Robovie robots in a corridor of a shopping mall for five weeks to provide information to the public [9].

Ginger Bot shared most of its features with the service robots mentioned above. However, since Ginger works in a restaurant, therefore it receives its own set of unexpected failure cases that hinder its long term autonomy.

3. Technical Specification

3.1. Overview

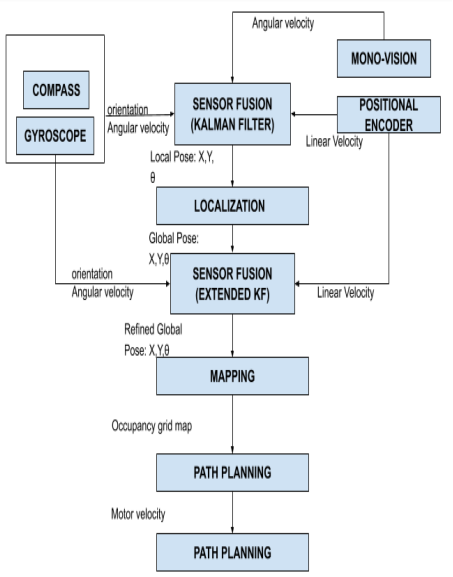


Figure 2: System Block Diagram

Each unit of ginger robot is equipped with _ lidar, _ camera , _ IMU, .Ginger has onboard Jetson Nano computers, and it utilizes many third-party components and open-source software. It is running the Robot Operating System (ROS) Melodic version (released in 2018). The ROS SDK provides SLAM Algorithms for localization and mapping , and the ROS navigation stack for lower-level control for the robot to move around. Although its Speech Recognition capability is beyond the scope of this paper, it is worth mentioning that the system leverages Google Cloud Speech API translates speech audio to text with nearly real-time feedback with the restaurant customers.

A complete navigation solution is realised by the combination of appropriate hardware and algorithms. Fig.1 shows our approach for navigation of a differential drive robot. Data from IMU, encoder and monocular camera are fused to obtain an estimate of the local position. Scan matching of LiDAR data with the available map assigns a score to each local pose computed for particles in Monte Carlo localization. This localization model can represent any arbitrary probability density function over the robot position. This approach is very general but, due to its generality, its can suffer from multi modal failure modes. Example, the robot's pose estimation can oscillate when navigating through similar corridors in our restaurant deployment.

Robots usually include a large number of heterogeneous sensors, each providing clues as to robot position and, critically, each suffering from its own failure modes. We used a powerful technique for achieving this sensor fusion, called the Extended Kalman filter.This results in a robust estimate of the global pose that is used both for mapping and subsequently for path planning. The result of path planning assigns motor velocities to motion control unit which runs PID control to give appropriate PWM signals to the motors.

3.2. ROS Navigation Stack

Robot Operating System (ROS) is a set of software libraries and tools that help us build robot applications[1].

ROS navigation stack was used as the foun-

dation of Ginger's autonomous navigation due to its open source community and flexibility in their modifications. The navigation stack provides the most innovative implementations of perception, localization, mapping and planning algorithms required for navigation.The drivers are relatively complete and work well for a basic environment, despite some limitations which will be discussed later in the paper.

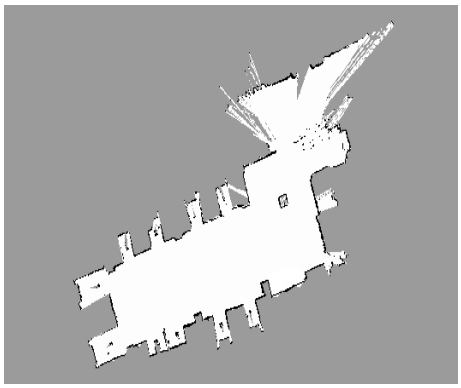


Figure 3: Map of one out the outlets of Naulo Restaurant

3.3. Perception

The ROS navigation stack uses information from perception sensors to avoid obstacles in the world. The inputs to the position and orientation estimation are sensor data acquired from gyroscope, digital compass and encoders.

