A Robotic Approach to Clean Beaches

Christian Vajgel
Projeto de Bloco: Sistemas Robóticos
ESTI – Computer Engineering
Instituto Infnet
Rio de Janeiro, Brazil
christian.vajgel@al.infnet.edu.br

***Abstract*— With a day after day growth of garbage production in the world together with a bad habit of throwing anything on the floor, caused by lack of education, everyplace the humans are present are dirty – streets, beaches, forests – and takes a lot of effort to clean all this mess. Beaches are mainly the biggest problem which conscient people faces on daily basis. Finding a way to deploy an automated solution for cleaning an environment and on this case the beach is the key point for conducting this research. A simple tracked robot containing sensors, a 360-degree flexible robotic arm and a basket was designed to accomplish it.**

Keywords—Robotics, ROS, TurtleBot3, Cleaning, Tracked

I. Introduction

According to Instituto Mar Urbano [1], 80% of the garbage founded on the sea are originated on the beaches. Every year 325.000 tons of plastic ends up floating on the sea of Brazilian's coast. This pollution reflects on a huge prejudicial crisis to the sea life and their habitat that sums up to other environmental problems of the world.

Another study from WWF Australia [2] explains how much time in years some materials we found threw away on the sand or floating on the sea normally takes to decompose. For example, only one material, plastic. A bottle of plastic left on the beach today will take 450 (four hundred and fifty) years to decompose, it is enough time to live four and a half generation of a family.

A video from Instituto Mar Urbano provided by ABC News [3] sadly shows a wave of plastic hitting the beach of São Conrado in Rio de Janeiro, a tourist and wealthy location of the "*marvelous city*".

The result expected is to build a robot to clean on this case the beach at the end of the day on an autonomous way: looking for trash on the sand with its sensors, moving next to it, picking-up with a claw and putting on its basket.

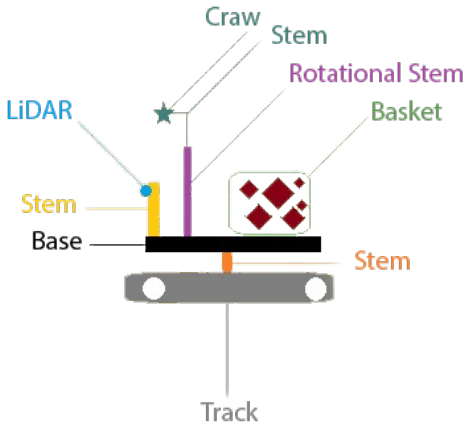
This paper is only focused on the navigation part of the cleaner robot, thus making a robot move to a given coordinate provided by the user. The robot model used will be the widely documented and easy-to-use *TurtleBot3 Burger* due to the abstraction of hardware's lower layers and its integration with ROS.

Although it will be only treated here the locomotion of the robotic unit, it is expected that in the future it could be integrated to other parts and make a physical robot to clean any type of environment, not only the beaches.

Honestly, this is a solution I did not want to deploy ever. Most part of the population has not the awareness of this real-world problem, they will keep ignoring it until there is no turning back. Hopefully, there is people trying to keep the environment clean.

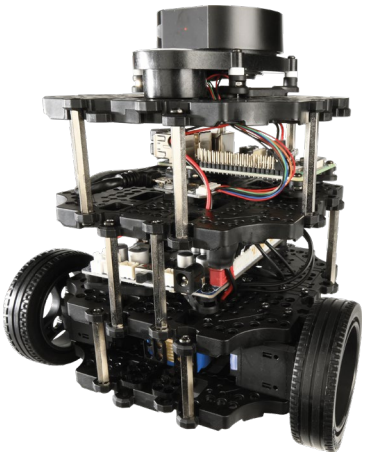
II. Robot

A. The whole model



The complete project mainly consists of a built model with a track for locomotion, LiDAR sensor for locating trash, a 360-degree flexible robotic arm with craw for collecting it and a basket for storing the garbage.

B. The navigation model



For the navigation scope it was previously chosen amid *Robotis*' portfolio the *TurtleBot3 Burger* model. With a solid hardware and functions such as 360º LiDAR for SLAM and navigation, Raspberry Pi Camera for perception, sprocket wheels and great capacity for maneuverability. This model also provides scalable structure for future developments and possibly the continuing of the project.

The robot also has Navigation Stack, a feature that allows to send a goal for the robot, without this tool it would be needed to manually handle the locomotion of the Turtlebot3, on this case he calculates the best route, speed and autonomously go to the destination.

