БЪЛГАРСКА АКАДЕМИЯ НА НАУКИТЕ • BULGARIAN ACADEMY OF SCIENCES

ПРОБЛЕМИ НА ТЕХНИЧЕСКАТА КИБЕРНЕТИКА И РОБОТИКАТА, **66**
PROBLEMS OF ENGINEERING CYBERNETICS AND ROBOTICS, **66**

София • 2015 • Sofia

# Optimizing the Navigation for Mobile Robot for Inspection by Using Robot Operating System

Denis Chikurtev

*Institute of Information and Communication Technologies, 1113 Sofia*
*Emails: dchikurtev@gmail.com*

***Abstract****: in this paper is presented ROS Navigation Stack for Simultaneous Localization and Mapping. By improving the functionality and the properties of the ROS Navigation Stack is optimized the movement, precision and orientation of the mobile robot. For localization and mapping is used Kinect sensor. The algorithm for navigation creates a map of the environment, after that generate a path to the desired destination and navigate the robot. The robot can be controlled distantly over internet or it's Wi-Fi. Experiments are made in indoor environment with static and dynamic obstacles.*
***Keywords:*** *mobile robot, navigation algorithm, PID control, obstacle avoidance, Internet, ROS.*

## 1. Introduction

Mobile robots are now widely used in many industries due to the high level of performance and reliability. Mobile robots for inspection are mainly used to replace human in dangerous environment. Because of that these robots must have autonomous navigation to reach desired objects that will be inspected. For the safety of the humans, robot and the subjects in the environment of movement, the robot must have obstacle avoidance function. In this paper for navigation is

introduced ROS Navigation Stack for navigation. It provides functions like Publishing Sensor Stream, Publishing Odometry Information, Transform Configuration and Map Building.

### *1.1 Navigation*

Mobile robot navigation systems require both sufficiently reliable estimation of the current robot location and precise map of the navigation area. These systems are separated into two levels of control: global path planning and local motion control. Path planning considers a model or a map of the environment to determine the geometric path points for the mobile robots to track from a start position to the goal. Whereas local motion usually use sensory information to determine a motion that will avoid collision with unknown obstacles or obstacles whose position in the environment had changed.

A complete mobile robot navigation system should integrate the local and global navigation systems: the global system pre-plan a global path and incrementally search new paths when discrepancy with the map occurs; the local system uses onboard sensors to detect and avoid unpredictable obstacles. The mobile robot executes an algorithm which permits it to follow the optimal global path by tracking its geometric points from a start position to the goal. If an obstacle obstructs this path, the robot executes another algorithm (collision avoidance algorithm) allowing it to move around the perimeter until the nearest point of the obstacle to the geometric path point is found, or pre-plan another optimal global path to reach the goal.

The local navigation systems are capable of producing a new path in response to the environmental changes. These systems can be divided into directional and velocity space based approaches (Seder, M.; Macek, K.; Petrovic, I., 2005). The directional approaches such as Potential field method (Khatib, O., 1986), Virtual Force Field (Borenstein, J. & Koren, Y., 1990) which extends to Vector Field Histogram (Borenstein, J. & Koren, Y., 1991) and Nearness Diagram algorithm (Minguez, J. & Montano, L, 2000), generate a direction for the robot to head in. Velocity space approaches such as Curvature Velocity method (Simmons, R., 1996), Lane Curvature method (Ko, N.Y. & Simmons, R., 1998), and Dynamic Window method (Fox, D.; Burgard, W. & Thrun, S., 1997), perform a search of the commands controlling the robot such as translational and rotational velocities directly from the velocities space.

In this paper we use SLAM (Simultaneous Localization and Mapping) for mobile robots *(Riisgaard, S. & Rufus, M. B. 2007)*. The SLAM process consists of a number of steps. The goal of the process is to use the environment to update the position of the robot. Since the odometry of the robot (which gives the robots position) is often erroneous we cannot rely directly on the odometry. We can use laser scans of the environment to correct the position of the robot. This is accomplished by extracting features from the environment and reobserving when the robot moves around. An EKF (Extended Kalman Filter) is the heart of the SLAM process. It is responsible for updating where the robot thinks it is based on these features. The EKF keeps track of an estimate of the uncertainty in the robots

position and also the uncertainty in these landmarks it has seen in the environment. An outline of the SLAM process is given below.

