# Mobile Robot Navigation: Turtlebot

DIALLO, Alpha Oumar
oumardiallo636@gmail.com

NNODIM, Kenneth Ugo
kennethugochukwu1@gmail.com

June 5, 2019

**Abstract:** Robotics is reshaping the landscape of industries, the military, and even our homes. From disarming bombs to cleaning our homes. Navigation is a crucial role in helping robots carry out their task. Our aim in this project is to study the turtlebot robot and perform navigation tasks using ROS ( Robot Operating System) .

## 1 Introduction

The first time we go to a new place we either ask someone to direct us or we make use of map guidance system like google map to get to our destinations. But when we visit the same place more than once we start to realize that we don't need any external input(help) to get to the destination. How do we achieve this?, how can we make a robot achieve this same thing?. The answers to these questions are the building blocks of mobile robot navigation. Robot navigation can be defined as the ability of a robot to localize itself in its environment and then lay out a path it can take to arrive at a particular destination without help. To achieve this, many techniques have been developed and some are being improved. This article introduces the literature survey of the various techniques used for mobile robot navigation. Past surveys on this field have provided different categorization based on the techniques used, the scale, and the types of hardware used (sensors). As such this survey is built on top those, as we survey the different techniques we dive a little deeper on their fundamental details to provide a strong framework for a better understanding.

Three Components come together to offer a proper navigation system. The first one is map-building, the second is localization and the third path planning. Path planning is an important task in navigation, when a robot has a complete prior information of the environment this is known as global path planning. Many methods have been developed for global navigation, i.e. Voronoi graph [2,3], This class of planning performs poorly in an environment where unknown obstacles may be located on a prior planned path. The second path planning method is known as sensor-based local path planning it uses sensory information to navigate in an uncertain environment. Also known as obstacle avoidance it's an important method of navigation that many mobile robots implement.

Navigation can be categorized into three major part, Global navigation, Local Navigation, and Personal Navigation. The foundation of global navigation for a mobile robot which can be an Automated Guided Vehicle (AVG) is reliably gaining location information in relation to a fixed point on the globe. This point is generally taken as the intersection of the Greenwich meridian and the equator line of latitude. A list of waypoint can be generated to allow a vehicle to move from one place to the other. This mode of navigation is used by the Global Positioning System commonly known as (GPS)[4]. In Local navigation, the robots determine it's position relative to its surroundings. As such a fixed frame of reference must be provided for the robot, so that it can continuously refer to this frame using onboard sensors, and therefore know its precise position in the surroundings. The last mode of navigation is when the robot is aware of the positions of its various parts[5].

## 2 Literature Review

Generally, two localization methods for mobile robots are available, relative positioning and absolute positioning, these types of positioning allows to categorize the different types of navigation that exist thus far. In relative positioning odometry also referred to as dead reckoning and inertial navigation are often used to calculate the robot position estimates from a start reference point. In contrast, absolute positioning relies on detecting and recognizing different features in the robot environment in order for a mobile robot to reach a destination and implement the specified task. These environmental features are normally divided into two types: one is feature-based, such as active beacons, artificial and natural landmarks[8]. Another is map-based such as different geometrical models.

Feature-based navigation such as active beacons is used in both global scale navigation and local navigation based on the beacon used. Radio Frequency (RF) beacons are useful for global scale navigation, due to the greater propagation distances available, and less suitable for indoor use. Where multipath reflections from walls can introduce large inaccuracies. In local navigation using the mobile robot, laser, sonar, and radio (or microwave) are common media for navigational beacons. As we will discuss in subsequent sections, most sensors used for the purpose of map building involves some kind of distance measurement which relies on these type of sensors. Our topic of interest in the project however is the Map based navigation.

In this Navigation mode, a mobile robots uses a map to localize itself. This map is built using sensor information; it could be previously stored in its memory or it can be built by the robot as it scans the environment. Different sensors can be used to build the map. Visual-based navigation can build a map using a camera as its sensor. A map can also be built by using a radar rangefinder sensor, this technique is commonly known as active beacon sensing.

