POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Executive Summary of the Thesis

Simulation and Testing of Navigation for an Autonomous Mobile Robot

Laurea Magistrale in Automation and Control Engineering

Author: Giovanni Porcellato

Advisor: Prof. Matteo Matteucci

Co-advisor: Dr. Simone Mentasti

Academic year: 2021-2022

1. Introduction

Now far from being science fiction, robots are little by little entering our world. The tendency to replace humans with machines has existed for centuries, but it is only during the 20th century that technology has enabled us to achieve astonishing results.
Hence, the robots we know today come in different forms: from robots with human form, to mechanical arms, to robotic heads. In each case, there was extensive research and development before arriving at the results we can enjoy today. Given the initial instability of robotic systems, the danger and cost, the creation of these devices went hand in hand with the development of special simulators.
It is for these reasons that it's indeed very inconvenient to repeat tests on hardware. Precisely in this circumstance that arises the need to simulate in a virtual environment the movements and actions that the robot would then have to repeat in the real world. Moreover, it is necessary to set up a testing module aimed at collecting data from the real robot. Its purpose is to build statistics that are used as benchmark for internal development and for commercial use.

Another goal of testing platform is to isolate bugs that may arise from an extended use of the robot. During the tests infact, the robot appeared to suffer from high light conditions. It is presented a possible solution to the problem of reflective glares that were causing issues in navigation. The approach consisted in the development of an online statistical filter applied on the pointcloud data received from the image depth topic.

2. State of the Art

2.1. Architecture

The followig section is entitled to explore how Robee, Oversonic Robotic's humanoid, is organized and explain the technology it exploits. The architecture split between the robot (also referred to as the edge) and the cloud, and these two components coexist in a hybrid. Describing the system from the cloud, the hardware component consists of a scalable node pool based on the 2.35Ghz AMD EPYC 7452 processor that can achieve a boosted maximum frequency of 3.35GHz with 32 GB RAM memory, running a Kubernetes instance on top of Linux Ubuntu

18.04 (Bionic Beaver). As far as the robot is concerned, all the computational power is provided by 2 Intel NUCs 8 including an Intel Core i5-8259U Processor (6M Cache, up to 3.80 GHz), 8 GB RAM and Integrated Graphics Intel Iris Plus 655. The operating system which is mounted on is Linux Ubuntu 20.04 (Focal Fossa), and all the modules are running containers that on turn are managed by KubeEdge, a containers orchestration system built on Kubernetes.

2.2. AMCL

The pose estimation used in Oversonic applications is based on AMCL pose (Advanced Monte Carlo Localization). This technique relies on the so called particle filter approach. An extremely large number of particles covering the entire state space are used to initialize particle filters. The robot has a multi-modal posterior distribution because it predicts and update the measurements as it obtains more measurements. A Kalman Filter approximates the posterior distribution to be a Gaussian, which is a significant departure from this. The particles converge to a single value in state space after several rounds. Maintaining the random distribution of particles over the state space is a major challenge for particle filters, which becomes impossible for large dimensional problems. These factors make an adaptive particle filter far superior to a simple particle filter in terms of convergence speed and computational efficiency.

2.3. Pointcloud

Point cloud is a data structure used to represent a collection of multi-dimensional points and is commonly used to represent three-dimensional data. In a 3D point cloud, the points usually represent the X, Y, and Z geometric coordinates of an underlying sampled surface. When color information is present, the point cloud becomes 4D. Point clouds can be acquired from hardware sensors such as stereo cameras, 3D scanners, or time-of-flight cameras, or generated from a computer program synthetically, [4].

