IMPLEMENTATION OF ROS NAVIGATION STACK ON LEGO MINDSTORMS NXT 2.0 ROBOT

Mykhailo Riabtsev
Department of Automobile Transport,
Sevastopol State University,
Sevastopol, Russia
riabtsev.m@gmail.com

Rafael Priego, Marga Marcos
Department of Automatic Control and System Engineering
ETSI Bilbao, University of the Basque Country UPV/EHU
Bilbao, Spain
rafael.priego@ehu.es marga.marcos@ehu.es

Abstract

The robot navigation problem for assembly system was touched in this paper. Using Robotic Operating System navigation stack on a real robot requires expensive Light Detection And Ranging (LIDAR) sensor for localization. This feature makes using navigation stack impossible on robots with limited capabilities like Lego Mindstorms NXT 2.0. This paper presents the method of using robot odometry data for robot localization. The method is based on running real robot with stage_ros simulation at the same time. Simulated robot follows the real one and provides the position to the navigation stack of a real robot. The node for creating connection between real robot and the simulator was created and described. Its design is based on discrete PID controller principles. The strait line experiment proves the efficiency of the proposed concept, but additional work on rotational component of odometry should be carried out.

Key words: Robotic Operating System, PID controller, localization, odometry.

1 INTRODUCTION

It is well known that robots are replacing people in those branches of industry, where preciseness, repeatability, or non-stop work is necessary. Also robots are more suitable for working under dangerous factors, like radiation, chemical polution, heat or outerspace applications. One of the most widespread robot applications is construction and assembly operations [1]. But typicaly robots that are used on the plants and factories are stationary placed. That feature makes impossible building objects that are significantly bigger than the size of an assembly robot, like planes, ships and buildings. For building big object, robot needs to be mobile and be able to move upon the structure, which is being built.

Several research have been devoted to the development of multi mobile robot assembly systems. Terada and Murata in 2004 [2] developed a system consisting of passive modular blocks and assembly robot. Block sizes are standardized, which simplifies their representation in the form of a grid. The modules may have different shapes, but structures which were used in the experiments had the shape of a cube. The robot arm has two links. One of the links has L-shaped connector, which has two surfaces and serves to capture the blocks. The manipulator has 4 degrees of freedom. The robot is able to move upon constructed object, like a caterpillar, by bending and straightening. In 2006, Werfel et. all have developed an assembly system for two-dimensional structures [3]. The system consists of mobile robots and passive bricks. The bricks have a unique marking, which can be changed by the robots. Also the bricks are able to make some calculations and data exchange. These features increase the availability of global structural data and increase the reliability and the speed of construction. The same principle of information storage inside the environment is used in colonies of social insects. In robot control swarm behavior is used, which is also borrowed from insects. Stein, Schoen and Rus in 2011 created an assembly system with heterogeneous group of mobile robots [4]. A distinctive feature of the system is a function of the mass, reflecting the priority of part location and preventing physically impossible states. The system has delivery and assembly types of robots. Special partitioning algorithm that takes into account the geometry of the parts was developed. This algorithm serves to distribute tasks among robots. The system work autonomously after the plan is given and the robot roles are distributed among the group. Next research was carried out by Petersen Nagpal and Werfel in 2011 [5], [6]. They created a system called TERMES, which was inspired by termites colony. Swarm behavior is used for robot control, as well as in the previous study [3]. Robots can move plastic blocks and build different structures from them, which also can be used as scaffolding and ladders. Robots use a method similar to masonry for the construction. After the end of construction, unnecessary staircases are removed. To build the objects the robot is guided by a set of primitives that allow breaking the structure to certain sequences of actions. Before starting the construction, plan the object must be loaded in all the participating robots.