Most active beacon methods are ”line-of-sight” dependant. Which means there's no obstacle between the mobile user and the beacons[4]. There are two principal methods of determining the user's position either by triangulation or by trilateration. Most sensors used in map building involves distance measurement. The range can be measured in three different ways. The first one is *time of flight* (TOF), the second is *phase-shift measurement* and the last one is *sensor based on Frequency-modulated FM radar.*

The combination of different sensors has also been employed in some approaches to increase the robustness and reliability of the map building procedure. In[11] range finders and cameras work in collaboration to create occupancy grids ( represents an observed region and cell of the grid is labeled with the probability of being occupied by an object). After acquiring the map the robot can navigate in the environment, matching the landmarks found in the online image with the expected landmarks. The process is known as self-localization

Nonreactive systems tend to incorporate map building and self-localization. In map-building standard approaches, it is assumed that the localization in the environment can be computed by some other techniques, while in pure localization approaches, the map of the environment is presumably available. Robots using this navigation approach need to track their own position and orientation in the environment in a continuous way. On the flip side there exist also Metric maps building methods which include information such as distances or map cell sizes with respect to a predefined coordinate system, and, in general, are also more sensitive to sensor errors[12].

If the exploration and mapping of an unknown environment is done automatically and on-line, the robot must accomplish three tasks: safe exploration/navigation, mapping and localization, preferably in a simultaneous way. Simultaneous Localization and Mapping (SLAM) and Concurrent Mapping and Localization (CML) techniques search for strategies to explore, map and self-localize simultaneously in unknown environments[12].

## 3 Objective

Our goal is to design a robot navigation system that allows the turtlebot to move from place A to place B while avoiding obstacles, without human assistance through the help of the Ros navigation stack.

## 4 Methodology

To achieve our goal we will create static map of the environment, localize the robot within the environment, make the robots perform path planning, visualize data of the different Navigation processes and debug errors using Rviz, configure the different Navigation nodes. Building the map is done by using the *the SLAM gmapping package* which also provides different packages for saving the map to our local machine and also providing the map to other nodes.

## 5 Implementation

### 5.1 Mapping

In order to perform autonomous Navigation, the robot MUST have a map of the environment. The robot will use this map for many things such as planning trajectories, avoiding obstacles, etc. A prebuilt map of the environment can be given to the robot or the robot can start by building the map first before providing it to the other nodes.

Simultaneous Localization and Mapping (SLAM). This is the name that defines the robotic problem of building a map of an unknown environment while simultaneously keeping track of the robot's location on the map that is being built. We need to use SLAM in order to create a Map of the Robot. The gmapping ROS package is an implementation of a specific SLAM algorithm called gmapping. This package contains a node called **slam_gmapping** which allows the creation of a 2D map using the laser and pose data that the robot is providing while moving around an environment. Basically the node reads data from the laser and the transforms of the robot, and turns it into an occupancy grid map (OGM). Another of the packages available in the ROS Navigation Stack is the map_server package. This package provides the *map_saver* node, which allows us to access the map data from a ROS Service, and save it into a file. The following command performs just that.

```
$ rosrun map_server map_server -f laser_map
```

This command will get the map data from the map topic, and write it out into 2 files, laser_map.pgm and laser_map.yaml.

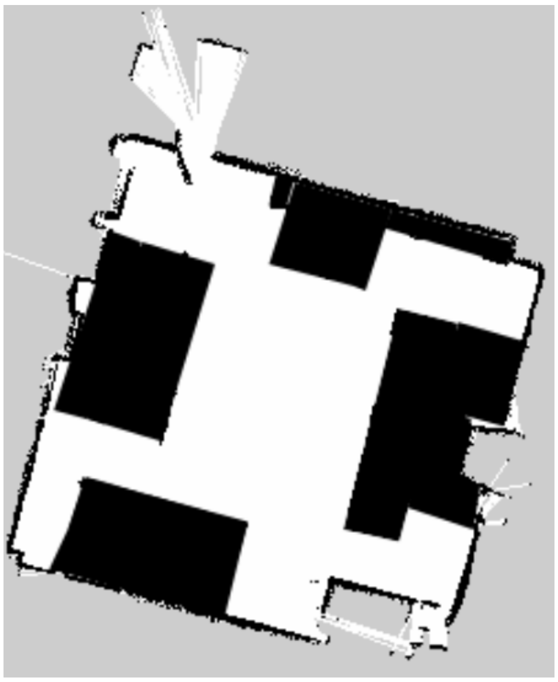


Figure 1: *laser_map.pgm*

A PGM (Portable Gray Map) file. A PGM is a file that represents a grayscale image. Whiter pixels represents free, blacker pixels are occupied and pixels in between are unknown.