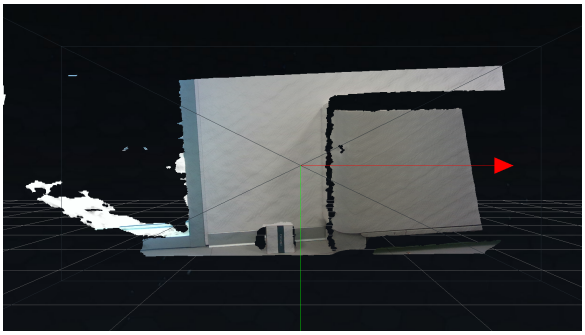

Figure 1: Pointcloud sample image

3. Simulator

The simulator is primarily intended to be a safe and reliable test environment where the new navigation algorithms can be tested. It is not intended to have a 1:1 simulation with the real robot, which is outside the scope of this thesis. The approach was to create a three-dimensional model , combined with a differential drive base, in URDF and XACRO that was as faithful as possible to the Oversonic autonomous mobile robot. The designed robot is made up of the following components:

- a core box, representing the structure of the robot
- two-wheel drive, one on the right and one on the left side
- two caster wheels, one in the front and one in the back. They are completely passive and the provide robot stability
- two Lidars, one place on the front and one on the back of the robot

A robot can be described as having a number of link components and a number of joint elements that connect the links. The link element describes a stiff body with collision characteristics, visual characteristics (specifying the shape of the object for visualization purposes), and inertia. The links used in this model were:

- Robot core: base link and dummy link
- Wheels: right and left wheel
- Caster wheels: right and left caster wheel
- Lidar: front and back lidar

The joint element sets the joint's safety restrictions and explains the kinematics and dynamics of the joint. There exists several types of joint, but in this specific application continuous joint was used for the casters and wheels, whereas fixed joint for the robot core and Lidar were used.

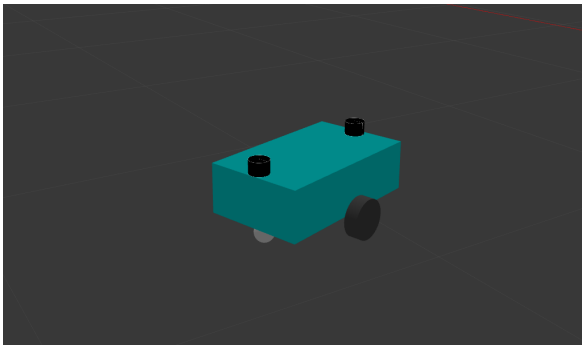

Figure 2: Robot Model

At this point the model is created but is no more than a mesh and cannot be used in navigation. In order to get the robot on track, it is necessary to duly integrate the model with a drive system, which mimics the physical motors, and lidar sensors. Gazebo provides plugins that meet this requirement. Two main plugins were used within the simulator: the differential_drive_controller for differential driving and the head_hokuyo_controller for lidar. In order to have the differential drive plug in working, a number of parameters must be set to bind it to the created XACRO model and to the already existing navigation stack. In particular, the plug-in needs to know the dimensions of the wheels and the joints to which they are connected, as well as the link of the robot body and the odometry topic. The speed command is acquired directly from the topic /twist_mux/cmd_vel, which is also used in the real robot. It is then necessary to specify the torque to be applied and the update rate. A similar argument must be made for the integration of the lidar sensor plug-in. It is necessary to specify all the topics that are normally used in the real robot to build a bridge between the simulated environment and the navigation stack. In particular, the topics used are /scanHF and /scanHB. It is also appropriate to modify the specifications of the simulated sensor to match those of YDlidar G4. In addition, the plug-in for the d455 camera and the t265 camera can also be imported. The simulated depth camera can be used for the recognition of tags placed in the virtual environment as well as for point-cloud generation. The simulated tracking camera, on the other hand, is used for visual odometry by replacing the default odometry source parameter of the differential drive with the topic of the tracking camera. next step was to create a virtual environment where the robot can move and where the navigation algorithms can be tested. The approach used was to replicate the Oversonic Robotics development space. After acquiring the measurements of the real environment with laser metre, the 1:1 scale measurements were transported to the Gazebo environment. Gazebo in fact allows the creation of rooms from scratch, building walls and doors with centimetre-accurate measurements. It also allows the insertion of obstacles and fiducial tags in the space.

