SLAM IMPLEMENTATION

Roberts Veselovs

Abstract

Goal of this paper is to create and implement a SLAM algorithm onto a Pioneer 3Dx robot platform using ROS and Python programming. The main assumptions are that internal odometry will render manually calculating positions unnecessary and laser scanner shall be used to scan robot's surroundings. The result is an occupancy-grid map that shows robot's locations and obstacles in the near vicinity.

02/02/2017

Class: Intermediate Project

Instructor: PhD Witold Paluszyński

Department of Electronics

Wrocław University of Technology

Introduction

Goal

The ultimate goal of this project was to fully implement SLAM into a provided robot platform Pioneer3 [1]. SLAM is an acronym for Simultaneous Localisation And Mapping [2]. At its core it is the problem of constructing and updating a map of an unknown environment while simultaneously knowing the position of the robot in it. In the end the robot should be able to scan its surroundings, mark them on a map, and know the position of itself in said map.

Figure 1 - Pioneer 3DX robot used

Assumptions

During this project we assume that Pioneer3 has all necessary equipment to facilitate the implementations of SLAM: sensors that allow vision, internal odometry to let us know its position, and wireless communication capabilities with PCs. Basically no additional work is needed from us regarding the robot.

Constraints

Number one constraint is going to be data transmission and processing speeds. There will be a lot of data coming in from the robot. It might take some time to transfer all of it and then process, so making the real-time in the sense that it will instantly show updates will be impossible.

Another constraint was that the robot was available only during certain hours in the lab which prevented meaningful work from home.

Project implementation

For this project a lab computer with Linux OS was used. To control and communication with robot, ROS environment was made use of. Python was chosen as programming language for this task.

Preparation

For easier communication with robot, RosAriaDriver package was installed. This library has built-in functions that allow us to connect to the robot quickly and hassle free, specifically reading the obstacle sensors and acquiring odometry data [3]. For robot movement autonomous exploration was considered, but was abandoned as it required too much work. Rather 'teleop_twist_keybaord' was used to manually control the robot [4].

Chosen robot has two type of sensors that are capable of 'seeing' the world around it: sonar and laser. Sonar consists of 8 ultrasounds sensors placed in a 180° arc in front of the robot. Laser, however, is a light-based sensor that outputs 512 readings in the same 180° arc as sonar. As per specifications, the range of sonar is around 4m and laser is 5m. The choice was obvious in favour of laser scanner.

SLAM algorithm creation and implementation

In this project it was chosen not to implement SLAM that strictly follow its definition, but rather an algorithm that does the same task, but in a different way from a 'pure' SLAM algorithm. Below a simple flowchart will showcase the basic principle of implemented SLAM-inspired algorithm.

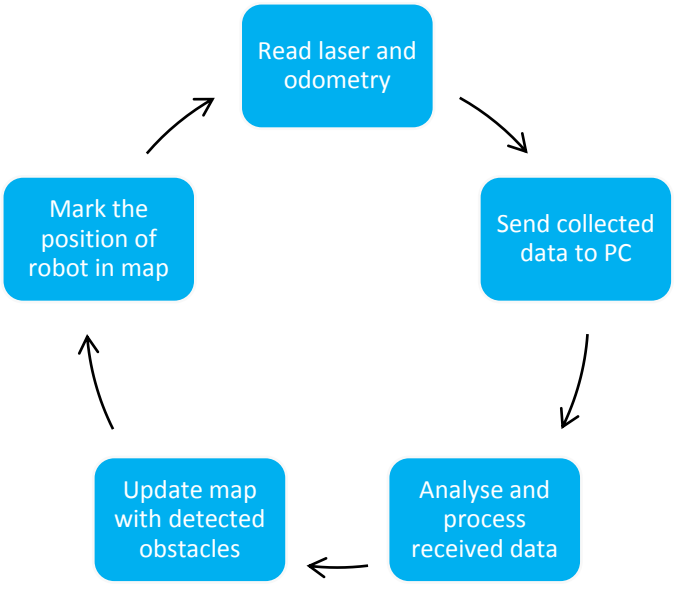

Figure 2 - Flowchart describing algorithm

Of course in the background a lot of actions are performed, but this chart does a good job of simplifying the whole algorithm.

Occupancy-grid map

An occupancy-grid map described as probabilistic robotics which addresses the issue of generating maps from noisy and uncertain sensor measurement all while the pose of robot is assumed to me know. This definition suits our case rather well. The pose is known indeed, and the sensor measurement are noisy and sometimes not precise due to various factors.

As we are using an occupancy-grid styled map, this required a lot of calculation. Each cell in grid has to be re-evaluated after each new set of sensor readings. That is the main point of program being rather slow. Multiple 'hits' on the each cell increases the certainty that and obstacle is there and each 'miss' decreases that certainty. With our algorithm it is possible to increase or decrease the number of cells and their size (that represents real life distances) in order to make it more precise, or to cover more area – it depends what are the needs in each scenario.