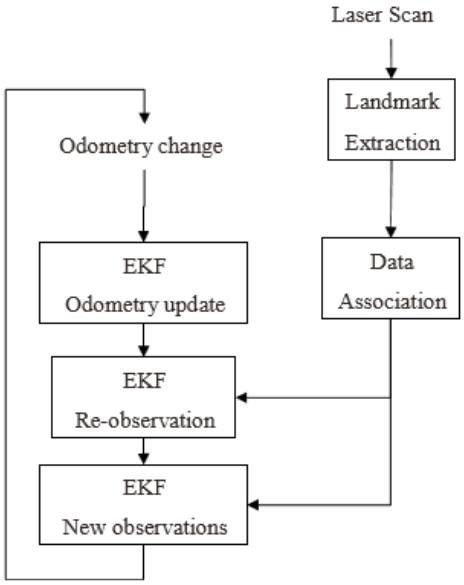


*Figure 1.* ***SLAM Process***

When the odometry changes because the robot moves the uncertainty pertaining to the robots new position is updated in the EKF using Odometry update. Landmarks are then extracted from the environment from the robots new position. The robot then attempts to associate these landmarks to observations of landmarks it

## *1.2 Sensors*

### *1.2.1 Kinect sensor*

Inside the sensor case, a Kinect for Windows sensor contains:

- An RGB camera that stores three channel data in a 1280x960 resolution. This makes capturing a color image possible.
- An infrared (IR) emitter and an IR depth sensor. The emitter emits infrared light beams and the depth sensor reads the IR beams reflected back to the sensor. The reflected beams are converted into depth information measuring the distance between an object and the sensor. This makes capturing a depth image possible.
- A multi-array microphone, which contains four microphones for capturing sound. Because there are four microphones, it is possible to record audio as well as find the location of the sound source and the direction of the audio wave.

- A 3-axis accelerometer configured for a 2G range, where G is the acceleration due to gravity. It is possible to use the accelerometer to determine the current orientation of the Kinect.

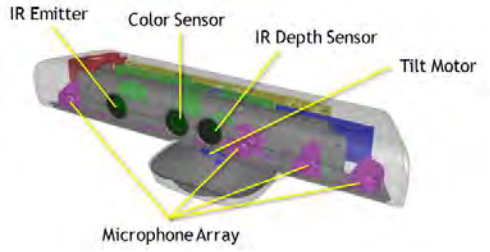


*Figure 2.* ***Kinect sensor***

Each frame, the depth sensor captures a grayscale image of everything visible in the field of view of the depth sensor. A frame is made up of pixels, whose size is once again specified by NUI_IMAGE_RESOLUTION Enumeration. Each pixel contains the Cartesian distance, in millimetres, from the camera plane to the nearest object at that particular (x, y) coordinate, as shown in Figure 1. The (x, y) coordinates of a depth frame do not represent physical units in the room; instead, they represent the location of a pixel in the depth frame.

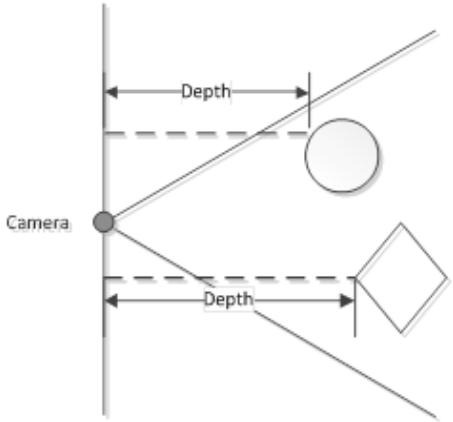


*Figure 3.* ***Depth stream values***

When the depth stream has been opened with the NUI_IMAGE_STREAM_FLAG_DISTINCT_OVERFLOW_ VALUES flag, there are three values that indicate the depth could not be reliably measured at a location. The "too near" value means an object was detected, but it is too near to the sensor to provide a reliable distance measurement. The "too far" value means an object was detected, but too far to reliably measure. The "unknown" value means no object was detected.