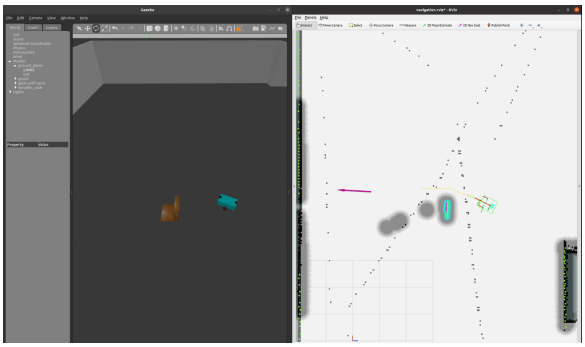

Figure 3: Obstacle Avoidance Simulated

4. Testing Platform

The testing module, described in section addresses the need to have an indication of navigation performance. This need comes both from within the company, where it is important to evaluate how developments are improving it, and from outside, where the various stakeholders and customers can evaluate the project. As reported by [1], among the many types of tests, two of them are significant:

- Reliability: obtain knowledge of failure occurrence patterns
- Performance: building reliable indicators on performance status

Tests are moreover meant to have the following features:

- Accuracy: whether the test can actually measure what it sets out to measure
- Resolution: how precisely it measures the proposed feature
- Repeatability: how repeatable it is, so that test performed over time are comparable

For the purpose of performance measurement, the following values of interest are given for each measurement configuration:

- AVG SPEED: average speed over the entire path [m/s]
- FW AVG SPEED: average speed while navigating (its the avg speed from which we exclude on-site rotations) [m/s]
- ROT AVG SPEED: rotational average speed [rad/s]
- NAV DISTANCE: total distance calculated by odometry [m]
- TOP SPEED: top speed recorded [m/s]
- NAV TIME: total time spent [s]
- FW MOVING TIME: moving forward time [s]
- ROTATING TIME: time the robot spends rotating in place [s]
- OBSTACLES: number of obstacles on the path [-]

The testing module is based on a callback structure. A ROS node called 'measures' is initialised and subscribed to three topics: goal_status, cmd_vel_mux_out and odom. We continue by explaining the concept of goal_status. In fact, it is necessary to know what status the robot is in in order to distinguish the various phases and obtain reliable statistics. The built in class of Oversonic `goal_status` arbitrates the management of statuses in a tree structure and decides to send the global status to the mission service. A global status, called goal_status arbitrates two sub-states: move_base_flex and motion_status. Arbitrage checks the statuses and the last updated one, if coincident, is sent to the mission service. For proper test handling, a subclass, called rel_movement_status, was then implemented. The states are defined as:

- Not started = 0
- Navigating = 1
- Rotating = 2
- Reversing = 3
- Stuck = 4

This class monitors the robot's movements and understands if it is advancing, rotating or if it is stuck. The status provided does not enter into the arbitration mentioned above, but is simply an aid to the collection of measurement data and does not communicate with the mission service. Only when both statuses are completed does the global status send the mission completion signal to the mission service via an MQTT message dispatcher. Cmd_vel_mux is a topic that exchanges messages regarding the velocity command: a multiplexer for command velocity inputs. The last topic is odom and simply provides the message regarding odometry but, unlike in the simulator, in the real robot the odometry data is provided by the T265 camera. At this point, the three topics are used within a callback where each time they communicate a message, data is collected and used to calculate statistics. Specifically the module not only provides overall statistics but also compute performance intra waypoints. This feature was introduced in order to get a better glance of where the robot suffered from issues or where navigation speed decreased. This was possible by managing goal status in a different manner: waypoints are sent to the robot via a MQTT message and a new waypoint is sent to the robot only when it arrives around a waypoint with customisable accuracy. So every time a new waypoint is delivered to the system, statistics are computed and saved. When eventually the robot reaches the goal, the overall statistics for the lap are computed and saved into a separated xls file. a small plotting module was developed. The task of this programme is to extrapolate the chosen map and waypoint coordinates from the database. A map is initialised in which the waypoints, identified by a number, and the goal are arranged. The linear path between the different waypoints is then plotted and on this the Euclidean distance between them is calculated. The minimum length that the path will have is then obtained, so as to have a reference with respect to the distance that the robot will travel in each lap. When the robot starts to move, two red and blue indicators, representing the amcl_pose and beacon_pose respectively, mark and keep track of the robot's movement on the map. This provides a tool that plots the robot's path in real time.