For robots with laser scanners, ROS provides a special Message type in the sensor_msgs package called LaserScan to hold information about a given scan. We've used LiDar that publishes sensor_msgsLaserScan messages. LaserScan Messages make it easy for code to work with virtually any laser as long as the data coming back from the scanner can be formatted to fit into the message. Publishing a LaserScan message over ROS is fairly straightforward[cite].

3.4. Odometry Information

This represents an estimate of a position and velocity in free space. Since sensors are fitted around the robot_base, transform tree is required to cast the coordinate to common base coordinate. We initially had two wheel encoders to primarily report instantaneous velocity of each wheel, gyroscope to report angular velocity and compass to report absolute orientation.

3.5. Global Localization

Adaptive Monte Carlo localization (AMCL) is one of the most popular localization algorithms, which represents estimated pose by particles. It is a variant of implementation of particle filter and an improvised version of MCL, to localize the robot in given world space of map. The algorithm estimates the position and orientation of a robot in the given map as it moves and senses the environment. The algorithm can be used for global localization problem. It implements the adaptive (or KLD-sampling) Monte Carlo localization approach (as described by Dieter Fox), which uses a particle filter to track the pose of a robot against a known map. It tracks the offset needed to compensate for overall sensor drift as the robot drives around, allowing the robot to more robustly create global path plans.

*The amcl node provided in the ROS navigation stack, subscribes to the /scan message published from LiDAR and provides a global localization estimate in the robot's world or /map frame. The coordinate frame called /map is the world fixed frame, with its Z-axis pointing upwards. The pose of a mobile platform, relative to the map frame, should not significantly drift over time. The map frame is not continuous, meaning the pose of a mobile platform in the map frame can change in discrete jumps at any time[**?**]. Therefore, amcl provides discrete updates of the robot pose in the map based on its sensor measurement and estimates.*

3.6. Control and Robot Path Planning

Motion planning and Control play an important role in mobile robot navigation. Planning and Control can be divided primarily into two two levels, namely - global planners and local planners. In this framework, global planner creates a plan from start state to goal state without taking into consideration the kinematic constraints of the

robot, and local planner takes in the plan produced by global planner and generates appropriate control signal taking into account the kinematic constraints and current state of the robot. A more general framework can be found in [?].

Given a plan to follow and a costmap, the controller produces velocity commands to send to mobile base.

We initially used DWA local planner that serves to connect the path planner to the robot. DWA is a sampling based approach to obstacle avoidance. The dynamic window is a window in the action or velocity space of the robot centered around the current velocity. The window represents the set of admissible velocities derived from velocity and acceleration constraints of the robot. Using a map, the planner creates a kinematic trajectory for the robot to get from a start to a goal location. Along the way, the planner creates, at least locally around the robot, a value function, represented as a grid map. This value function encodes the costs of traversing through the grid cells. The controller's job is to use this value function to determine dx,dy,dtheta velocities to send to the robot. [Copied from ROS website]

3.7. Recovery Behaviors

The navigation system as described works well most of the time, but there are still situations, due to hardware failures and failures induced by human-robot interactions, where the robot can get stuck. To make the system as robust as possible, a number of recovery behaviors were built into the navigation system. [http] describes recovery behavior as follows When the robot finds itself in entrapment, where the robot is surrounded by obstacles and cannot find a valid plan to its goal, a number of increasingly aggressive recovery behaviors are executed by default from the move_base to attempt to clear out space. First, the robot clears all obstacles in its map that are further than 3m away by reverting to the initial map for all obstacles outside of a 2m×2m local window. If this fails, the robot performs an in-place rotation to attempt to clear out space.If this too fails, a more aggressive map reset is attempted where the robot clears all obstacles that are outside of its circumscribed radius. After this, another in-place rotation is performed to clear space, and if this last step fails, the robot will abort on its goal which it now considers infeasible.

