Path planning using Matlab-ROS integration applied to mobile robots

Conference Paper · April 2017

DOI: 10.1109/ICARSC.2017.7964059

CITATIONS
7

READS
1,066

4 authors:

Marina Galli
University Carlos III de Madrid
5 PUBLICATIONS **7** CITATIONS
SEE PROFILE

Ramon Barber
University Carlos III de Madrid
67 PUBLICATIONS **468** CITATIONS
SEE PROFILE

Santiago Garrido
University Carlos III de Madrid
132 PUBLICATIONS **1,185** CITATIONS
SEE PROFILE

Luis E Moreno
University Carlos III de Madrid
266 PUBLICATIONS **2,181** CITATIONS
SEE PROFILE

2017 IEEE International Conference on
Autonomous Robot Systems and Competitions (ICARSC)
April 26-28, Coimbra, Portugal

Path planning using Matlab-ROS integration applied to mobile robots

M. Galli, R. Barber, S. Garrido, L. Moreno
Robotics Lab. Departamento de Ingeniería de Sistemas y Automática
Universidad Carlos III de Madrid
Madrid, Spain
Email: mgalli,rbarber,sgarrido,moreno@ing.uc3m.es

***Abstract*—In this paper the possibilities that Matlab provides to design, implementation and monitoring programs of autonomous navigation for mobile robots, on both simulated and real platforms, through its new toolbox for robotics will be explored.**

Robotics System Toolbox has established itself as a solid tool of integration between Matlab and robot operating under ROS environment. It will be studied the tools available in this toolbox that allow achieving the connection between the two platforms, in addition to the generation of algorithms of location, path planning, mapping and autonomous navigation.

I. Introduction

Software development for robots is becoming a process more and more difficult, mainly due to the increasing complexity of the tasks that are required for them to perform. Nowadays, many robots are specialized to perform very specific tasks, generating a variety of different hardware, which produce software and specific programming, that greatly difficult code reuse [16] [3].

As a result, to overcome these problems, different working environments were born to facilitate the development of robots. *Robot Operating System* (*ROS*) [10] is the leading development environment in robotics, and offers tools and libraries for the development robotic systems. In recent years ROS has gained wide currency for the creation of working robotic systems, not only in the laboratory but also in industry.

Path planning is a well-known problem with a well understood mathematical basis. From an initial point to a final point, the path planning objective was to create an algorithm able to find a path ensuring completeness.

This objective has been largely solved and computational capacity has increased exponentially, now the objective has become more challenging. It is to find the shortest or fastest path while maintaining safety constraints. These solutions are also expected to provide smooth, human-like paths.

Many different path planning algorithms were proposed. LaValle proposes a classification into two big groups depending on how the information is discretized [18]: Combinatorial Planning, which constructs structures which capture all the information needed in path planning [17], and Sampling-Based Planning, which incrementally searches in the space for a solution using a collision detection algorithm.

The main objective of this work is to study the viability and possibilities of this tool, as well as its usefulness. Various simple algorithms already used with ROS, would be used to verify the efficiency of communication taking advantage of the tools of data processing that Matlab provides. The aim is to perform the same actions that are performed in a ROS environment through the Matlab *Robotics System Toolbox*, that it will allows to take advantage of the ROS message structure in a Windows environment without using Linux.

II. Connection Matlab-ROS

Matlab is an important tool for designing prototypes and test robotic systems. However, until now the most common way to connect a robot working with ROS into other platforms for control and analysis was through bridges, in the case of Matlab bridge based in Java [1].

Robotics System Toolbox is a new toolbox, that was included for the first time on the R2015A Matlab's realese. A direct and complete integration between Matlab and Simulink and ROS is provided by the interface. The toolbox eneables to write, compile and execute code on ROS-eneable robot's and on robots simulators such as Gazebo, not only in Matlab's enviroment, but also it supports C++ code generation, allowing to generate ROS node from Simulinks model and implement it into the ROS network. [11]

The first step to work with this toolbox is to stablish connection with the robot. ROS allows the communication between different devices working for the same *Master*, Matlab will be performing as one node under the command of it. To achieve this connection the computer working with Matlab and the robot must be connected to the same network.