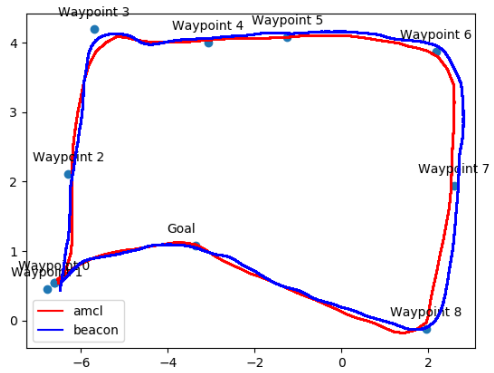

Figure 4: Real Time Plot Sample

Another addition that was made to the testing module concerns the measurement of currents during navigation. It is indeed important to have an estimate of the current peaks that occur, the hypothetical and actual consumption of the robot. For this purpose, therefore, a programme was created which subscribes to the topic motor_current in a callback and saves all data relating to the left and right motor. This is easily implementable due to the fact that the current message arrives in the form of a multi-array where at positions 0 and 1 are the current data for the left and right motor respectively.

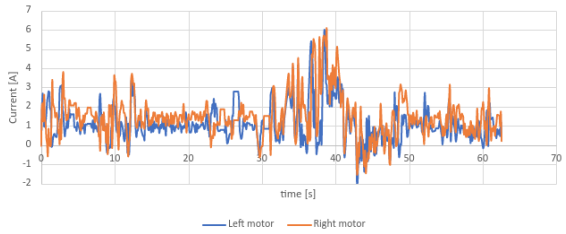

Figure 5: Motor Current Plot

5. Pointcloud Processer

During testing phase, it was realised that, in certain particular indoor and/or outdoor environment conditions, the robot perceived phantom obstacles, which did not exist in reality. This is critical as the objective for the robot's navigation is to perform the programmed path optimally and in the shortest possible time. It is therefore clear that if the local planner encounters these phantom obstacles, a great deal of time is lost and the robot travels a greater distance, also resulting in greater energy consumption. It was therefore decided to devote time to identifying and solving this problem. The conditions that encountered particular problems are:

- Reflective surfaces
- Tiled floor with highly repetitive patterns

An online post-processing solution was attempted. For this purpose, as already mentioned, the PCL library was used. This library provides methods for filtering pointcloud data. The approach was to treat ghost reflections as a mass of data too dense to be analysed and a statistical outlier removal filter was tried. In fact, the idea behind it is to remove noisy measurements from the pointcloud, as these scattered outliers corrupt the results. The solution therefore proposes statistical analysis around each point neighborhood: for every point compute the distance from the point itself to all of its neighbors. A similar approach was proposed by [2]: The sparse pointcloud P is defined by N points such that $P = p_1, p_2, p_3, ..., p_N$ whereas the k nearest neighbor points of a point p_i are defined as $KNN(p_i)$, a set of k points such that $Q = q_1, q_2, q_3, ..., q_k$. The algorithm removes points that are considered outliers based on geometric information.
The average geometric distance of a point p_i from its neighbors is defined as:

$$d_i = 1/k \sum_{j=1}^{k} dist(p_i, q_j) \quad (1)$$

where dist(x,y) is a function that returns euclidean distance between two points. By assuming that the resulted distribution is Gaussian with a mean and a standard deviation, we can derive μ_d the overall mean distance and σ_d, the standard deviation of distances. Hence, by tuning α, the so-called standard deviation multiplier, we can define a threshold $T = \mu_d + \alpha\sigma_d$. Every point in the cloud, whose its kNN distance falls out of the interval defined by T, are directly discarded.