The map_server package also provides the **map_server node** this node reads a map from a file then provides it to other nodes like the **move_node** for performing path planning or the **amcl** for localization. We can have access to the map by either calling the service **static_map** or by listening to the **/map** latched topic. When a topic is latched, it means that the last message published to that topic will be saved. That means, any node that listens to this topic in the future will get this last message, even if nobody is publishing to this topic anymore.

$ **rosrun map_server map_server laser_map.yaml**

It is important to note that the map created is a static map. This means that the map will always stay as it was when it was created. Also the map is a 2D map. This means that, the obstacles that appear on the map don't have height. Another important topic is configuration. configuration is VERY IMPORTANT in order to build a proper Map. Without a good configuration of the robot, we will end up with a bad Map of the environment. And without a good Map of the environment, the robot will not Navigate properly. In order to build a proper map, two requirements needs to be fulfilled. the first is to provide a Good Laser Data, the second is to provide good Odometry data.

### Transforms

In order to be able to use the laser readings, we need to set a transform between the laser and the robot base, and add it to the transform tree. This basically tell the Robot Where (Position and Orientation) the laser is mounted on the robot and it is refered to as transform between frames. A transform specifies how data expressed in a frame can be transformed into a different frame. This relationship

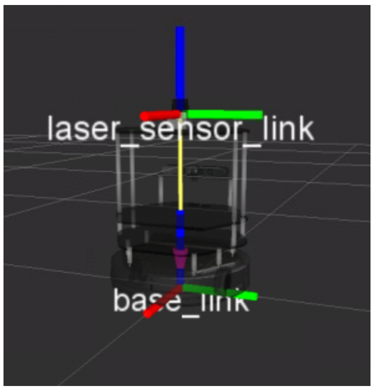


Figure 2: */laser_fram to /base_link transfrom*

between the position of the laser and the base of the robot is known in ROS as the TRANSFORM between the laser and the robot. The **slam_gmapping** node needs two transforms.

1. the /laser_frame to /base_link
2. /base_link to /odom

### Visualize Mapping in RVIZ

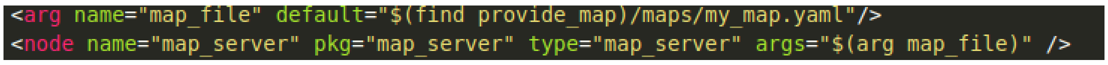
For Mapping, we basically need to use three displays of RViz:

1. LaserScan Display
2. Map Display
3. Robot Model

## 6 Localization

When a robot moves around a map, it needs to know which is its POSITION within the map, and which is its ORIENTATION this is known as the pose of the robot. Determining the pose using the robots sensors is known as **Robot Localization**. In this section we will present the way we configured the turtlebot for a better localization. The AMCL (Adaptive Monte Carlo Localization) package provides the amcl node, which uses the MCL system in order to track the localization of a robot moving in a 2D space, which generates many random guesses as to where it is going to move next. These guesses are known as particles. Each particle contains a full description of a possible future pose. When the robot observes the environment it's in (via sensor readings), it discards particles that don't match with these readings, and generates more particles close to those that look more probable. The amcl node reads the data published into the laser topic (/scan), the map topic(/map), and the transform topic (/tf), and published the estimated pose where the robot was in to the /amcl_pose and the /particlecloud topics.