System which is being discussed in this paper uses the same principle as TERMES. The robot, of proposed system has forklift design. It gives several advantages, compared to which was used in TERMES. For example in TERMES robots can place blocks only at the same level with the robot. The forklift is able to put a block on the top of the two blocks, which is 80 mm high. This feature makes the building process more optimal as it decreases the length of additional auxiliary staircases. Robots have hierarchical control. It provides ability to download the plan of a structure simultaneously to every robot. The important safety feature of emergency stop becomes available, as well as dynamical changes in the structure during the assembly process run. Heterogeneous blocks are used in the system. This feature will increase structure strength and make it closer to real applications. There are two types of blocks: square and rectangular. The size of the square block is 36x36x4 cm, it was defined by the size of a robot. The rectangular block is a half of the square block; its dimensions are 36x18x4 cm. The blocks have special openings for the fork from four sides. The openings have guides for compensation of the impreciseness of robot navigation.

To implement desired system features, Robotic Operating System (ROS) was chosen. ROS has modular structure, which consists of programs (nodes) and communication channels (topics). This structure is easy to reconfigure, so it is easy to implement hierarchical control for the system. ROS has many useful built in nodes, which are united in packages and stacks, but it is possible to write custom node in case if existing nodes do not satisfy the system requirements. ROS provides navigation stack, which makes possible robot movement through known or unknown environments. The navigation through construction territory is necessary for the system. In most of considered researches, navigation was implemented using beacons and infrared sensors. Lego Mindstorms NXT 2.0 was chosen for the implementation of multi robot assembly system, but it does not provide those kinds of sensors in basic kit. In ROS navigation stack it is also impossible to use beacons with basic means. Most of the robots, which work under ROS, use LIDAR sensors for estimating the position of a robot on a map. LIDAR provides 3-d image of the surrounding environment to the robot. The drawback of LIDAR is its high cost. Also it cannot be connected to a Lego robot directly. That is why other localization principle should be used.

Research on using other kinds of sensors for localization, like ultrasonic and infrared, has been carried out before [7]. But there were no attempts to use simple sensors for making localization with ROS as there was no suitable package for that. There are two main packages which can be used in ROS for localization – *amcl* and *fake_localization*. Amcl is a package, based on adaptive Monte Carlo localization, which is used for simulation as well as the real robot. Its disadvantage is that it needs LIDAR sensor for its work [8]. *Fake_localization* can work with LIDAR, as well as only with odometry data, however, it is suitable only for simulation, as it gets the precise position of a robot directly from the simulator [9], [10].

The goal of this research is to use *fake_localization* with an additional package which will make possible its usage on a real robot. The package, which will connect real robot and simulation is based on the well known PID algorithm.

The layout of the paper is as follows: section II presents the package based on PID algorithm; implementation of PID algorithm is placed in section III; a case study of a multi mobile robot assembly system and experimental design is described in section IV; section V presents experimental results and discussion. Finally, section VI is dedicated to the concluding remarks and future work.

2 PID CONTROL NODE FOR ROS NAVIGATION

As it was mentioned before, in general, robots under ROS use LIDARs for localization. In this paper the real Lego NXT robot is used with *stage_ros* simulator. There is a map of a real environment, downloaded to *stage_ros*. The idea of this work is to synchronize movement of simulated robot in *stage_ros* and the movement of the Lego NXT in the real world.

The simulator will run simultaneously with the real robot and the both robots will move at the same distances on the map, in theory, their positions will be exactly the same. In this case, *fake_localization* package will provide information about simulated robot position to navigation stack, which will consider it like the information of a real robot position. According to the position, navigation stack will send velocity commands to the real robot. The real robot will start moving, and providing odometry information, by which its position can be estimated. Simulated robot should also start moving in this moment. In theory, sending the same velocity commands to the real robot and to simulator will make them pass the same distance and provide the same odometry data. But in reality, due to different errors and inaccuracy, it is impossible. The odometry data of real robot and the odometry data of the simulated one will be always different. To make the odometry data equal, the package that uses PID algorithm is proposed. This package connects Lego packages with *stage_ros*, and receives odometry data from the both parts (see figure 1). It calculates error between two odometry data and sends velocity commands to *stage_ros* to eliminate the error. The simulated robot receives the command and accelerates, trying to reach the real robot position. The node stops sending velocity commands to simulator when the error between data is equal zero.

Odometry information consists of three coordinate position data and three coordinate orientation data. Terrestrial robots, like NXT, typically work in 2-D spaces. It means that corrective velocity commands should be calculated for X and Y coordinates. Also, the robot can rotate only around Z axis, so rotational component of velocity should be calculated only to Z axis.