4. Lessons Learned

In our long deployment of Ginger robot, we have seen people enjoying and walking with Ginger, but we have also seen many shortcomings in the system. These shortcomings include failures and faults, improper robot operation workflows, and imperfections in native ROS-navigation stack to premeditate human-robot interactions. We will use the above mentioned components as a starting point, discuss their shortcomings in our case and provide our solution for each problem.

It is common for engineering solutions to have hardware failures. In contrast to software failures, hardware failures are uncommon, but long-term deployment reveals some failures cases.

4.1. Device Driver failure

The robot computer fails to recognize a USB device during deployment, and reconnecting it did not solve the issue. The only known solution to us was to restart the robot computer.

4.2. Perception failure

LiDAR was one of our constant points of failure for long term autonomy. The ROS node handling the LiDAR would fail to restart the LiDAR once cancelled and it never booted up unless it was manually booted.

Because hardware malfunction is less common, they are often easy to ignore. However, to achieve long term autonomy we devised a high level controller to restart the computer if it consistently fails to recognize vital USB components including wifi-module and lidar.

4.3. Software failures and improvements

4.4. Odometry

We use Kalman Filters to fuse the data from raw sensor sources in our three tiered approach. Firstly, we wrote a robo_base driver to read left and right wheel rotation counts and raw gyroscope data. Secondly, we fused the raw gyroscope data with our readings from compass sensors to give an absolute orientation estimate. And finally, we wrote a driver to fuse encoder and orientation estimate from step 2 to give final odometry estimate. Since the wheel encoders primarily reports velocity of the body, we integrated it over time to get a rough estimate of the robot's position in the real world. All transforms were published in the base_link frame.

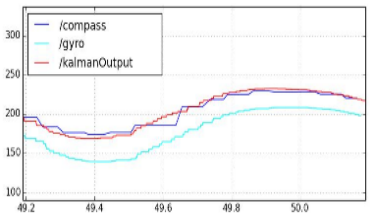


Figure 4: Sensor Fusion with Kalman Filter

However, the odometer's estimate of the robot's position in the map is very prone to error due to odometry drift. To solve this problem to a certain context, an altogether different low cost solution, visual odometry was implemented. Visual Odometry was originally intended to be used on Mars Rover [5], where there is no GPS and wheel odometry becomes unreliable due to slip on the sandy martian surface. We followed the standard approach to extract a sparse set of salient image features (e.g. points,lines) in each image; match them in successive frames using invariant feature descriptors; robustly recover both camera motion and structure using epipolar geometry; finally, we refined the pose and structure through reprojection error minimization. Due to scaling estimation error in linear velocity, we discarded it but we further refined the Instantaneous Rotation estimation.

The two odometry messages, one in the base_link frame and other in camera frame,were fused with the EKF_Localization. The two different odometry sources when fused together gave a very accurate estimation of the robot's orientation pose over time.

4.5. Localization

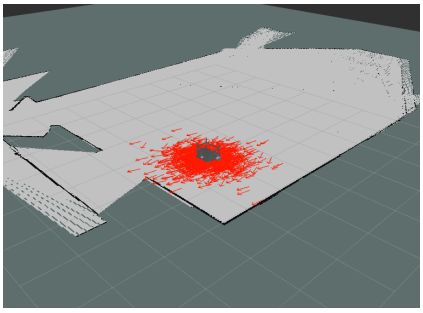
As amcl provides discrete jumps in identical environments, amcl pose estimate is prone to abrupt changes, which perturbed our estimate. We've implemented two methods to alleviate this problem.

- *Fuse the final amcl pose as shown in figure*
- *Continously compare amcl and odometry pose estimates*
- *If the two pose estimates vary by more than a certain threshold,*
- *Reinitialize amcl pose*