In the previous section we talked about how the map_server node is used to provide the map. So once it has been provided the amcl node makes use of that map. We achieved this by calling the map_server node into the amcl launch file. The code for that is as follows:



```
<arg name="map_file" default="$(find provide_map)/maps/my_map.yaml"/>
<node name="map_server" pkg="map_server" type="map_server" args="$(arg map_file)" />
```

Figure 3: *Providing map*

Configuration is also very important to properly localizing the robot in the environment. In order to get a proper Robot localization, we fullfilled 3 basic requirements:

1. Provide Good Laser Data
2. Provide Good Odometry Data
3. Provide Good Laser-Map Data

As discussed previously transform plays an important role in navigation. the amcl node transforms incoming laser scans to the odometry frame. So there must be a path through the transform tree from the frame in which the laser scans are published to the odometry frame.

The amcl node provides a service called global_localization in order to restart the positions of the particles, and is able to make use of a service called static_map in order to get the map data it needs. The textbfglobal_localization service initiate globlocalization wherein all particles are dispersed randomly throughout free space in the map. We will make use of this feature to predict the localization of the robot in one of our packages. To have a more acurate localization we used RVIZ to estimate a pose of the robot then we used the get_pose topic to get the pose of the robot then hardcoded that pose so that anytime the navigation process start the robot is well localized.

## 6.1 Visualize Localization

Localization can be visualized in RVIZ by adding the following displays:

1. LaserScan Display
2. Map Display
3. PoseArray Display

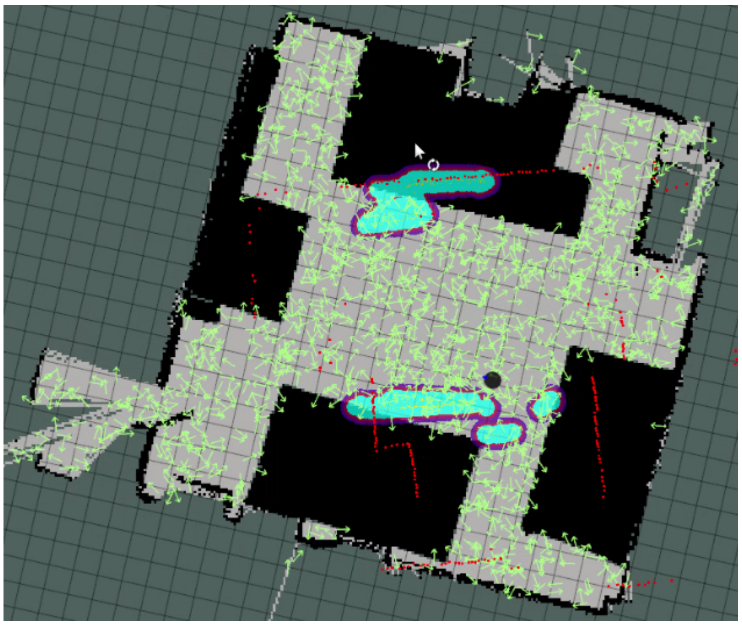


Figure 4: *Global Localization*

## 7 Path Planning

In this section we will look at what it takes to move the robot from point A to point B. As it's a custom to not reinvent the wheel we will use an exiting package to implement this task. The package that allows this behavior is called **The Move_base package**, it is part of the Ros navigation stack just as all the packages discuss thus far. The package has a node called move_base which main function is to move the robot from its current position to a goal position. The node is an implementation of a **SimpleActionServer**. the action Server provides the topic *move_base/goal* which is the input of the navigation. stack. This topic is then used to provide the goal pose. When this node receives a goal pose, it links components such as **the global planner, local planner, recovery behavior and costmaps** and generates an output which is a velocity command and send it to the /cmd_vel topic in order to move the robot.

### 7.1 The Global Planner

A goal received by the move_base node is immediately sent to the global planner, which is in charge of calculating a safe path in order to arrive at that goal pose. This path is calculated before the robot starts moving, as such it will not take into account the readings of the sensor of the robot as it moves, it then publishes the found created path to the /plan topic.

The planning of a trajectory by the global planner is done according to a map called the **The Global Costmap**. A costmap is basically a map that represents places that are safe for the robot to be in a grid of cells. Usually a costmap has binary valies, representing either the space or places where the robot would be in collision. But to be more specific each cell in a costmap has an integer value in the range (0, 255). Amoung these values there are special values frequently used in this range.