The depth sensor has two depth ranges: the default range and the near range. The image in figure 4 illustrates the sensor depth ranges in meters.

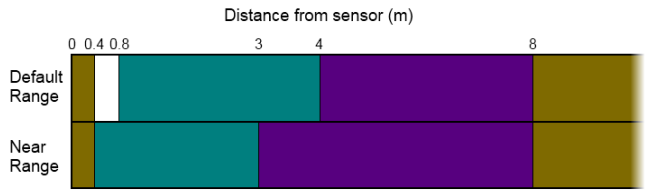


*Figure 4.* ***Depth range of Kinect***

### *1.2.2 Encoders*

An absolute rotary encoder determines its position using a static reference point. The method is slightly different depending on whether the absolute rotary encoder is optical or magnetic, but the principle is the same either way. There are two discs, both with concentric rings with offset markers. One disc is fixed to the central shaft; the other moves freely. As the disc turns, the markers along the track of absolute encoders change position on the fixed disc. Each configuration along the disc of an absolute rotary encoder represents a unique binary code. Looking at the binary code within the absolute rotary encoder determines the absolute position of the object. For optical absolute encoders, the marker is an opening which lets through light. For magnetic absolute encoders, the markers are a magnetic sensor array that passes over a magnet and detects the position of the magnetic poles.

By having an integrated reference, an absolute rotary encoder is intrinsically able to deliver higher quality feedback:

- Higher overall resolution and orientation
- Better start up performance because of low homing (or initial position) time
- Accurate motion detection along multiple axes
- Multiple output protocols for better electronics integration
- Better recovery from system or power failures

Another key feature of absolute encoders is the different output options. Encoders can't just collect feedback data; they have to send it somewhere in a language that the larger system can understand. Absolute encoders use binary coding, which is translatable into many different protocols. If you have multiple components using the same communications bus (such as multiple electronics systems on a fire truck), then it is critical that the absolute rotary encoder can communicate with the bus.

## *1.3 ROS Navigation Stack*

We are programming the robot by using Robot Operating System (ROS). ROS is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behaviour across a wide variety of robotic platforms.

One of the basic goals of ROS is to enable roboticists to design software as a collection of small, mostly independent programs called nodes that all run at the same time. For this to work, those nodes must be able to communicate with one

another. The part of ROS that facilitates this communication is called the ROS master. A running instance of a ROS program is called a node. The primary mechanism that ROS nodes use to communicate is to send messages. Messages in ROS are organized into named topics. The idea is that a node that wants to share information will publish messages on the appropriate topic or topics; a node that wants to receive information will subscribe to the topic or topics that it's interested in. The ROS master takes care of ensuring that publishers and subscribers can find each other; the messages themselves are sent directly from publisher to subscriber (O'Kane, 2013).

The Navigation Stack is fairly simple on a conceptual level. It takes in information from odometry and sensor streams and outputs velocity commands to send to a mobile base. Use of the Navigation Stack on an arbitrary robot, however, is a bit more complicated. As a pre-requisite for navigation stack use, the robot must be running ROS, have a "***tf***" transform tree in place, and publish sensor data using the correct ROS "***Message types***". Also, the Navigation Stack needs to be configured for the shape and dynamics of a robot to perform at a high level. To help with this process, this manual is meant to serve as a guide to typical Navigation Stack set-up and configuration.