Results

The algorithm was successfully implemented and the manual control of robot was also established with success. In the next pages two scenarios will be shown and described.

Scenario 1

In the first scenario three obstacles were in placed in front of the robot's front 180° arc.

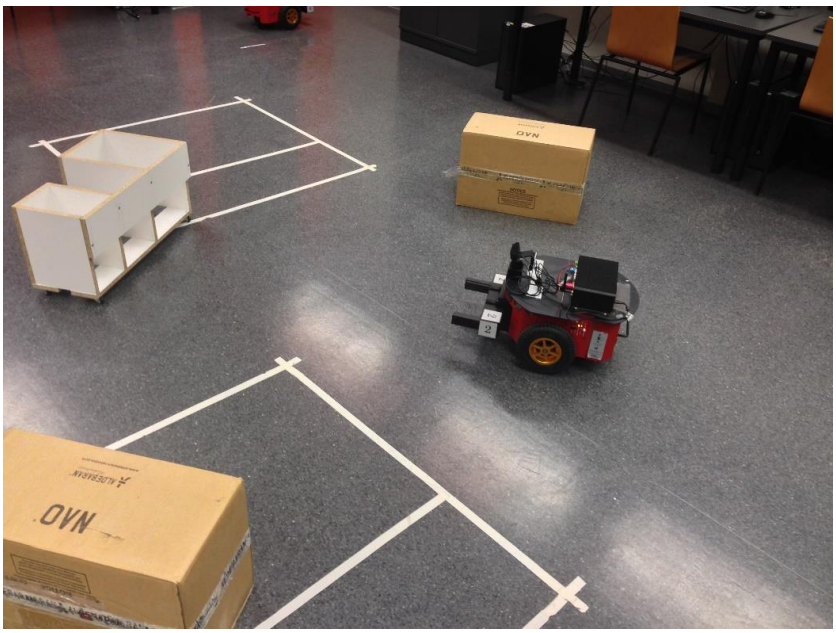

Figure 3 - Robot's and boxes' positions during scenario 1

Running the algorithm from this position of robot yielded the following map in Figure 4.

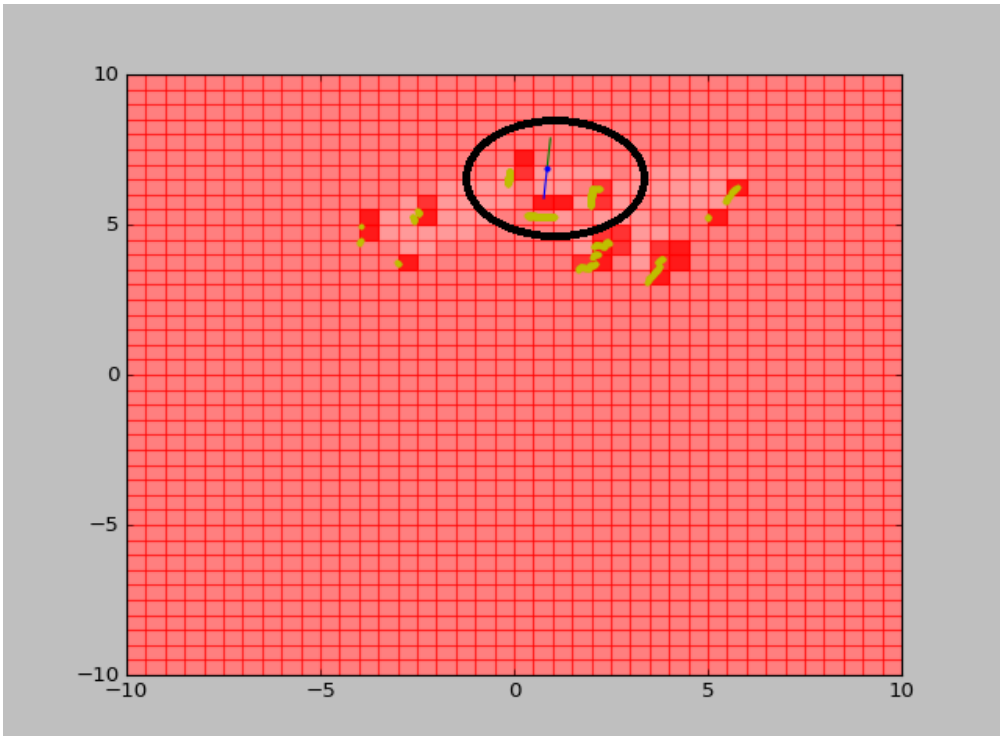

Figure 4 - Occupancy map for scenario 1

With the black circle the scene from the picture represented in map is marked. From the picture it could be estimated that the distance to boxes is around 1m or so. Based on parameter provided in the algorithm that specifies a cell being 0.5x0.5m, we can see that the right cells have been marked with higher certainty that there is an obstacle there. The green dots are laser's reading points where an obstacle was detected. For this scenario results were satisfactory.