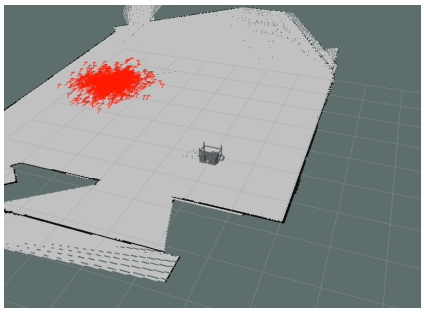
First of all we fused the amcl pose estimate with odometry information to get a better overall estimate of the robot's global pose. We used wheel encoders as the primary source for x_vel ,y_vel and fused data from IMU and gyroscope as primary source for absolute orientation. Additionally, we used the x and y position estimate provided by the amcl package. The EKF localization provided with robot_navigation package node fused x,y, x_vel, y_vel and theta to provide the accurate global pose robot that was less susceptible to abrupt jumps in the environment. Additionally, we developed a simple heuristic function to discard the amcl pose estimate if instantaneous pose estimate is too large in comparison with odometry change. For example to solve multi-modal problem in localization, we continuously check odometry and amcl pose estimates. Since odometry pose estimate cannot have discrete jumps, we developed a mechanism to detect amcl jumps and reinitialized the amcl pose which solved our problem. To summarize we improved the localization estimate in the following way,

4.6. Control and Robot Path Planning

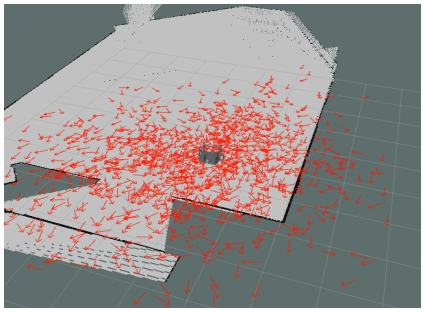
DWA is rather sufficient in most cases however we faced the following problems when deploying



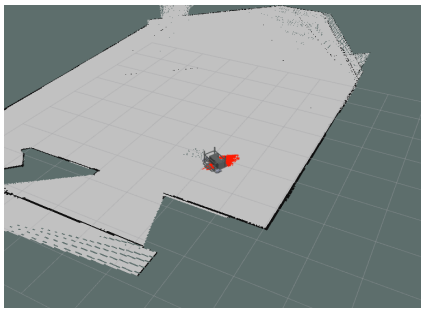
(a) initial amcl estimation of robot's pose



(b) amcl forced to estimate the robot's pose elsewhere in the map



(c) Re-estimation of amcl pose



(d) amcl pose converging to true value

Figure 5: amcl pose re-initialization

our robot with DWA local planner plugin

Since robot serving foods are a major fascination to our customers, they would deliberately get in the robot's way in a jest and to play with the robot. Although driving around the obstacle could be germane for other cases, in our case, the robot would try to drive around all its human obstacles in tight environment space, and would ultimately abort its goal and fail miserably. Although the idea behind DWA is to evaluate samples of velocities (linear and angular in case of differential drives) in the dynamic window against an objective function at each control loop to select the best scoring velocity, we could not tune the parameters to have a biases towards zero velocity when an obstacle is right in front of the robot. Furthermore, DWA could not perform in place rotation. But this feature is highly desirable for waiter-robots, especially when it delivers the food and needs to return to their home position. Additionally due to space constraints too, the robot would more often get stuck and abort its goal when trying to rotate with DWA sampling methods in enclosed space.

Such cases were constantly experienced in our narrow corridor deployments, which led us to explore other rather simpler strategies to help tackle this issue.

We came up with a simple PD Controller that follows the global plan without obstacle avoidance, only waiting. This local navigation plugin has a simple PD controller for path following and a little look-ahead stopping movement once it encounters an obstacle. Furthermore, to fulfil our business need for in-place rotation as well as for further stability in the path generated by this PD Controller, we defined a heuristic to send a rotation command only if the current orientation diverges with the goal orientation by a certain amount. Extensive testing was done in this regard to achieve the appropriate constants for the controller and maximum velocities that suited varied scenarios and spaces present on the restaurant.

The above mentioned solution solved our problems. Our PD-Controller waits indefinitely if there's a human obstacle in front of it. Second of all, after the ginger robot delivered food to the customer's table, it was required for it to rotate around

90 degrees to get back to its dock. This feature was realized by adding a heuristic to rotate in our PD Controller move base plugin.