In order to initialize the Matlab-node the same structures used in ROS are needed, in this case with a simple *rosinit* the connection is fully accomplished. Not only the similarities with the structure are seen to initialize, but also in data communication publishers and subscribers are used. The main three subscribers used in this paper are odometry, laser and camera, and the subscriber to control the movement of the robot. [9]

III. Algorithms

Two different algorithms were tested with the toolbox to prove the real usefulness to develop and run code. PRM and

Fast Marching Square algorithms were chosen for being two of the most important navigation program for mobile robots.

A. PRM

PRM (*Probabilistic RoadMap*) is a motion planning algorithm used for answering path planning inquiries. Looking for a free collition path from a initial configuration to a goal point, it connects possible random configurations of the robot on the free configuration space *(C-space)* [7].

The algorithm consists of two stages:

Learning phase: On this phase, random configurations are created. Then it is checked if the robot collides or not in each configurations. This is a process repeated until a set of suitable samples of the desired size is obtained. In order to find a solution, it must always be added the initial and target positions to the sample set.

Query phase: The algorithm connects the possible configurations to the roadmap by simple paths (straight segment) building probabilistic roadmap and then searches the roadmap for a sequence of edges from one connecting node to the next one.

If the environment in which the robot is located is very complex, this type of algorithm allows finding a solution without a large computational requirement. Therefore, it is used in space with a large number of dimensions.

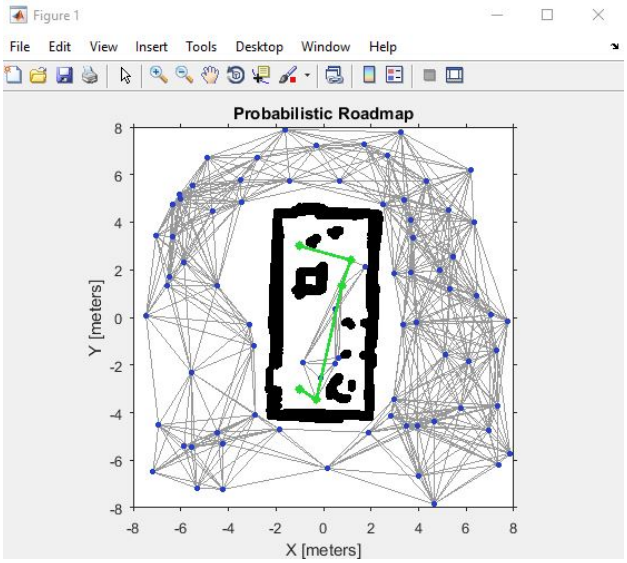


Fig. 1: PRM on robot-constructed-map using *Robotics System Toolbox*.

One of the main problems of PRM is that the solutions it finds do not have to be the optimal path. In addition, because the way of generating the nodes is completely random, it produces a non-homogeneous distribution of the samples in the space. These two disadvantages are seen in Fig.1 [8].

B. Fast Marching Square

Fast Marching Square (FM^2) is a path planning algorithm, which unlike PRM looks for the optimal path between two points.

FM^2 [6] could be considered to be based on the creation of potential fields, but it differs from this type of algorithm for taking into account the concept of time. Considering the robot as a particle that moves under the influence of time, the path chosen will be the one that employs as little time as possible to reach a goal.

It uses a function with behaviour is similar to the propagation of a wave in a fluid [19]. The way that the wave propagates, following the Eikonal Equation, from an initial point to reaching a target, is the most efficient way in terms of time to reach a goal point.

This wave is called the interface, the algorithm calculates the time (T) that the interface uses to reach each point of space starting from an initial point. FM2 starts from the discretization of the environment in a grid of cells on which will iteratively solve the equation of Eikonal.

