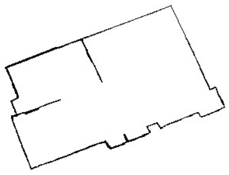
BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2017
http://excel.fit.vutbr.cz

Mobile robot navigation and objects seeking

Tomáš Sýkora*



Abstract
The goal of this project was to create a software for an autonomous mobile robot capable of navigation in indoor scenarios and of detection and recognition of the objects on its way. The most important condition which had to be met in the final solution was to find the cheapest solution possible. I achieved the given goal and its conditions using an open source software framework Robotic Operating System (ROS) and its tools. It allowed me to replace often used but quite expensive robot hardware parts with just one relatively cheap depth sensor. The final system is a set of ROS modules communicating with each other, using a simple sensor as an input data stream and a small motor which is able to send information about its velocity and rotations to the control part of the system. This mobile robot can, without any significant problems, navigate through a mapped area while it tries to find trained objects. Such a robot could be easily used to help retired or disabled people, to cooperate with industry workers or in many other fields. Of course, the robot would have to be equipped with specific mechanical parts, depending on the requirements of the specific problem.

Keywords: Robot navigation — Objects detection — Objects recognition — Objects seeking

Supplementary Material: Demonstration Video — Downloadable Code

*xsykor25@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

Autonomous and smart machines are an interesting technology area which is being used more and more in many fields. Its potential is undoubtedly huge and we are still extremely far from using any significant percentage of it. Getting these technologies closer to the ordinary people or people in need is what I was trying to achieve. The result is an open source software which can, placed to some mobile robot platform, autonomously seek known objects in a room. That can be, with specific mechanical equipment, used to fulfill many different tasks to make human life easier. The important fact here is the amount of used resources. While the common robots used by researchers can cost thousands of dollars, the components needed by the robot system described in this paper will not be more expensive than several hundreds. Because of the simpler equipment I had to choose such algorithms and techniques which would not be very compute intensive. With these basic capabilities such as the sense of direction and vision and with its low-cost requirements, many people could find this platform helpful in many different areas.

The initial idea of this project was to bring up a robot which knows where it is according to a saved map and can move from this location to a given destination on the map. Interesting extensions were added later to the original idea. That means abilities to explore indoor areas, detect objects on its way and recognize the known (trained) ones. The best final solution should do this tasks as accurately as possible in the shortest time possible.

2. Related work

There are lots of robots from different research groups capable of vision and navigation. They use combinations of several types of sensors, such as cameras, depth sensors and lidar scanners. Various softwares can be run on these platforms, including the one from this paper. ROS[1] offers great amount of packages/modules solving specific partial tasks of the navigation and vision problems from processing raw data to mapping and other more complex tasks. Some approaches use SLAM (Simultaneous localization and mapping) [1] techniques which are trying to solve computational problem of constructing or updating a map of an unknown environment while simultaneously keeping track of an agent's location within it. Although this is a widely used approach it is quite compute intensive. It continuously computes a new map based on the dynamically changing surroundings. In our use case the static map is sufficient enough. There are several approaches using static maps. The most widespread one is AMCL which implements Monte Carlo localization [2] using a particle filter to track the pose of a robot against a known map. Although these techniques are quite old, they are still widely used in the computer vision and robotics areas. One navigation solution using these approaches is available in ROS and is called Turtlebot. It even uses just a single sensor but it lacks the object detection and recognition parts.

Objects detection problems using depth data can be solved by great amount of different algorithms. All of them are provided by Point Cloud Library (PCL)[2]. Especially segmentation algorithms are important in my approach to detection. Every one of them is more or less effective in different situations and their combinations can sum up to interesting results. The objects recognition part of the system had one restrictive condition. One particular objects recognition module had to be used so there was no place for discussion or other ideas. It is the part of my solution which could be improved in the future.