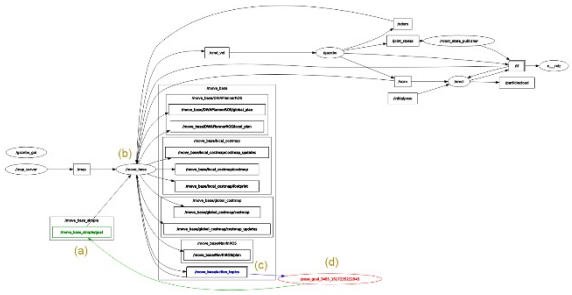
III. DEVELOPMENT

Navigation code:

- Robot receives the coordinates;
- Robot moves to the destination;
  - Robot check if goal was reached;
- Robot sleeps for five seconds;
- The routine loop starts until the end of provided coordinates;

The coordinates are calculated by the input number of beach tracks by the user. It was pre-determined that standard beach quadrant has an area of 10 x 10 meters and only 6 x 6 meters are used for navigation due to the local cost area, then each beach track has the width in meters of six divided by the number of inputted tracks. Robot's X axis is pre-fixed on -3.0 or 3.0 in relation with the map depending on the orientation of the movement. Robot's movement logic is explained later on this section.

In relation with the five seconds sleep state of the robot, this was intended to be the time period the trash collecting process is being executed. It was chosen to be settled on the extreme points of the circuit with a random generated position on Z axis referred as the orientation of the garbage on the sand which the supposed LiDAR on the complete model scanned, the robot navigated to and collected it. Although on the full functional model the robot keeps moving along through the beach tracks and collecting them while the trash is on its LiDAR's range field.



Project's ROS nodes [4]

The *pose_goal node* sends geometry messages of the type *PoseStamped* to *move_base_simple/goal* node containing data for the next pose the robot must go. The complete message has a frame id, X and Y positions and a quaternion. There is also a subscriber listening to *move_base/result* receiving move base messages of the type *MoveBaseActionResult* with the status of robot's pose.

Summarizing the graphic, the request to move is sent on *move_base_simple/goal* (a), this topic send another message to the *move_base* (b) which send a message to the velocity controller node to adjust the linear or angular speed and on meanwhile the *move_base/action_topics* (c) send a status message back to *move_base* (b) informing if the robot has reached its destination or not and publishes the goal coordinates on the anonymous topic (d). This flux continues to loop until ROS is shutting down or the robot arrives at the destination point.

Robot's navigation code is divided in three parts: goal, movement and thread sleep.

Before starting the main part, *pose_goal* is started as anonymous node and six global variables are pre-defined:

- *status*

Integer value which can be 0, 1, 2 or 3 and on this case the number "3" is what the code looks for representing the number of the message which shows the robot reached destination.

- *next_y*

Float point value representing the next position on Y axis it will feed one of the parameters on *talker_main* function and technically the next position on Y axis the robot will assume.

- *previous_y*

Float point value which receives the last valid position value of Y axis to be used as parameter to calculate the *next_y* value on the *define_next_y* function.

- *up_down*

Boolean value which indicates if robot's position on X axis, *true* means the robot is turned to up direction and *false* means the opposite.

- *needs_new_y*

Boolean value determining if the robot needs to make a turn on its next position and used as decision make parameter on *define_next_y* function.

- *tracks*

Integer value that holds the inputted number of beach tracks entered by the user.

- *track_width_y*

Float point value that is calculated based on the width of the quadrant secure area (6m) divided by number of tracks provided by the user. It is the value which is incremented to *next_y* variable on *define_next_y* funtion that calculates the position on Y axis.

On the main part of the script global variables are called and referenced, if this action was not done Python will see the variables only on local scope, thus crashing the code on execution time. User inputs the number of desired beach tracks and *define_track_width* method is called to calculate the width of each track. A sequence of sleep for three seconds followed by calling the method *talker_main* which moves the robot is called to slow down the execution of the code and set the robot 0.1 meter bellow the initial pose is sent to the compiler to position the robot for starting.

The for loop is where the code really start to act and is stated by starting with a *discard* variable holding the value firstly with "1" and increment by one on each repetition in a range of tracks multiplied by two then added two on the end, this is necessary to complete the rectangle draw of each track on the map with the chosen number of tracks.

The value for X is settled to 0.0 for later definition, Y receives the *next_y* variable value and Z, as previously stated, is randomly chosen to simulates the trash orientation on the sand.

The first position is the only one that is manually defined on the code this was made to make code's logic flow works

correctly. A little push is made on the robot through robot's locomotion function *talker_main* and then the actual values of the second position is provided, the Boolean variables responsible for the direction and Y position are also changed.

The code continues and *previously_y* variable receives the value of the last valid Y position, *talker_main* function is fed with the correct X, Y and Z parameters and the robot start to go to the new pose. A five seconds sleep is sent to the thread to give the robot time to reach destination and avoid pose conflict on real time execution. A while loop is used to check if the status variable has not the value "3" – robot reached the goal – and inside of it there is a subscriber listening to this topic, decision maker structure *if* is used to break the code if this value equals to "3". Finally, it is needed to define the *next_y* value for Y coordinate on the next iteration and *define_next_y* method is called once again.

On the other positions iterations, two *elif* decision maker structures defines if a number is even or odd and based on the result the value of X coordinate is settled positively or negatively representing the up or down direction of the goal and the global Booleans variables responsible for changing direction and the need of a new Y coordinate value is settled on each iteration. The logic flow mentioned on the first position explanation starts once again, setting the Y coordinate, moving the robot, subscribing to *status* node and the thread sleeps for the five seconds again representing trash collecting on the beach of the complete model. This routine is executed until the completion of the number of tracks provided by the user.