Each one of this cell can be defined by one of the following statements:

Unknown: Cell where the wave has not passed yet, thus the value of T is not known.

Narrow Band: Set of cells where the wave will expand in the next iteration. These have a value of T that can be changed in successive iterations.

Frozen: All of cell that were touched by the wave, consequently, have a fixed value of T.

The algorithm consists of three stages:

- Initialization: The value $T = 0$ is set for the cells where the wave will be created, and they are labelled as *frozen*. Afterwards, the adjoining cell will be labelled as *narrow band*, and the time T elapsed till the wave arrives will be calculated.
- Main loop: The Eikonal Equation will be sequentially solved by cells labelled *narrow band* to be *frozen* when they store the shortest arrival time value T.
- Finalization: When all the cells are *frozen*, *narrow band* is empty, the algorithm finishes. [4]

The trajectories obtained have two drawbacks: great abruptness of turns and trajectories that get very close to the obstacles. Making it impossible to use the algorithm as a real robotic trajectory planner. In order to solve these problems a velocity map is generated and modifies the expansion of the wave taking into account the proximity to the obstacles [5].

IV. Experimental Results

In this section, the experimental data obtained will be commented. Several experiments were perform to tests the effectiveness of the algorithms, in which the robot must follow a fix trajectory or move through the environment and reach an objective position avoiding collision. Therefore, the concepts to be studied are:

- Achievement of the goal position.

- Execution time.
- Real path versus theoretical path.
- Movement fluidity.

All data have been taken in the laboratory of the Department of Systems Engineering and Automatic of Carlos III University , using the mobile robot Turtlebot.

1) Turtblebot: Turtlebot is a low-cost mobile robot based on the open-source software and hardware architecture philosophy. It is a robot especially indicated for the investigation and test of algorithms owing to allowing the access of information of lower level [22] [12].

In particular, the second version of this robot, Turtlebot 2, will be used Fig.2.

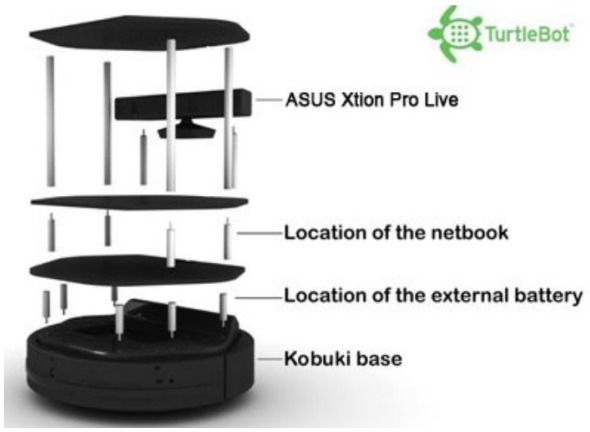


Fig. 2: Turtlebot 2.

In this case the robot is equipped with the following components:

Mobile Base: Kobuki, main component of the robot. It consists of a robot of differential kinematics which allows the locomotion, contains the sensors for the odometry, besides the power supply in the form of battery.

Sensor 3D : ASUS Xtion PRO Live, it has an infrared sensor, adaptive depth detection and color image camera.

Computer It works with *ROS* and Ubuntu 14.04 LTS (Trusty Tahr). *ROS* version that it used on the robot is *ROS Indigo Igloo* [20].

A. PRM's trajectories

PRM is a path planning algorithm that needs an existing map of the environment to work. The maps used were created by the robot using *Robotics System Toolbox* and Binary Occupancy Grid algorithm. [15]

The output of PRM will depend of two different parameters: the number of nodes and the distance of connection between nodes.

This not only will have a importance on the final path, but also, in the required time to obtained it. Consequently, for the same map and initial and final position exists a remarkable tendency to increase the computation time by increasing the number of nodes, owing to the increase of combinatorial possibilities, as it is showed on Fig 3.