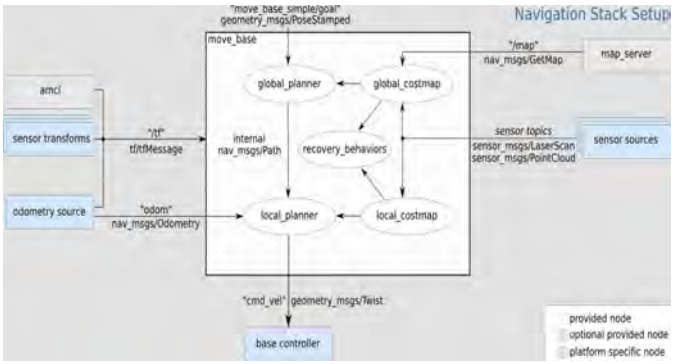


*Figure 5.* ***Navigation Stack Setup***

The navigation stack assumes that the robot is configured in a particular manner in order to run. The diagram above shows an overview of this configuration. The white components are required components that are already implemented, the gray components are optional components that are already implemented, and the blue components must be created for each robot platform. The pre-requisites of the navigation stack, along with instructions on how to fulfil each requirement, are provided in the sections below.

## METHODS

### Navigation stack

The navigation stack requires that the robot be publishing information about the relationships between coordinate frames using "**tf**". The "tf" library enables sensor data, or any data with a Stamp, to be transmitted in its original frame across the

network as well as to be stored in its original frame. When an algorithm wants to use data in the coordinate frame most relevant to the computation, it can query the "tf" library for the transform from the coordinate frame of the Stamped data to the desired coordinate frame. Using the resultant transform to transform the data at the time of need prevents unnecessary intermediate transforms, saving both computational time and degradation of data due to repeated processing with potential rounding issues.

For convenience, to get the latest data available, a request at time zero will return the latest common time across the queried values. If such a time does not exist the Listener will raise an exception, in the same way as if an unavailable time was queried outside of the cached history. The "tf" library enables sensor data, or any data with a Stamp, to be transmitted in its original frame across the network as well as to be stored in its original frame. When an algorithm wants to use data in the coordinate frame most relevant to the computation, it can query the tf library for the transform from the coordinate frame of the Stamped data to the desired coordinate frame. Using the resultant transform to transform the data at the time of need prevents unnecessary intermediate transforms, saving both computational time and degradation of data due to repeated processing with potential rounding issues.

For convenience, to get the latest data available, a request at time zero will return the latest common time across the queried values. If such a time does not exist the Listener will raise an exception, in the same way as if an unavailable time was queried outside of the cached history.

Sensor information

The navigation stack uses information from sensors to avoid obstacles in the world, it assumes that these sensors are publishing either sensor_msgs/LaserScan or sensor_msgs/PointCloud messages over ROS. Publishing data correctly from sensors over ROS is important for the Navigation Stack to operate safely. If the Navigation Stack receives no information from a robot's sensors, then it will be driving blind and, most likely, hit things.

For robots with laser scanners, ROS provides a special Message type in the **sensor_msgs** package called `LaserScan` to hold information about a given scan. `LaserScan` Messages make it easy for code to work with virtually any laser as long as the data coming back from the scanner can be formatted to fit into the message. This package is used to get information from the Kinect sensor.

Odometry information

Odometry is one of the most important parts to reach good accuracy and precision. Odometry parameters of our robot are:

- Wheel radius – 76,2 mm
- Distance between wheels – 390 mm
- Encoders resolution – 144 ticks / turn

The odometry data is used to estimate the position of the robot relatively to the starting location. Our robot has encoders on the axis of the powered wheels. We use those sensors to compute the odometry. The algorithm is described

a) We calculate the distance travelled by each wheel in meters, given the current readings of the encoders:

left_distance := (current_left_encoder_ticks – last_left_encoder_ticks) / LEFT_TICKS_PER_METER

right_distance:= (current_right_encoder_ticks – last_right_encoder_ticks) / RIGHT_TICKS_PER_METER

last_left_encoder_ticks := current_left_encoder_ticks

last_right_encoder_ticks := current_right_encoder_ticks

b) Next, we estimate the total distance between the current and previous positions in meters:

distance := (left_distance + right_distance) / 2.0

c) We calculate the heading of the robot:

theta := theta + (left_distance – right_distance) / DISTANCE_BETWEEN_WHEELS

d) Finally, we can estimate the current position (x, y) of our robot:

x := x + distance * cos(theta)

y := y + distance * sin(theta)

All these calculations are needed to the “tf” for path generating and navigation.