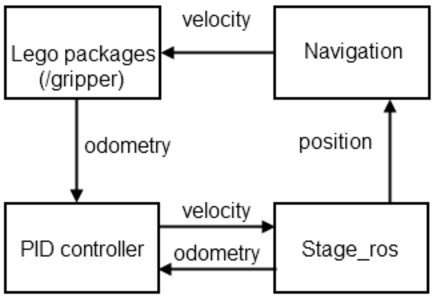

Figure 1: Connection diagram

3 IMPLEMENTATION OF PID ALGORITHM

3.1 DEFINITION OF PID

The equation, which underlies the package, is:

$$V(t) = P(t) + I(t) + D(t) \tag{1}$$

In equation 1, V(t) is velocity, which needs to be sent to *stage_ros* for position correcting; P(t) is proportional component; I(t) is integral component and D(t) is derivative component. In this case, the equation has a discrete form which is more convenient for programming. The components in this equation can be found like:

$$P(t) = K_p \cdot e(t) \tag{2}$$

$$I(t) = I(t-1) + K_i \cdot e(t) \tag{3}$$

$$D(t) = K_d \cdot (e(t) - e(t-1)) \tag{4}$$

$$e(t) = odom_{gripper} - odom_{stage} \tag{5}$$

In equations 2, 3, 4, and 5, K_p is proportional coefficient; K_i is integral coefficient, K_d is differential coefficient; e(t) is error between */gripper* odometry data ($odom_{gripper}$) and *stage_ros* odometry data ($odom_{stage}$). There are several methods of tuning coefficients for PID controllers, but in this case they were found in empirical way. $K_p = 1$, $K_i = 0.1$ and $K_d = 0.5$. Coefficients end errors have different units, depending on what velocity component is calculated. For linear components, e(t) will be in [m], and coefficients in 1/s. Rotational component's e(t) is represented like difference of the rotation angle cosines, and coefficient will be in [m/s].

3.2 ROS IMPLEMENTATION

Other ROS packages of the configuration, shown on figure 2, it can be seen that it consist of two parts.

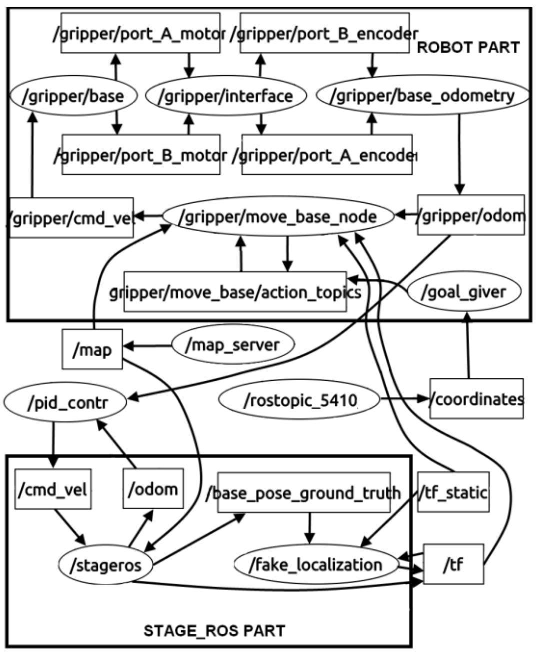

Figure 2: Configuration graph

The stage_ros part includes *stage_ros* simulator and *fake_localization* which serves for robot locating. In this part there are three topics, in which velocity, odometry and position of the simulated robot are published. The real robot part consists of several Lego robot operating packages (*gripper_base ,gripper_interface* and *gripper_odometry*), a *move_base_node* package, which is responsible for navigation and a *goal_giver* package which sends coordinates of a goal to *move_base_node*. Lego part also has topics for velocity and odometry of a robot. Also it has specific topics related to the motors and encoders and */gripper/move_base/action_topics*, to which *goal_giver* sends coordinates of a destination point, and waits for a reply about success or fail in reaching it. *Goal_giver* receives coordinates through */coordinates* topic, to which coordinates can be published manually. Except *pid_contr*, Lego robot part and *stage_ros* part have three more connections. *Tf* and *tf_static* provide coordinate transform from *fake_localization* to *move_base_*node, which is responsible for navigation. *Map_server* stores a map and provides it to *move_base_node* and *stage_ros*.