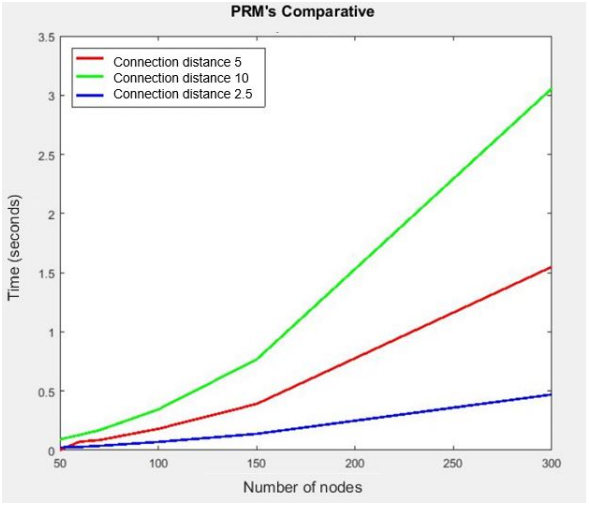


Fig. 3: Comparative of execution time for PRM.

But it is also verified that as a greater number of nodes is imposed the computational effort grows by varying the connection distance, this is highly related to the environment where the robot moves, as seen in Fig. 4 . The map represents a work area of about 25 m^2, so if large connection distances are imposed the chances of finding a solution decrease.

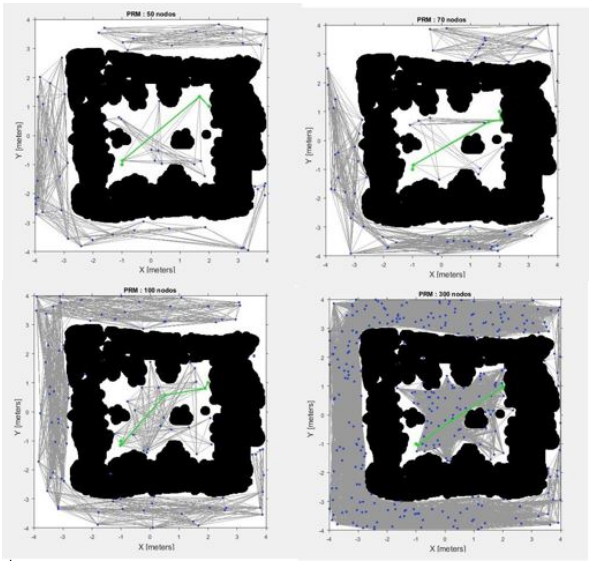


Fig. 4: Variation of nodes.

All this information is displayed into the following Table I, where it is observed that the execution times are relatively low. In it the time of calculation for different number of nodes and distance connection are shown.

Nodes	50	60	70	100	150	300
Dist 2.5	0.0234s	0.0274s	0.0366s	0.0699s	0.1385 s	0.4708 s
Dist 5	0.017s	0.0723s	0.0855s	0.1809s	0.3913 s	1.5501 s
Dist 10	0.0918s	0.1287s	0.1690s	0.3448s	0.7679 s	3.0579 s

TABLE I: Time of calculation of PRM.

Due to PRM is a random algorithm, it does not always ensure that the number of nodes increases as the final trajectory improves, a concept that can be easily observed in Fig. 4.

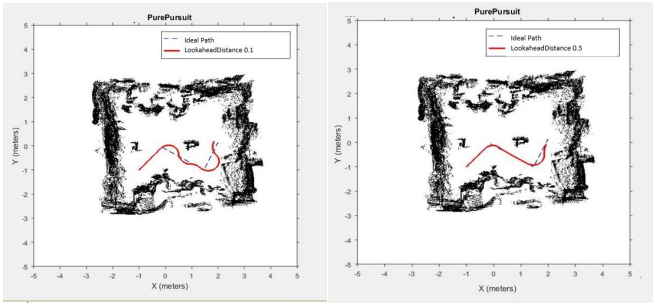
It is verified that there are trajectories obtained with 70 and 300 nodes that pass near obstacles, although these have been inflated to avoid collisions, in the real life trajectories near obstacles are not safe due to the different errors accumulated by the robot, such as inaccuracy in location.