Base controller

The navigation stack assumes that it can send velocity commands using a geometry_msgs/Twist message assumed to be in the base coordinate frame of the robot on the "cmd_vel" topic. This means there must be a node subscribing to the "cmd_vel" topic that is capable of taking (vx, vy, vtheta) <==> (cmd_vel.linear.x, cmd_vel.linear.y, cmd_vel.angular.z) velocities and converting them into motor commands to send to a mobile base.

The algorithm for navigation of our robot is shown on fig. 6.

This is simple algorithm that shows how navigation operates. The first thing that have to be done is to open existed map or build one. After that the robot finds its position and orientation. Without knowing his position navigation can’t continue. Base position can be given by robot operator for better performance. When the robot knows his position we can set a target or destination. Usually we choose that by the map and the navigation knows where it is. Now is a time for path generating to the given point. The build-in algorithm for path generation always finds the shortest way.

When the robot starts moving system check its position and orientation in every moment. This is necessary because the floor is never smooth and the robot start to turn or swing and then it must to back in the right way. This loop of the algorithm continues till the target is reached. I cases when the flour is smooth this is not

necessary and the odometry is enough for the good navigation to the desired destination. When the robot goes to the wanted point the navigation knows that the mission is complete and wait for the next point.

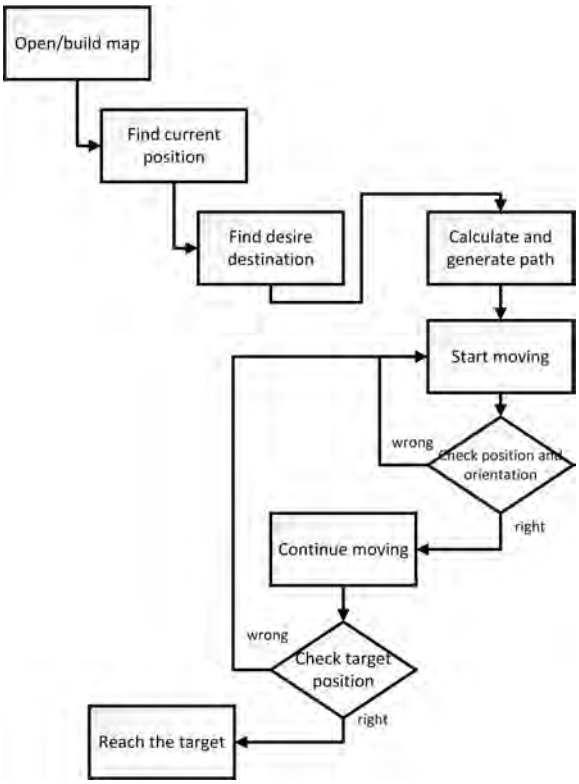


*Figure 6.* ***Algorithm for autonomous navigation***

Ones the robot finds its position the navigation known in every moment where it is and is not necessary to search for that position. This algorithm is good for obstacle avoidance, because it checks for position and destination incessantly and Kinect sensor can see if there is obstacle.

Very important part of the good and correct movement is odometry data. If parameters are not calculated correctly robot will never reach the final. By using odometry date the navigation knows how to control the motors of the robot wheels. The navigation calculates what to be the power on each wheel, how much degrees to rotate it or how many steps to go of the encoder. All this date makes the navigation possible and currect.

Mapping

This function is used to create and generate map. It is very useful for navigation. All maps can be used in the Navigation stack for better orientation and precision. In figure 7 is shown one of the maps that was built by our robot.

Map is generated by scanning the area with the robot and using the "slam_gmapping" node for building a map. When we have generated map we can use it any time when the robot is in the same area. Without a map the robot can't find his current position and the navigation won't work.