1. **255** reserved for cells where not enough information is known
2. **254** indicates that a collision causing obstacles was sensed in this cell
3. **253** indicates no obstacle, but moving the center of the robot to this location will result in a collision
4. **0** cells where there are no obstacles and, therefore moving the center of the robot to this position will not result in a collision

.

The global planner uses the global costmap in order to calculate the path to follow

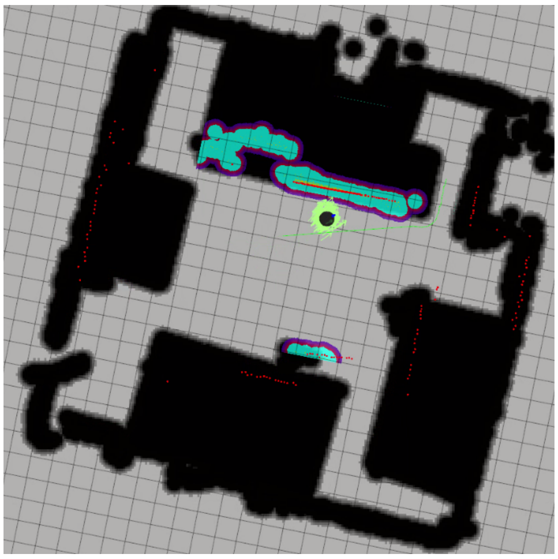


Figure 5: *Global Costmap*

The global costmap also has its own parameters, which are defined in a YAML file. The most important that we used are the following:

1. **global_frame**: the global frame for the costmap to operate in.
2. **static_map**: Whether or not to use a static map to initialize the costmap.
3. **rolling_window**: whether or not to use a rolling window version of the costamp. if the static_map is set to true, this parameter must be set to false.

To visualize the global path planning using RVIZ use the **path** display and listening to the topic
**path**. Also we can visualize the global costmap using the **Map** display and listening to global costamp topic.

## 7.2 The local planner

Once the previously discussed global planner has calculated the path to follow this path is sent to the local planner. The local planner, then will execute each segment of the global plan (let's imagine the local plan as a smaller part of the global plan). So given a plan to follow and a map, the local planner will provide velocity commands in order to move the robot. the local planner can recompute the robot's path on the fly in order to keep the robot from striking objects, yet still allowing it to reach its destination. Different types of local planners also exist to choose from depending on the setup like The robot used, the environment etc., and the type of performance wanted. This project makes use of the **base_local_planner**. It provides implementation of the trajectory rollout and Dynamic Window Approach (DWA) algorithms in order to calculate and execute a plan. Let summarize the basic idea of how this algorithms works:

1. Discretely sample the robots control space
2. For each sampled velocity, perform forward simulations from the robot's current state to predict what would happen if the sampled velocity was applied.
3. Evaluate each trajectory resulting from the forward simulation.
4. Discard illegal trajectories.
5. Pick the highest-scoring trajector and send the associated velocity to the mobile base.
6. Rinse and Reapeat

The local planner also has its own parameters. These parameters will be different depending on the local planner you use. These parameters are set in a YAML file they include **Robot Configuration Parameter, Goal Tolerance parameters, Forward simulation parameters** The list of all these parameters can be found on the move_base documentation site.

We previously spoke about the global costmap as being used by the global planner in the same way, the local planner also makes use of a costmap called the local costmap, in order to calculate local plans. It is important to Note that the obstacle layer uses different plugins for the local costmap and the global costmap. For the local costmap, it uses the costmap_2d::ObstacleLayer, and for the global costmap it uses the costmap_2d::VoxelLayer. The obstacle layer is in charge of the marking and clearing operations. There is one last parameter files related to both the global and local costmaps that we did not discuss, that is the common costmap parameter file. It configuration is used for both the previously mentioned costmaps.

## 7.3 Recovery Behaviors

It could happen that while trying to perform a trajectory, the robot gets stuck for some reason. Fortunately, if this happens, the ROS Navigation Stack provides methods that can help your robot to get unstuck and continue navigating. These are the recovery behaviors. We made use of the two recovery behaviors provided by the Ros navigation stack. The Rotate Recovery behavior attempts to clear out space by rotating the robot 360 degrees. The other one is the Clear Costmap recovery, this behavior simply clears out space by clearing obstacles outside of a specified region from the robot's map. Basically, the local costmap reverts to the same state as the global costmap.