Another factor that cannot be controlled by modifying the execution parameters is the optimality of the trajectory with respect to the length of the trajectory and abrupt twists that can not be realized by the robot's kinematic and dynamic constraints.

B. Trajectories of the real robot

The tests performed in this section were using PurePursuit algorithm [13] with a maximum angular velocity of 0.45 rad/s and 0.15 m/s of linear velocity, on a path generated by PRM with 50 nodes and a distance connection equal to 5 meters.

Variations in the real path were studied by modifying the value of LookaheadDistance variable, which corresponds to the distance to the point the robot is tracking during movement.



(a) LookaheadDist 0.1m (b) LookaheadDist 0.3m

(c) LookaheadDist 0.6m (d) LookaheadDist 1m

Fig. 5: Real trajectories with PurePursuit.

Experimentally it is observed (Fig. 5) that with low values of LookaheadDistance the robot follows a path somewhat distant to the desired one, due to the abrupt turns that it is forced to realize, whereas values from one meter the controller smooths the path to a point where the non-collision with objects is not guaranteed.

As it was mentioned for PRM, the value of LookaheadDistace will heavily depend on the environment and the length and curvature of the path. In this particular case of study the range of optimal values between 0.3-0.6 meters.

C. Particle Filter

Particle Filter or Method of Montecarlo is a technique to estimate the state of a dynamic system using Markov chains with discrete time. [2]

It consist on generating a set of particles uniformly distributed on the space that must converge in the true position of the robot.

Fixing a value of LookaeadDistance of 0.6 and same values velocities that the previous section was executed the PRM algorithm and PurePursuit adding a filter of particles.

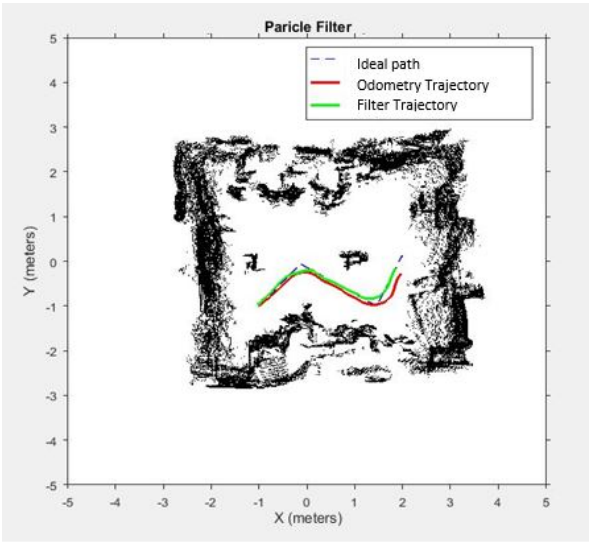


Fig. 6: Path with Particle Filter.

Fig. 6 shows that the robot follows more accurately the imposed trajectory. Two well differentiated stages can be observed, a first one where the path followed without and with the particle filter are almost identical, and a second where they differ. This is due to the convergence of the filter that corrects the position of the robot.

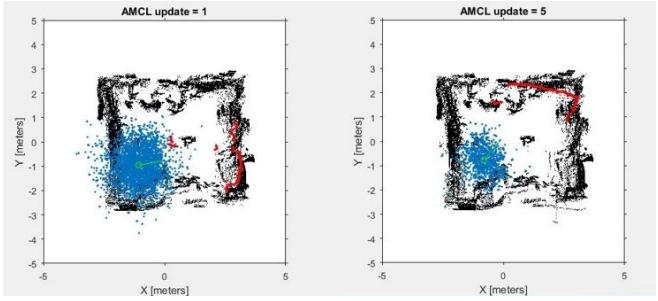


Fig. 7: Convergence of Particle Filter (1).