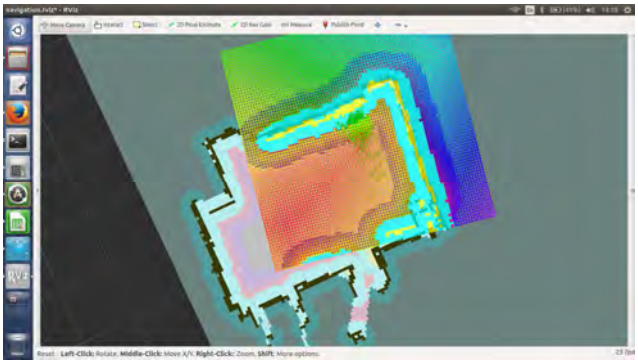

*Figure 7.* ***Generated map***

RESULTS

The navigation was tested in indoor. The experiment is to show what is the accuracy and precision of the navigation and proposed algorithm. The robot has made map on a floor of a building and then navigate in it. We set tolerance to at 100 millimeters from the center of the robot platform and desired point.

The robot finds its position and orientation autonomously and after that it is waiting for to be given destination pint. When we give destination point, the robot starts moving to it. Every time the robot reached the given point with the precision that was set. The navigation recognize if there are obstacles and avoid them. On figure 8 is shown the moment when the robot is starting to move. It is positioned in the center of the square and is illustrated as a point. Green vectors are showing robot orientation, red line is the generated path and the end of the red line I the final point that have to be reached. This experiment was in a single room and the robot perform all the tasks correctly.

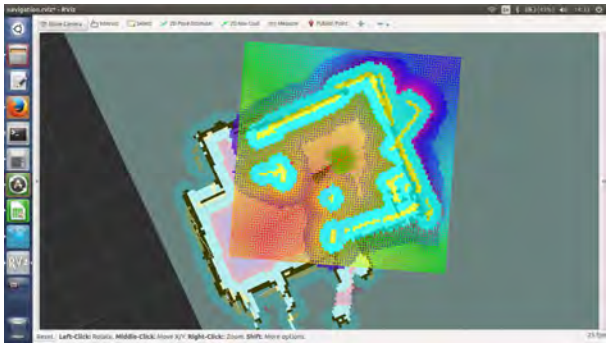

*Figure 8.* ***Robot base position and path generation***

The next experiment was in the whole floor and the robot had to move from one room to another. This is more complicated task but the robot perform it correctly again. In figure 9 is shown how robot goes from the first room to the corner of the lobby. It goes threw two doors and didn't clash. The navigation pilot the robot smooth and correct.

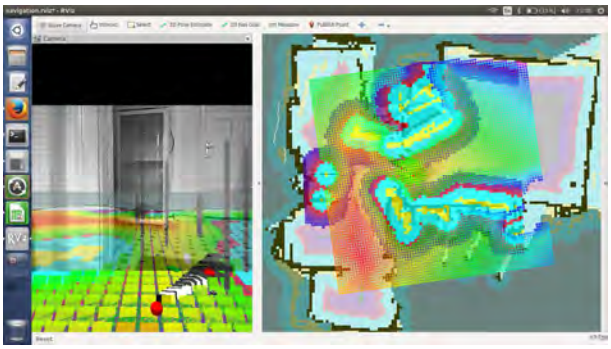

*Figure 9.* ***Robot movement in complicated environment***

The robot follows the generated path not exactly, but thanks to the algorithm it don't miss the path and correct its position, orientation and movement.

## CONCLUSIONS

For the mobile robots is very important to be autonomous. That's why they have to be equipped with sensors for navigation and obstacle avoidance. Navigation provide safety and good positioning to the robots and is very applicable in inspection. Using navigation we don't have to be worry about how to control the robot by joystick or phone, we just set desire point and the navigation do its job. This is done because of the improved algorithms and technics for better navigation and mapping.