4.7. Recovery Behaviors

Almost every time the recovery policy described in the previous section failed to recover our robot from entrapment. Therefore drawing inspiration from the same, we devised our customized recovery behaviours for some specific cases.

Recovery Behavior 1: When a robot goes past near the obstacle and if it receives an immediate command to rotate, regardless of whether there's the obstacle, due to lack of full range coverage from the lidar, it fails to update the position of the obstacle. Such behavior can be seen in figure. We wrote our custom recovery behavior to detect such corner cases and clear costmaps if such cases are encountered.

Recovery Behavior 2: After the robot delivers food to customers, sometimes, due to imperfect plan execution, the robot's footprint overlaps with the static obstacle. In such cases all recovery policies described in the previous sections fail because as soon as the costmap clears the costmap, static obstacles are detected in the next costmap update loop. Additionally, since the footprint overlaps with the table, it cannot perform inplace rotation either. See figure to get an idea on how this case might occur with our mechanical design. We solved this function by adding a logic to detect such conditions and by backing off by 0.2m and running our updated custom recovery policies.

5. Conclusion

Long-term deployment reliability is one of the most crucial parts of any feasible autonomous service robots. This paper discussed Ginger's design and summarized lessons we learned from its deployment. Just like any engineered system, failures are unavoidable in robotics. The most significant challenge of robots is still human beings. As mentioned in the previous papers on similar subject, and from our observation, people tried to block the path of the robot. We considered many corner cases and improved our model from vanilla ROS navigation stack implementation. Our future goal is to improve GingerBot to meet and exceed the expectations of its users and capture a global market of waiter robots.

References

[1] Morgan Quigley and Brian Gerkey and Ken Conley and Josh Faust and Tully Foote and Jeremy Leibs and Eric Berger and Rob Wheeler and Andrew Ng, title="ROS: an open-source Robot Operating System", booktitle="Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics", month = may, year=2009, address="Kobe, Japan"

[2] ROSWiki, title = ROS Navigation Stack, year = 2020, url = http://wiki.ros.org/navigation, urldate = 2020-09-12

[3] A Roadmap for US Robotics: From Internet to Robotics, `http://www.hichristensen.com/pdf/roadmap-2020.pdf` year=2020

[4] S. Wang and H. I. Christensen, booktitle=2018 27th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN), title=TritonBot: First Lessons Learned from Deployment of a Long-Term Autonomy Tour Guide Robot, year=2018, pages=158-165,

[5] David Paz and Po-Jung Lai and Sumukha Harish and Hengyuan Zhang and Nathan Chan and Chun Hu and Sumit Binnani and Henrik Christensen, title = Lessons Learned From Deploying Autonomous Vehicles at UC San Diego, howpublished = EasyChair Preprint no. 1295, year = EasyChair, 2019

[6] R. Gockley and A. Bruce and J. Forlizzi and M. Michalowski and A. Mundell and S. Rosenthal and B. Sellner and R. Simmons and K. Snipes and A. C. Schultz and Jue Wang, booktitle=2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, title=Designing robots for long-term social interaction, year=2005, pages=1338-1343,

[7] Burgard, Wolfram and Cremers, Armin B. and Fox, Dieter and Hähnel, Dirk and Lakemeyer, Gerhard and Schulz, Dirk and Steiner, Walter and Thrun, Sebastian, title = The Interactive Museum Tour-Guide Robot, year = 1998, isbn = 0262510987, publisher = American Association for Artificial Intelligence, address = USA, booktitle = Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence, pages = 11–18, numpages = 8, location = Madison, Wisconsin, USA, series = AAAI '98/IAAI '98

[8] O. A. I. Ramírez and H. Khambhaita and R. Chatila and M. Chetouani and R. Alami, booktitle=2016 25th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN), title=Robots learning how and where to approach people, year=2016,pages=347-353,

[9] T. Kanda and M. Shiomi and Z. Miyashita and H.

Ishiguro and N. Hagita, journal=IEEE Transactions on Robotics, title=A Communication Robot in a Shopping Mall, year=2010, volume=26, number=5, pages=897-913,