In order to understand these changes, the sequence of convergence of the filter must be observed. Fig. 7 shows how

the particle cloud starts with the maximum size of components and is gradually reduced.

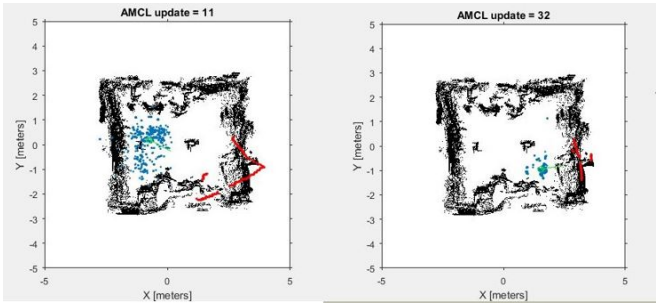


Fig. 8: Convergence of Particle Filter (2).

It is on Fig8 where the number of particle is considerably reduced. Nonetheless, the distribution of the sample is not uniform, it is on this moment when the errors of odometry are considerably important and the trajectories start to differ.

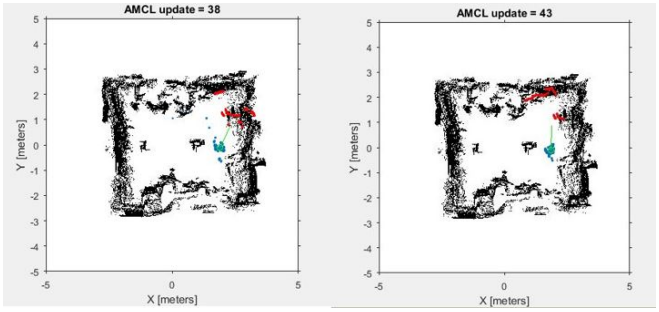
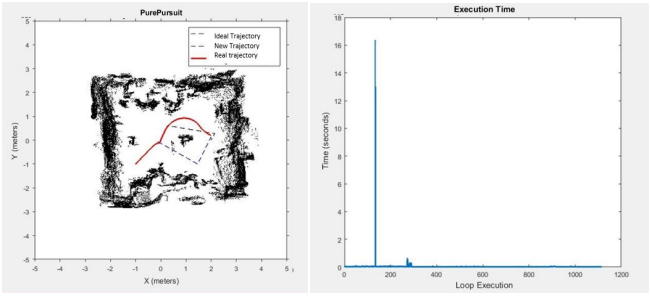


Fig. 9: Convergence of Particle Filter (3).

Finally, Fig 9 shows the complete convergence on the final step of the path.

D. New obstacle on the trajectory

Other of the cases studied is when the robot encounters an object not included in the initial map. As it is observed (Fig. 10a), the robot initiates the movement as in the previous cases until in the middle of the map it sees a new obstacle. On that moment, it stops, adds the new obstacle to the map and calculates a new free collisions path.



(a) New path (b) Execution time

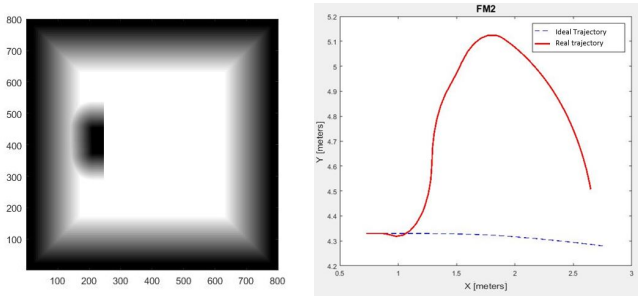
Fig. 10: New path due to an new obstacle.