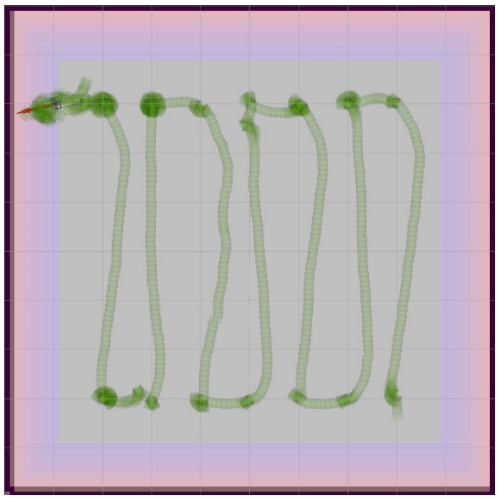
The function *talker_main* receives three parameters related to the coordinates the robot must go. A goal node is initiated to send the messages and settled to be anonymous to avoid conflicts with others nodes. Also a publisher with a 1/5 second rate is defined to send PoseStamped messages to the topic responsible to move the base of the robot. This object of type PoseStamped is created and has its attributes values declared. While ROS execution is not shutting down, this message is published on this base topic and it is expected to see the robot moving through the pre-defined routine.

A function named *listener_status* sets the value of the global variable *status*, responsible to hold the value of the robot movement status.

The function *define_next_y* is responsible for changing the value of the global float variable *next_y* used on the for loop on script's *execute_main* method. It is just a decision maker evaluating the state of the Boolean variable *needs_new_y* responsible to tell if the robot needs to change its Y coordinate and make a turn. If *true*, the *previous_y* has its value increased with the *track_width_y* float value which contains de width of each track previous calculated, otherwise, the value remains the same and on the next iteration it will be needed to make a turn and change the *next_y* value, consequently it is the value of Y coordinate on robot's pose.

Responsible for calculating each beach track width, the function *define_track_width* has access to the two global variables, the Integer tracks with the desired number of tracks chosen by the user and the track_*width_y* which will be lately settled on this method. The operation is simple, *track_width_y* variable receives the value of the width of the secure area of beach quadrant (6m), as previously mentioned, divided by the number of tracks. To avoid a massive float point number the result is rounded to one decimal case using Python built-in native math function of round and this operation has its interference despised on the result during execution time.

IV. TESTING



Robot's path with six tracks input [5]

Tests were made using TurtleBot3 Burger model on Gazebo empty world environment together with ROS and a custom map with 100 m² area. The empty world was chosen because the goal here is navigation and it was created this custom map simulating the quadrant area the robot must navigate until the end of the routine. Before starting the script, the user enters the value of desired tracks, respecting the rule: bigger the number smaller is the width of each track or smaller the number bigger will be the width of each track. Robot's initial pose is settled using RQT's interface.

The circuit was designed to simulate a beach work with the robot picking up a quadrant and defining tracks to move through the tracks, thus representing the navigation part and on the future this part can be added to a complete model that during this execution will also collect the trash and cleaning the area. When the robot stops it is being simulated the trash collecting for five seconds.

The cost map value was set to 1.5 to provide a precise locomotion but sometimes his navigation system gets confused and the robot locomotion pattern becomes a little shaky.

With a maximum linear speed of 0.22 m/s and angular speed of 2.84 m/s both speeds are automatically controlled by robot's controller and on this case we will take this subject as abstract.

It takes on average six minutes and thirty seconds to complete the circuit. When reached the destination point the robot exists the secure area to simulate the end of the circuit.

V. CONCLUSION

This project involved a lot of work to make the navigation part work properly. The robot is smoothly moving through the circuit with a good performance to exactly the destination points that was settled before.

There were a lot of difficulties through this journey but the expected result on navigation was reached. On the future it would be interest to see the robot receiving the area in square meters and calculating everything based on this value and on the number of beach tracks given by the user.

It is expected that this navigation model may be upgraded on a more autonomous way and composed with the rest of the tracked robot for cleaning any environment. The real problem was not resolved since TurtleBot3 Burger does not collect trash and it was only tested the navigation subject with a plain terrain and series of restrictions but at the end it worked.

REFERENCES

[1] INDEPENDENT, M. Mathers, "Wave of plastic washes up on Brazil beach", January 2021.
https://www.independent.co.uk/news/world/americas/plastic-beach-brazil-rio-de-janeiro-b1782743.html

[2] WWF Australia, "The lifecycle of plastics", June 2018.
https://www.wwf.org.au/news/blogs/the-lifecycle-of-plastics#gs.ty7ml4

[3] INSTITUTO MAR URBANO (ABC News), "Waves of plastic pollution spoil Brazilian beach", January 2021.
https://www.facebook.com/ABCNews/videos/480864016237901/

[4] VAJGEL Christian, "Project's ROS Nodes", March 2021.
https://i.imgur.com/z6xL8Tb.jpg

[5] VAJGEL Christian, "Robot's path", April 2021.
https://i.imgur.com/KMPb3JJ.jpg