The working principle of configuration shown on figure 2 is the following: at first, three *.launch files should be started. First launch file contains Lego packages. Then the second launch file starts simulation part and *map_server*. After that navigation part of a real robot and *pid_contr* package are started. Next the *goal_giver* package is started. It sends coordinates to *move_base_node* and the real robot starts moving. NXT provides odometry to *pid_contr*. *Pid_contr* compares odometry from NXT which is moving with simulated robot which is standing, because it does not receive commands from *move_base_node*. *Pid_contr* gives simulated robot velocity commands, so it is starting to shorten the distance between it and the real one. The *move_base_node* package is monitoring the position of a simulated robot through *fake_localization*. When simulated robot reaches the goal, *move_base_node* sends real robot command to stop. At this moment, odometry data for both robots become equal. Quite important is that in *pid_controller* correction for Y component is calculated. The real Lego robot is non-holonomic, so it cannot execute the command of moving by Y axis. But as the commands are sent to simulator, it is to compensate the drift of a real robot with sending Y axis commands. To implement that, in the *.world file in "*define robot position*" section "*drive omni*" is written.

4 A CASE STUDY AND THE EXPERIMENTAL REALIZATION

Lego Mindstorms NXT 2.0 is one of the most popular robots for educational purposes. This robot has its own firmware, which allows to download comparatively simple programs inside the main brick. Though it's capabilities are limited it is widely used in various research. This robot was chosed for testing the concept of multi mobile robot assembly system. ROS has a special package for Lego Mindstorms. This package can add more abilities to NXT robot by using other ROS packages with it [11].

For multi mobile robot assembly system, robot should be able to navigate autonomously. There are obstacles on this way. The most significant is that for autonomous navigation, the NXT robot needs a computer, which will be small and light enough to be able to be transported by a robot. The solution of this problem can be Raspberry Pi, on which Linux and ROS can be installed [12]. This idea was left for future work and was not implemented in the experiment.

The robot, which was created for the system was used in the experiment. It has a forklift structure; its dimensions are 35x21x30 cm (figure 2).

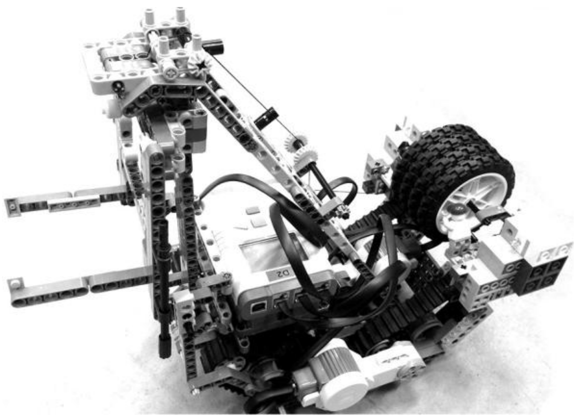

Figure 3: Lego Mindstorms NXT robot configuration

As it can be seen from figure 3, robot has tracks. The servomotors transfer torque to the tracks through the reduction gears, with ratio of ½. Ultrasonic range sensor is placed between the tracks. The tracks drive is designed for climbing the obstacles of 40 mm height. The robot should be able to work on the structure, which is being assembled, so it should be able to climb the height of the block. The robot is able to rotate on a block with the fork in the upper position (see figure 4).

Navigation part is made by PC due to the limited abilities of the main brick of a robot. NXT is connected with PC by wires. The PC, which controls the robot in experiment, runs under Ubuntu 14.04 and uses ROS Indigo. The NXT Lego packages were adapted to ROS Indigo, as originally they were designed for ROS Electric.

For the experiments, an empty map of 3x3 meters was created. The robot with the same shape was created in stage *.world file. All the packages were organized and launched as it was described in previous sections.