Performing these three actions takes a considerable amount of time, if the normal execution of the code is equal to 0.048 seconds, since the new obstacle is recognized until the loop restarts approximately 16 seconds will pass, Fig. 10.

E. Fast Marching Square's results

The last set of experiments was performed with the algorithm of FM^2, developed in UC3M (Carlos III University of Madrid).

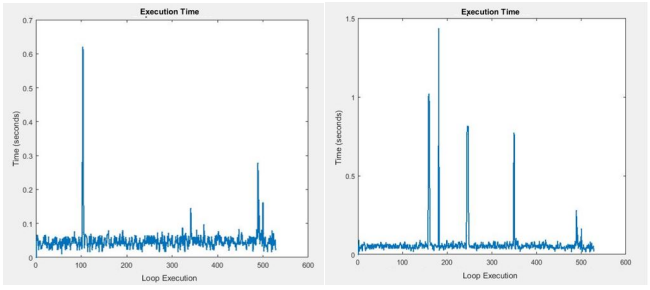
Using a blank map, the main function to be studied is the location of obstacles in the path. Unlike the PRM algorithm, the execution is performed continuously without interruptions when detecting objects and recalculating trajectories.



(a) Fm2 map (b) Trajectory

Fig. 11: FM2 path (1).

It begins with the recognition of a single object in a straight path. It is observed, Fig.11b, the algorithm generates an optimal path to avoid the collision of it. The average time of execution is 0.051, with a local maximum somewhat greater than 0.6 seconds at the time the object is recognized, Fig. 12a.

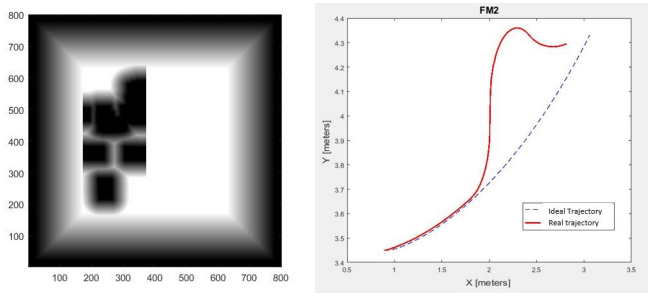


(a) Execution time one obstacle (b) Execution time several obstacle

Fig. 12: FM2 execution time.

Although the detection times are lower than those obtained with PRM, exceeding 0.6 seconds produces an intermittent movement due to the slowness in sending the speed messages to the robot.

At the moment that the algorithm, Fig. 13, must detect more than two objects is when the movement start to not be fluid , because it requires a greater computational effort, Fig 12b.



(a) Case I (b) Case II

Fig. 13: Simulation results for the network.

V. Conclusion

The functionalities of *Robotics System toolbox* have been studied, verifying that a stable and complete integration between Matlab and ROS is achieved, where the connection allows to send and receive information simplifying the process to develop and test robotics algorithm. The unique environment to generate and visualize data is what makes this toolbox an interesting element.

Several experiments were carried out in the laboratory in which the different planning, localization, mapping and navigation algorithms that are characteristic of an autonomous robotic system were approved, executing these tasks successfully on a real robot. Also this toolbox has allowed to test algorithms that were merely used in simulation.

The results of experiments using PRM and Fast Marching Square algorithm are exposed, reaching to the conclusion that a satisfactory performance of the algorithms is achieve. As it was possible to prove, the trajectories perform by the robot were adjusted to the theoretical ones and the error to reach the goal position was minimum.

The main problem resided is execution time, it has an important influence in several aspects. The fluidity of movement will depend on the frequency which the speed messages are sent to the robot. Thus, it has been proven that this lack of fluency can occur for execution delays due to algorithms that require a great computational effort and can delay the sending of information.

The lack of reaction to external events such as the perception of an object is also influenced by the execution time. The time it takes for the robotic system to recognize objects with sensors plus the time the algorithm takes to process that information are key concepts in automatic navigation cases. Thus, in the cases the algorithms have a longer execution time it has been due to the fact that the minimum distance with which the robot must perceive objects increase caused a major reaction time.