## REFERENCES


Bräunl, Thomas (2008). *EMBEDDED ROBOTICS- Mobile Robot Design and Applications with Embedded Systems*, page or chapter numbers if relevant, Springer-Verlag, Berlin Heidelberg.

Borenstein, Johan (1996). *"Measurement and Correction of Systematic Odometry Errors in Mobile Robots"*, IEEE Transations on Robotics and Automation, Vol 12, No 6, pp. 869-880

Borenstein, J. & Koren, Y. (1990). Real-time obstacle avoidance for fast mobile robots in cluttered environments, proceeding of the IEEE International Conference on Robotics and Automation, Vol. 1, pp. 572-577, ISBN 0-8186-9061-5, Cincinnati, 13-18 May 1990, OH, USA.

Borenstein, J. & Koren, Y. (1991). The vector field histogram – fast obstacle avoidance for mobile robots, IEEE Transactions on Robotics and Automation, Vol. 7, Issue 3, June 1991, pp. 278-288, ISSN 1042-296X.

Borenstein, J. & Koren, Y.(1991). Histogramic in-motion mapping for mobile robot obstacle avoidance. *IEEE* Journal of Robotics and Automation, Vol. 7, No 4, 1991, pp. 535-539, ISSN 0882-4967. International Journal of Advanced Robotic Systems, Vol. 6, No. 2 (2009) 108.

Encoder sensor: **http://www.dynapar.com/technology/absolute-rotary-encoders/**.

Foote, Tully (2013). tf: The transform library. Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on, ISSN = 2325-0526}.

Fox, D.; Burgard, W. & Thrun, S. (1997). The dynamic window approach to collision avoidance, *IEEE* Robotics & Automation Magazine, Vol. 4, Issue 1, Mar 1997, pp. 23-33, ISSN 1070-9932.

Kinect Sensor:

**https://msdn.microsoft.com/en-us/library/hh973078.aspx#Depth_Ranges**;

**https://msdn.microsoft.com/en-us/library/jj131033.aspx**.

Ko, N.Y. & Simmons, R. (1998). The lane curvature method for local obstacle avoidance, Proceeding of the IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS'98, pp. 1615-1621, Victoria, October 1998, Canada.

Khatib, O. (1986). Real time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research*, Vol.5, Issue 1, Spring 1986, pp. 90-98, ISSN 0278-3649.

Mapping: **http://wiki.ros.org/slam_gmapping/Tutorials/MappingFromLoggedData**.

Minguez, J. & Montano, L.(2000). Nearness diagram navigation (ND): A new real time collision avoidance aproach, Proceeding of the IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS'00, pp.2094-2101, 2000, Takamatsu, Japan.

O'Kane, Jason M., (2013). *A Gentle Introduction to ROS*, University of South Carolina, Columbia.

Riisgaard, S. & Rufus, M. B. (2007).SLAM for Dummies, A Tutorial Approach to Simultaneous Localization and mapping.

Simmons, R. (1996). The curvature velocity method for local obstacle avoidance, Proceeding of the IEEE International Conference on Rbotics and Automation, ICRA'96, pp. 3375–3382, ISBN 0-7803-2988-0, Minneapolis MN, 22-28 April 1996, USA.

Seder, M.; Macek, K.; Petrovic, I. (2005). An integrated approach to real time mobile robot control in partially known indoor environments. Proceeding of the 31st Annual Conference of the IEEE Industrial Electronics Society, pp.1785.

# Оптимизация навигации мобильного робота для инспекции с использованием операционной системе для роботов

*Денис Чикуртев*

*Институт информационных и коммуникационных технологий, 1113 София*

*Р е з ю м е*

*В работе представлена роботская операционная система (ROS) для симуляционной навигации. С улучшение функциональности оптимизировани движения, точности и ориентации мобильного робота. Для локализации использован Kinect сенсор. Альгоритм навигации создает карта среды, а потом генерирует целевой путь и навигирует робот. Робот может управляться по Интернет и Wi-Fi. Сделаны експерименты при наличие препятствиях.*