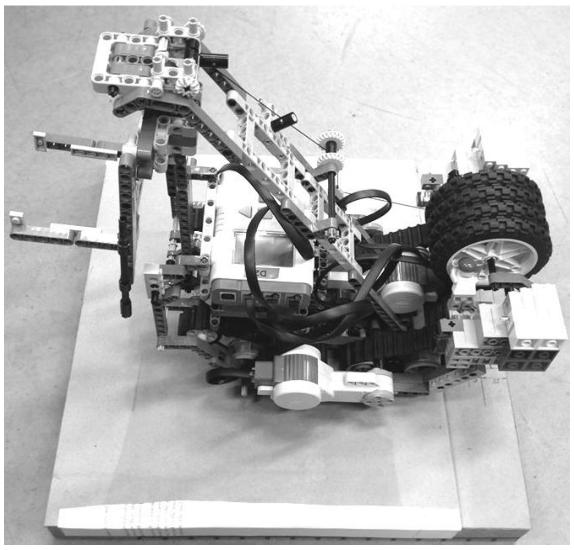

Figure 4: Robot on a block

To test the configuration, a straight line test, a rotation test and a combined test were carried out. The main idea of a straight line test was to estimate the error of the robot odometry with the real distance between start point and finish point. The tape measure was used for measuring the distance that real robot passed. The front rim axle was used as a reference point. The straight line test was carried out for several distances. It was checked on 0.5 meter, 1 meter and 2 meters distance. Each distance was passed 20 times.

The rotation test was necessary to estimate the error in rotational component of odometry data. In this test, robot rotates from its initial position to a certain angle. These angles were: 90, 180 and 270 degrees. The angle measurement was carried out by the compass application of mobile phone attached to the robot. This test was carried out 10 times for each angle.

Combined test consist of two linear movements and two rotations. After the test starts robot moves 30 cm forward, then rotates to 180 degrees. Next it makes the same actions, so at the end of the test, robot comes to its initial position. The difference between the position of X Y and angular component were estimated. This test was carried out 10 times.

5 EXPERIMENTAL RESULTS AND DISCUSSION

After final adjustments, there was no noticeable error on the 0.5 meter distance. The average error on 1 meter distance was 0.5 cm, and the average error on 2 meter distance was 1 cm.

For the short distance error calculations the measuring method provided some precision difficulties. On longer distances the error becomes noticeable, so the conclusion can be made, that error is accumulative. During the adjustments, it was found, that the parameter responsible for radius of the wheel in *diff_base_odom.cpp* was not correct. The wheel radius was decreased on 0.9 mm from its rated size. This can be caused by the non-uniform radius of the track, because of the grousers on the surface of a track. The other source of an error is the speed of data transfer, which is still quite slow for the motor encoders. Also the inertia of the motors, which cannot stop immediately, and can make the distance the robot walkthrough, longer.

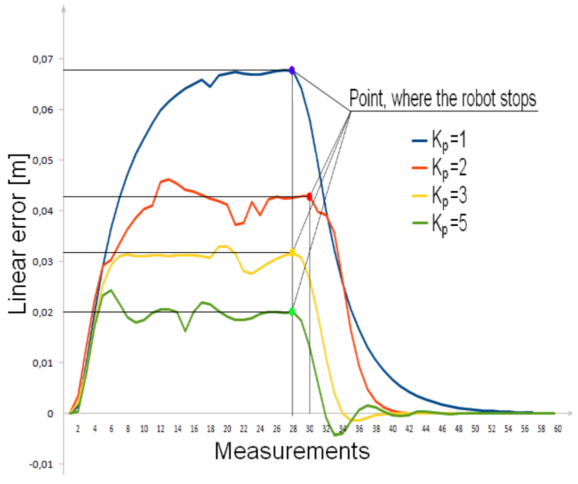

Figure 5: Odometry error graph

During tests the error in the odometry data was measured. The results are placed on a graph in figure 5. The graph was created during the strait line test. The robot was passing the 50 cm distance. As it can be seen from the graph, when the real robot starts moving, the error between odometry data increases rapidly. Then there is a zone of comparatively stable error until the point where real robot stops. After this part the error decreases very fast, as the real robot's odometry is not changing any more. The experiment showed that the value of an error can be changed by adjusting coefficients K_p, K_i and K_d. The biggest influence on the error value has K_p coefficient (see figure 5).