Algorithm 1 Postprocesser

```
Set magnitude of decimation filter
Set k
Set α
Subscribe pointcloudtopic
Input_data: pointcloud topic
for P_i in input_data do
    Convert from PointCloud2 to PCL::cloud
    Locate kNN to point P_i
    Compute d_i
end for
Compute μ_d
Compute σ_d
Compute T = μ_d + ασ_d
if d_i > T then
    Trim point P_i from cloud
end if
Output_data = cloud
for P_i in cloud do
    Convert from PCL to PointCloud2
end for
publish pointcloud topic and pointcloud_filtered topic
```

The problem can be solved from another point of view that is by analyzing clusters of isolated pointcloud data, [3]. Local density function, $LD(p_i)$ of p_i is introduced as:

$$LD(p_i) = \frac{1}{k} \sum_{q_j \in KNN(p_i)} exp(\frac{-dist(p_i, q_j)}{d_i} \quad (2)$$

The probability that a point belongs to outlier can be defined as $prob(p_i) = 1 - LD(p_i)$, where $prob(p_i) = 1 - LD(p_i)$ and it ranges between 0 and 1, so that a higher value means a higher probability for the point to be outlier. The point p_i is retained if it falls under some predefined threshold, like T in previous approach, so $prob(p_i) < \delta$. The threshold is not intended to be universal, rather its value depends on its case of application. In order to assess the improvements made, particular attention should be paid to the Navigation and Stuck time values. These in fact come into play when the robot recognises obstacles to be overcome and consequently, increase in value. It can be seen that the initial configuration presents average navigation and stuck times of 54.48 s and 0.24 s respectively. By inserting the pointcloud filter with a filter power dictated by K=9, the values drop to 54.05 s and 0.104 s. Since the computation time still proved to be high, i.e. one tenth of a second, it was decided to try prefiltering the incoming data to the filter in order to lighten the amount of data to be analysed. The results were promising, leading to a browsing time of 53.1 s and a stuck time of 0.052 s, at the cost of an average computation time of 0.05 s. Furthermore, it should be noted that the tests were done in the office under not so unfavourable conditions, and that the light and distortion conditions of an industrial hall are much worse, so that 0.2 seconds could become much more. These differences may seem marginal, but when transported in the context of the use of many working hours, they take on a non-negligible importance. The inclusion of the pointcloud filter has also resulted in a more continuous navigation that allows the robot to reach its objectives in less time and avoiding discontinuities in movements that can accelerate wear and tear in the long run.

6. Conclusions

In the first part of this work, a simulator was developed from scratch to test new navigation algorithms. This made it possible to test the new custom recovery behaviour, for example, first in the simulated environment and once the goodness of the development was seen, it was implemented in the real robot. Consequently, a testing platform was developed that provides a comprehensive overview of performance measurements, motor current data and a robot trajectory visualiser. During testing, a problem was observed with the depth camera, which recognised reflections on the floor as obstacles. A real time filter was then implemented to decrease the density of the pointcloud data. Thanks to this, it was possible to solve the problem of phantom obstacles by decreasing the stuck time, in a computational time that does not affect the result. The simulator could be subject to further improvements in the future: one possible route that could be taken would be to break free from gazebo plug-ins, for example by modelling the differential drive on a par with that used in the real robot. Furthermore, in order to make the simulated model completely mirror the real one, another possibility that could be pursued would certainly be to analytically model friction and wheel slip. As far as the pointcloud

filter is concerned, other possible solutions can be tested: there are several filters for manipulating the point cloud, the results of which have yet to be tested.

7. Acknowledgements

Thanks to Oversonic Robotics, to Fabio Puglia for giving me the opportunity to work on such an ambitious project, to engineer Lorenzo Romanini for his support and to all the people I met who made my working days easier. I would also like to express my gratitude to Professor Matteo Matteucci, for his patience and his willingness to assist me in this thesis.

References

[1] B. S. Dhillon. *Robot Testing and Information Related to Robots*, pages 210–225. Springer New York, New York, NY, 1991.

[2] Xiaojuan Ning, Fan Li, Ge Tian, and Yinghui Wang. An efficient outlier removal method for scattered point cloud data. *PLoS ONE*, 13, 2018.

[3] Xiaojuan Ning, Fan Li, Ge Tian, and Yinghui Wang. An efficient outlier removal method for scattered point cloud data. *PLOS ONE*, 13(8):1–22, 08 2018.

[4] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.