For localization the solution from this paper uses a fast particle filter implementation and sensor models which always take into account the most recent state of the world. This means that if the world representation improves while the robot is running, localization becomes better. The localization module is more efficient and accurate than the well-known AMCL. The objects finding part consist of two modules, detection and recognition. Detection of the objects is done by removing the segmented floor plane from the point cloud and creating clusters for every remaining object. Images of the clustered objects are send to a recognition module which is based on the TensorFlow[3] library.

The result of this approach is an autonomous robot platform which can be used as a base for other more complex tasks such as a robot finding a specific object, grabbing it and taking it to a given destination. One can imagine many areas where this could be found interesting. Definitely there are parts where several major or minor improvements would be appreciated but the system is fully usable as it is now.

3. System architecture and ROS

Robotic Operating System (ROS) is a collection of software frameworks for robotic software development. It consists of nodes (modules) communicating with each other by sending messages of different types. A node sending a message (e.g. input data from a sensor) is a publisher. Another node can subscribe to this topic and receive its data. ROS community offers lots of packages and it is easy to create own ones. Beside publisher-subscriber communication ROS also implements several server-client types of communication. One of them is actionlib server and client. The actionlib server works on some computations or tasks. Clients can send commands to the server which process it according to the specific implementation.

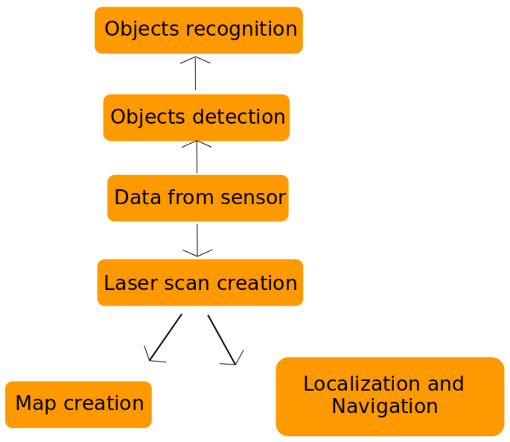


Figure 1. This is the system architecture. Input data coming from the sensor are processed by these nodes in the order of the arrows.

[1]http://www.ros.org/
[2]http://pointclouds.org/
[3]https://www.tensorflow.org/

The object finder system consists of several parts. Every one of them is implemented by one or more ROS nodes. Figure 1 demonstrates these nodes. Arrows show the direction in which data go through them. The first node takes point cloud data as an input. Laser scan is created from them which is later used to build a map and localize the robot within it. Actionlib server node explores the given area, while object detection and recognition nodes try to find known objects in front of the robot. If they are successful they send a command to stop exploration and move to the found object to the actionlib server.

4. Creation of a map using SLAM

Creation of a static map is a prerequisite for later localization and navigation (figure 2).

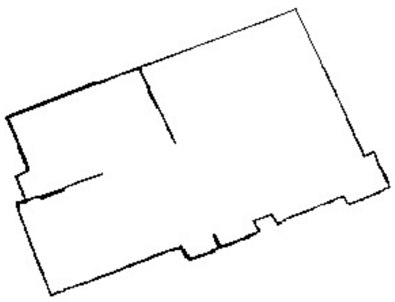


Figure 2. An example of a static map created by SLAM algorithms.

Although the navigation system uses the static map approach, it has to use SLAM algorithms to create an initial static map. It is not a problem here as it uses SLAM just once (or more times if we want to create a new map). In ROS, SLAM techniques provided by a package called *gmapping* use laser scans to build the map. Although the robot has just a depth sensor and no lidar scanner, it is possible to create a fake laser scan from the depth data. It can be done by selecting a single point from every column of the depth image. It is usually the whole middle row of the image (or another one, depends on the configurations). This functionality is implemented by a ROS package named *depth_image_to_laserscan*. Having a fake laser scan, there is one last prerequisite before building the map. It is the robot setup. That means configuring the system to know what parts capable of publishing some data is the robot equipped with and what their relative position is. It is called transformations between coordinate frames (figure 3). In our system it is setting the distances between the motor and the depth sensor frames.