Rotation tests showed that the robot has small error in rotational component (1-5 degrees), but this error tends to accumulate. Especially it becomes visible after several oscillations, when the robot tries to get into desired position. The oscillations were caused by

inertia of a real robot. This effect can be decreased by adjusting the coefficients of PID controller. Although, the error tends to accumulate, the behaviour of the system showed that PID control is able to work with rotational component of odometry data.
In order to solve the accumulation of the angular error other type of sensor can be introduce like the magnetometer which will provide the absolute angle to a robot.

The combined test result showed the same problem as a rotation test. In 50% of cases robot X and Y error were small or insignificant, but the angular error after final rotation was more than 20 degrees. In 10% of cases real robot position was good, but the position of the robot in stage_ros showed the error in rotation component of odometry data. In other cases the fails were caused by the error accumulation after several oscillations during first rotation.

6 CONCLUSIONS AND FURTHER WORK

This paper has introduced the use of ROS *fake_localization* package for Lego Mindstorms NXT 2.0 robot navigation. The concept was tested with crawler drive robot which has differential steering.

The package that implements PID control principles appeared to be effective for localization the robot, when it moves strait forward. The main advantage of using only odometry data for localization is the removal of expensive LIDAR sensors.

Rotation tests showed small error after single rotation. Several rotations cause the accumulation of error, which makes the navigation more complex. The oscillation effect, caused by inertia of a real robot, increases the error accumulation. On-going work is been done in introducing a magnetometer sensor to eliminate the accumulation of rotational error. To decrease the oscillations, coefficients for PID rotational component should also be adjusted.

Tests showed that PID control concept is able to work for both: linear and rotational components of odometry, but the packages need further adjustments.

Acknowledgements

This work became possible thanks to grant from Erasmus Mundus EWENT program.

This work was financed in part by the University of the Basque Country (UPV/EHU) under project UFI 11/28, by the Regional Government of the Basque Country under Project IT719-13

References

[1] Guizzo, E. The Rise of the Machines. Spectrum, IEEE Volume:45, Issue: 12 p. 88, Dec. 2008

[2] Terada Y. Murata S. Automatic Assembly System for a Large-scale Modular Structure. Hardware design of module and assembler robot. Proceedings of IEEWRSJ International Conference on Intelligent Robots and Systems September 28 - October 2, 2004, Sendai, Japan

[3] Werfel J., Bar-Yam Y., D., Rus R., Nagpal Distributed Construction by Mobile Robots with Enhanced Building Blocks Proceedings of the IEEE International Conference on Robotics and Automation Orlando, Florida - May 2006

[4] Stein, David, T. R. Schoen, and D. Rus. “Constraint-aware Coordinated Construction of Generic Structures.” IEEE, 2011. 4803–4810.

[5] Petersen K., Nagpal R., and Werfel J. TERMES: An autonomous robotic system for three-dimensional collective construction. In Proc. Robotics: Science & Systems VII, 2011.

[6] Petersen K., Nagpal R., and Werfel J. Distributed Multi-Robot Algorithms for the TERMES 3D Collective Construction System Workshop on Reconfigurable Modular Robotics. IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS), 2011

[7] Beevers K. R., Huang W. H., SLAM With Sparse Sensing Proceedings of the 2006 IEEE International Conference on Robotics and Automation Orlando, Florida - May 2006

[8] NeilTraft "Amcl" Available at: http://wiki.ros.org/amcl

[9] Dio Eraclea "Navigation with real robot" Available at: http://answers.ros.org/question/210583/navigation-with-real-robot/?comment=210998#comment-210998

[10] Jihoonl "Fake_localization" Available at: http://wiki.ros.org/fake_localization

[11] Raul.perula "Robots/NXT" Available at: http://wiki.ros.org/Robots/NXT

[12] StevenShamlian "ROSberryPi/Installing ROS Indigo on Raspberry Pi" Available at: http://wiki.ros.org/ROSberryPi/Installing%20ROS%20Indigo%20on%20Raspberry%20Pi