## 8 Conclusion

In order to navigate around a map autonomously, a robot needs to be able to localize itself into the map. And this is precisely the functionality that the amcl node (of the amcl package) provides us. In order to achieve this, the amcl node uses the MCL (Monte Carlo Localization) algorithm. The Robot can't navigate without a map. But, surely, the robot can't navigate if it isn't able to localize itself in it either. So, the localization process is another key part of ROS Navigation. Bascially, the amcl node takes data from the laser and the odometry of the robot, and also from the map of the environment, and outputs an estimated pose of the robot. The more the robot moves around the environment, the more data the localization system will get, so the more precise the estimated pose it returns will be. After getting the current position of the robot, we can send a goal position to the move_base node. This node will then send this goal position to a global planner which will plan a path from the current robot position to the goal position. This plan is in respect to the global costmap, which is feeding from the map server. The global planner will then send this path to the local planner, which executes each segment of the global plan. The local planner gets the odometry and the laser data values and finds a collision-free local plan for the robot. The local planner is associated with the local costmap, which can monitor the obstacle(s) around the robot. The local planner generates the velocity commands and sends them to the base controller. The robot base controller will then convert these commands into real robot movement. If the robot is stuck somewhere, the recovery behavior nodes, such as the clear costmap recovery or rotate recovery, will be called.

## 9 References

[1] Wikipedia Contributors (2019) *Robot navigation*, Wikipedia. Wikimedia Foundation. Available at: https://en.wikipedia.org/wiki/Robot_navigation. [Accessed 22 March 2019]
[2] Takahashi O, Schilling RJ (1989) *Motion Planning in a Plane Using Generalized Voronoi Diagrams.* IEEE Robotics and Automation, 5(2):143-150.
[3] Bhattacharya P, Gavrilova ML (2008) *Roadmap-Based Path PlanningUsing the Voronoi Diagram for a Clearance-Based Shortest Path.* IEEE Robotics and Automation 15(2): 58-66.
[4]Jonathan D. (1997) *Mobile Robot Navigation An Overview of Global Mobile Robot Navigation: Global Positioning* [online] pages available at: https://www.doc.ic.ac.uk/ nd/surprise_97/jo
[Accessed: 24 March 2019]
[5] Jonathan D. (1997) Mobile Robot Navigation final report [online] pages available at: https://www.doc.ic.ac.uk/ nd/surprise_97/journal/vol4/jmd/ [Accessed: 24 March 2019]
[6] https://www.gps.gov/systems/gps/performance/accuracy/
[7] L. Matthies, E. Gat, R. Harrison, B. Wilcox, R. Volpe, and T. Litwin (1995). Mars Microrover Navigation: Performance Evaluation and Enhancement. In Proc. of IEEE Int'l Conf. of Intelligent Robots and Systems (IROS), volume 1, pages 433–440.
[8] Huosheng H. and Dongbing G.(1999) *Landmark-based Navigation of Mobile Robots in Manufacturing* [online] Barcelona:p114-121 available at: https://pdfs.semanticscholar.org/1bab/3f0
[Accessed 25 march 2019] [9] B. Nickerson et. (1991). The ARK (Autonomous Robot for a Known environment) project. -1. 76-77.
[10] G.N. DeSouza and A.C. Kak. Vision for Mobile Robot Navigation : A Survey. IEEE Transactions on Pattern Analysis and Machine Intelligence, 24(2):237–267, 2002.
[11] R. Chatila and J.P. Laumond. Position Referencing and Consistent World Modeling for Mobile Robots. In Proc. of IEEE Int'l Conf. on Robotics and Automation (ICRA), pages 138–145, 1985
[12]Francisco Bonin-Font, Alberto Ortiz and Gabriel Oliver. Visual Navigation for Mobile Robots: a Survey [13] http://wiki.ros.org/navigation [14] http://wiki.ros.org/amcl
[15] http://wiki.ros.org/move_base