These transformations are done by ROS nodes which subscribe to certain nodes (motor and sensor in

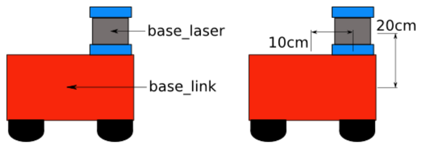


Figure 3. Data published in a certain coordinate frames such as motor and sensor frames must be transformed so the ROS knows what is their relative position to each other.

this case) publishing data with information containing distances, transform them accordingly to configured values and publish the transformed data. Now the system knows the relationships between these coordinate systems. Having the laser scan data and the transformations between the frames the *gmapping* package can start building the map. The result is an image with black and white pixels where black ones mean occupied and white ones mean free space.

5. Localization and navigation using a static map

Before the system is able to navigate itself to a given destination using a map it has to determine, where it is within the map. In other words the system has to be able to localize itself within the map. The solution described in this paper uses a fast particle filter implementation and sensor models from ROS for localization. This technique gets laser scan data as an input and tries to match them to the black pixels in the map. It is important to have a good map prepared which means clear stable objects such as walls or wardrobes without any noisy black pixels. If the map from *gmapping* was not so good, it can be easily repaired in any editor by whitening the noisy black pixels.

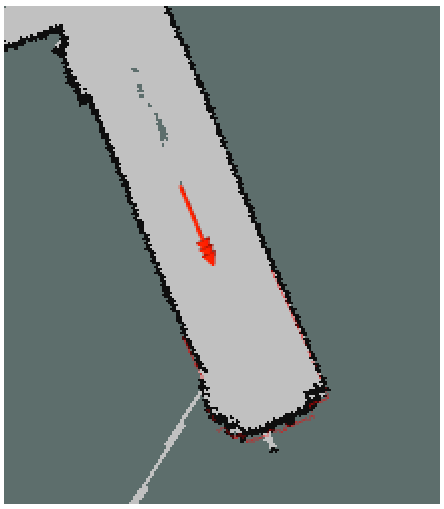


Figure 4. Monte Carlo Localization is visualized here. Amcl module tries to match laser scan data (red) to the created static map (black).

If the map is good and the localization module runs the laser scan data should match the map precisely if displayed in *rviz*. This *rviz* output can be seen in the figure 4.

Navigation techniques I used are based on dividing the whole problem of getting to a given destination into two parts. That means local and global planning of the path. Global planning means planning the path from the current position to the final destination avoiding the obstacles on the saved map. Local planning means creating a temporary map of near surroundings with all obstacles near the robot, also those which are not in the saved static map. The local planner has to avoid these local map obstacles but also it has to keep the robot as close to the original global path as possible. Starting navigation means bringing up the ROS navigation stack nodes such as *move_base*. There are several configuration files describing how should the system create these maps and how should it use them and move within them. The basic principle of this is that if the system determines that there is an obstacle in front of the robot, it adds it to the map but not just the obstacle. It adds black pixels also around this obstacle. The amount of them is configured in the configuration files. It is known as a costmap.

If the system can navigate through the known area, it can start continuously explore these areas and find objects within them. The core of exploration is another ROS package *frontier_exploration*. It can be configured to keep going through given bordered area until it "saw" every peace of it. The running exploration node is controlled by an actionlib client, which is another ROS node, which receives some data and according to them sends tasks to an actionlib server, which is the *frontier_exploration* in this case. In my solution the actionlib client tries to receives messages from the objects detector module. The detector module sends either a message whether it found some objects or not. After receiving a message about finding no objects the actionlib client sends a new goal to the server or just let the server to continue exploring. After finding an object, the client sends a stop command to the server, so the recognition module can determine what are those objects in front of the robot.

6. Objects detection and recognition

The last part of the system tries to detect objects in front of the robot, sends the message with the result so the actionlib client can make a decision whether to explore or not and if there are some objects it tries do classify them. If it finds a known one it can move to it. The detector itself if implemented as a subscriber to depth (point cloud) data. Handling point cloud data received from the sensor in real time is a compute intensive job. That is why the algorithms doing that should be well optimized. The idea used here is to segment the floor plane in the first received image.