Scenario 2

This is more uncommon situation, since the intention was to test whether the robot will see this gap as wide enough for it to pass or no. The setup can be seen in Figure 5.

Figure 5 - Setup for Scenario 2

From the results it is impossible to say what the algorithm decided about this gap between two boxes as the orientation was almost 45° from either axis. Combined with how the map is displayed it would be highly unlikely that the gap would be deemed passable, as the smallest division of cell was 0.5m which means that there has to be at least that much free-space for our algorithm to mark it as unoccupied.

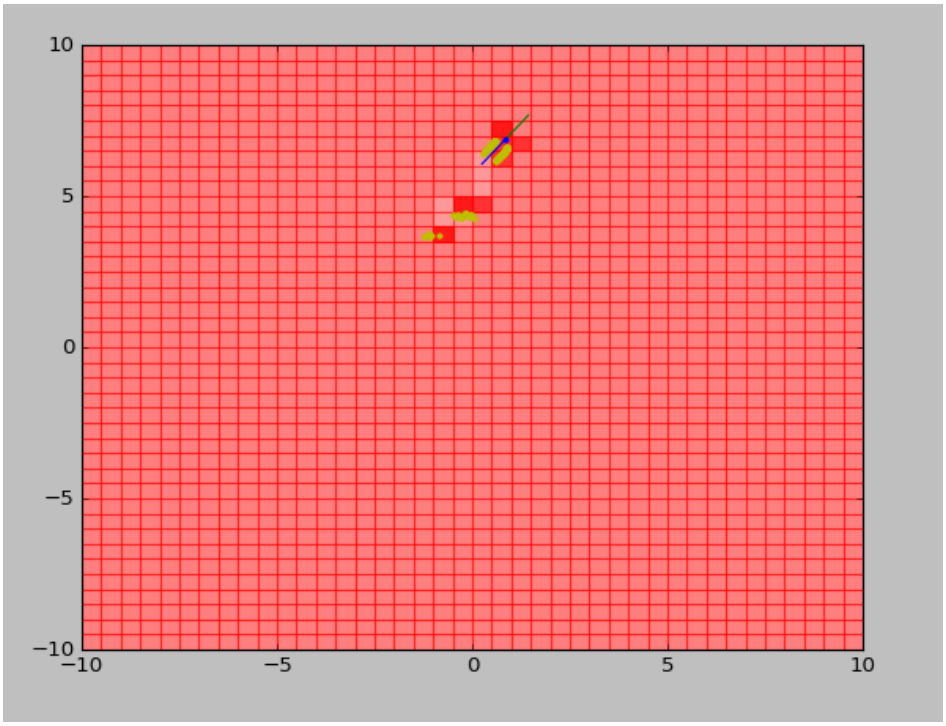

Figure 6 - Occupancy map for Scenario 2

Discussion

Overall the implemented SLAM-styled algorithm was successful in mapping the environment and localising robot in it. Due to chosen parameters in the algorithm, the map was not very accurate and precise. With a smaller cell size, the algorithm would have performed more accurate, but that would make it even slower, as there would be more cells, thus more calculations each program cycle. Regarding the speed of algorithm many improvements could potentially be made, for example, a better PC with higher processing speeds, faster wireless communication between the robot and PC, sending laser scanner reading not as often – instead of every time they are available, but perhaps every second.

Conclusions

To sum up, a SLAM-styled algorithm was proposed and implemented. Pioneer 3DX was used as platform for gathering information about surroundings. An occupancy-grid map was created, filled,

and showed. While the algorithm did its jobs, it was long way until it could be considered a proper SLAM algorithm. The main issue was the slow updating of map, which most likely was caused by sensors reading overload. Another issue was chosen parameters – the cell size was too big and that made the map less accurate than it could have been. Fine-tuning of cell size and grid size parameters this advised. The built-in (odometry) and template (laser scanner and control) functions helped greatly with realising this project. Overall, this was good project as a stepping stone for mobile robotics as it covers two most often encountered issues in the field.

References

[1] "Pioneer 3DX Manual", 2017. [Online]. Available: http://diablo.ict.pwr.wroc.pl/~jjakubia/MobileRobotics/doc/Pioneer3DX-P3DX-RevA.pdf. [Accessed: 16- Jan- 2017].

[2] S. Riisgaard and M. Blas, SLAM for Dummies, 1st ed. [Online]. Available: https://ocw.mit.edu/courses/aeronautics-and-astronautics/16-412j-cognitive-robotics-spring-2005/projects/1aslam_blas_repo.pdf

[3] *ROSARIA - ROS Wiki*, 2017. [Online]. Available: http://wiki.ros.org/ROSARIA. [Accessed: 01- Feb- 2017].

[4] "teleop_twist_keyboard - ROS Wiki", *Wiki.ros.org*, 2017. [Online]. Available: http://wiki.ros.org/teleop_twist_keyboard. [Accessed: 01- Feb- 2017].