Through this work it has been verified that *Robotics System Toolbox* is a viable option for the implementation of autonomous programs for mobile robots. Which is much closer to the vast majority of students and engineers who are not knowledgeable in Linux or ROS, but who also provides interesting tools for research.

References

[1] Christopher Crick, Graylin Jay, Sarah Osentosiki, Benjamin Pitzer y Odest Chadwicke Jenkins. *Rosbridge: ROS for Non-ROS User*. 2011

[2] Frank Dellaerty, Dieter Foxy, Wolfram Burgardz y Sebastian Thruny *Monte Carlo Localization for Mobile Robots* ELsevier , 2000.

[3] H. Utz, S. Sablatnog, S. Enderle, G. Kraetzschmar *Miro - middleware for mobile robot applications* IEEE Transactions on Robotics and Automation (Volume: 18, Issue: 4, Aug 2002).

[4] Jose Pardeiro Blanco. Pre-Doctoral thesis: *Algoritmos de planificacin de trayectoria basado en Fast Marching Square*. Systems Engineering and Automatic of Carlos III University, 2014.

[5] J.V. Gomez, S. Garrido, L. Moreno. *The Path to Efficiency: Fast Marching Method for Safer, More Efficient Mobile Robot Trajectories.* IEEE Robotics and Automation Magazine. No. 4, vol. 20, pages: 111 - 120, 2013.

[6] K. Yang, M. Li, Y. Liu, and C. Jiang, *Multi-points fast marching: A novel method for road extraction*, in The 18th International Conferenceon Geoinformatics: GIScience in Change, Geoinformatics 2010, Peking University, Beijing, China, June, 18-20, 2010, 2010, pp. 15.

[7] Li Gang, Jingfang Wang. *PRM path planning optimization algorithm research* 2016.

[8] Lydia E. Kavraki, Mihail N. Kolountzakis, and Jean-Claude Latombe *Analysis of Probabilistic Roadmap for Path Palnning* IEEE Transactions on robotics and automation, Vol.14, No 1, February 1998.

[9] Matlab: *Robotics System Toolbox*, `http://mathworks.com/help/robotics/index.html`

[10] Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler y Andrew Ng. *ROS: an open-source Robot Operating System*. ICRA Workshop on Open Source Software, 2009.

[11] Peter Corke. *Integrating ROS and MATLAB*. IEEE Robotics & Automation Magazine. June 2015.

[12] R.Patrick Goebel *ROS By Examble. A do-it-yourself guide to the Robot Operation System*. Volume 1, 2015.

[13] R. Craig Conlter.*Implementation of the Pure Pursuit Path Tracking Algorithm*, 1992.

[14] Sebastian Thrun*Particle Filters in Robotics*, 2002.

[15] Sebastian Thrun. *Robotic Mapping : A Survey*. Pittsburgh, February 2002.

[16] S. Masoud Sadjadi, Philip K. McKinley, *Transparent autonomization in CORBA, Computer Networks*, Volume 53, Issue 10, 14 July 2009, Pages 1570-1586, ISSN 1389-1286.

[17] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*, 3rd ed. SpringerVerlag, 2008.

[18] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006. [Online]. Available: http://planning.cs.uiuc.edu/.

[19] S. Osher and J. A. Sethian, *Fronts propagating with curvature dependent speed: Algorithms based on hamilton-jacobi formulations*,Journal of Computational Physics, vol. 79, no. 1, pp. 1249, 1988.

[20] Turtlebot: Turtlebot2, `http://www.turtlebot.com/`.

[21] Vicent Girbs, Leopoldo Armesto, Josep Tornero, and J. Ernesto Solanes. *Smooth Kinematic Controller vs. Pure-Pursuit for Non-holonomic Vehicles*.

[22] Willow Garage `https://www.willowgarage.com/`.