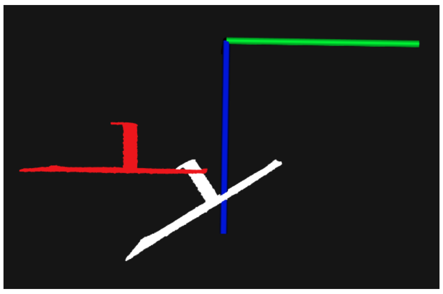


Figure 5. An optimization made to make the object detector module faster. The floor plane is segmented in the first received point cloud. The angle between the sensor and the floor is computed. Then every received point cloud (the white one in this image) can be just transformed by this angle, so one of the floor coordinates is the same value everywhere (the red point cloud). Every point with this coordinate value is simply removed. Objects which remain in the point cloud are detected objects or there is an empty point cloud.

The plane is segmented with sample consensus algorithm from PCL [3]. With the segmented floor it is easy to get the angle between the floor and the plane. Then we can just rotate every received point cloud with this angle so the floor plane has one coordinate with the same value everywhere. Now the floor plane can be removed easily from every received point cloud just by removing every point with certain value of one particular coordinate and everything else what remains in the point cloud is a detected object. This process is visualized in the figure 5 The result is that the robot has now detected objects in front of it. The angle between the sensor and the floor is checked in periodical intervals. In the next step every remaining point of the point cloud has to be added to one cluster (or doesn't if it is considered to be some noise). I used the region growing algorithm [4] here, but it doesn't really matter so much. What we want to do now is to send object images to the recognition module so the neural network can determine whether it is a known object or not. Creation of the images to be send for recognition is done by transforming 3D points of the clusters to 2d pixel coordinates of a corresponding image. That means the module has also subscribe to an image topic. So the node receives a point cloud and an image with the same timestamps, detects an object in the point cloud, transform coordinates to 2D and creates an image containing found object. This image is send to the *image_recognition* module. The module is a neural network which can be trained to classify objects in given images. The biggest probability of returned probabilities should be the one of the object in the image. If the probability is good enough, the node sends a navigation goal for the robot. The coordinates of the goal are easily computable, it is just the centroid of the specific cluster.

7. Conclusions

This paper described software creation of an autonomous mobile robot capable of navigating through indoor scenarios finding trained objects on its way. The solution had to be as cheap as possible, that means robot equipped just with necessary hardware or other components. The goal was accomplished by software tools provided by Robotic Operating System, a single depth sensor and a mobile robot base with a motor. Such a solution can be used as a corner-stone of other specific robots solving more complex tasks where finding objects is just a prerequisite. For example after adding a mechanical arm and a human computer interaction interface, it could help old or disabled people by bringing them some objects. The robot could also help industry workers or many other people in different fields.

Although this solution works well and solves the initial problem there are parts which definitely need some improvements. It is especially the recognition part, which is quite slow. I am currently studying machine learning problematics to learn more about this area.

Acknowledgements

I would like to thank my supervisor Ing. Vítězslav Beran, Ph.D. and Ing. Zdeněk Materna for their help.

References

[1] M. W. M. G. Dissanayake, P. Newman, S. Clark, H. F. Durrant-Whyte, and M. Csorba. A solution to the simultaneous localization and map building (slam) problem. *IEEE Transactions on Robotics and Automation*, 17(3):229–241, Jun 2001.

[2] D. Fox, W. Burgard, F. Dellaert, and S. Thrun. Monte carlo localization: Efficient position estimation for mobile robots. *AAAI-99 Proceedings*, 1999.

[3] R. B. Rusu and S. Cousins. 3d is here: Point cloud library (pcl). In *2011 IEEE International Conference on Robotics and Automation*, pages 1–4, May 2011.

[4] Dirk Holz and Sven Behnke. *Fast Range Image Segmentation and Smoothing Using Approximate Surface Reconstruction and Region Growing*, pages 61–73. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.