FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Robert Cesar

Planning in the context of active localization and multi-hypothesis tracking

Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the master thesis: RNDr. David Obdržálek, Ph.D.

Study programme: Computer Science

Specialization: Theoretical Computer Science

Prague 2018

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In…..... date.....

I would like to thank my supervisor, David Obdržálek, who guided me the whole time with incredible patience. I also want to thank my family for their undying support; I would never finish my work without them.

Title: Planning in the context of active localization and multi-hypothesis tracking

Author: Robert Cesar

Department / Institute: Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the master thesis: RNDr. David Obdržálek, Ph.D., Department of Theoretical Computer Science and Mathematical Logic

Abstract: In this work, a solution for global localization of small robot in ambiguous environment is proposed using "Multiple Hypothesis Tracking" approach. The ROS navigation package is extended to work with multiple possible locations of the robot over a known map. This extension manages position theories spread over the working area of the robot, and global plans navigating robot to goal from these positions. It provides them to MHT plugins that can adjust creation of global plan for robot movement based on those theories and corresponding plans. To further help with robot localization, the solution contains a modification of AMCL node from the ROS navigation package to accommodate fusion of different sensor types.

Keywords: multiple hypothesis tracking, planning, localization.

Contents

Introduction

Imagine yourself as a small robot in a huge world. You know the world, you have a map of it. You can see walls and furniture around you, you have probably travelled through here many times before, but cannot find out where you are. You check the map but that does not help – several places could fit such surroundings. Where are you? You do not remember this place from before you went to sleep. You were in a tiny room under desk where you finished your last task. Not here, something happened. Maybe the lazy IT guy that loves to ride on his chair across the room and always gets in your way moved you. Or maybe the cleaning lady thought that you did not belong to the place where she had found you. Anyway, you need to find your docking station. And rather sooner than later, because your batteries are running low…

This scenario may be a fiction but it can happen: this situation may occur when robots have to work in the same place as people do. Robots in the real world need to determine their position on a map and deal with an ambiguous environment. Be it in an open space office or in a warehouse, there are places with multiple similar looking areas that can confuse localization algorithms. Actual robots do not possess full information about themselves or their situation. They usually start without knowledge about their initial position. They must perceive nearby objects and discover their location from distances to or other properties of detected obstacles. In the case there were landmarks distributed all around, robots would localize themselves in an instant. However, it is not always possible or affordable to rebuild company premises to accommodate new tools. Still, robots have to work there and navigate themselves. It is quite usual to command the robot to localize itself in a heterogenous environment, where it can be achieved within few cycles of inputs from sensors. The challenging part comes when several areas of the working space of robot are the same for its sensors.

We prepared a solution that can help with navigation in such places. We augmented the sensing capability of the robot by integrating messages of all standardized sensors. In particular, we extended the popular ROS navigation package with knowledge about location uncertainty. The planning algorithm can work with

ambiguity – with multiple theories about current location in the known world. We prepared several plug-ins that can process these theories in diverse ways and you can choose among them. Do you prefer the currently most probable position or the location that is possibly the closest to the goal? Do you want to turn in the direction most of the plans would go? Or do you want to create your own planning module? With our package extension all of these options are possible.

The following text is organized as follows. In Chapter 1, we analyze the problem of a lost robot. We go through the theoretical background of this topic, the multiple hypothesis tracking theory and fusion of sensor data. Then we move to the robot operation system that our test robot uses and we end with comparison of available simulators.

In Chapter 2, we specify the goal of our work and its implications. We describe how the model robot should be equipped, which messages should be integrated and how we understand the concept of multiple hypothesis tracking.

Chapter 3 focuses on design of our solution. First, we describe components from ROS framework we need to use. Then we continue with list of sensor types and how they can help with navigation. The last part reveals our approach to multiple position hypotheses and their management.

Chapter 4 is dedicated to our implementation of multiple hypothesis tracking, sensor fusion and how we assembled the collection of ROS nodes into working solution.

The last chapter summarizes results of our experiments and compares our solution with the current ROS navigation package.

The bibliography we used in this work consists of large amount of conference articles and online documentation. Therefore, we decided to reference sources in the footnotes instead of the usual list at the end of the work to simplify navigation from text to its source.

1. Analysis

In this chapter we will define the problem outlined in the previous chapter – how can we navigate a robot in an ambiguous environment. We will also provide a brief introduction to domains of theory that relate to our task. Localization and navigation must tackle uncertainty all the time, so in the next section, we will focus on current methods that try to solve the problem, their shortcomings and why they matter. And in the following section we will describe the Multiple Hypothesis Tracking theory and its application to our problem.

1.1. The Lost Robot Problem

In the real world the robot does not have full information about its environment. The problem arises when it is tasked with movement to any specific location – it has to start with a global localization, without information about its previous activity or the initial position. This problem is also known as the wake-up problem – the robot is aware that it is lost. Most robots cannot acquire their absolute position on demand or in a brief time span. They usually have to rely on data from their sensors and such information is just relative – it states how far the robot is from nearby obstacles. They have to deduce their position from sensed distances and stored map. Unfortunately, in a world without usable landmarks, that computation will result in numerous possible locations, among which it is impossible to choose at once. Robot with such uncertainty about its position has to move around, keep track of its possible positions, update them when new sensory data arrive and eliminate wrong options whenever possible. The problem is solved when a single position remains. After that the robot can plan a path to its destination and navigate itself there.

1.2. Introduction to theoretical background

In this section we will provide elementary information about domains of mobile robotics that apply to our problem. Next, we will present the popular open source platform in robotics which we used for implementation. And finally, we will compare available simulators for testing of our prototype.

1.2.1. Localization, Planning and Navigation

Localization, known also as position estimation, is a problem of "determining the pose of a robot relative to a given map of the environment"[1]. The robot needs to establish a relation between its local coordinates and the global (map) position. Only then it can calculate positions of known objects on the map into its coordinate system and proceed with other tasks. The localization problem itself branches into several dimensions, according to nature of problem at hand and information available to the robot.

We will focus on Active Global Localization, where "the initial pose of the robot is unknown. The robot is initially placed somewhere in its environment, but it lacks knowledge of its whereabouts."[2] The "active" part means that localization algorithms interfere with planning to augment its localization efficiency.

Path planning is "concerned with the synthesis of a geometric path from a starting position in space to a goal and of a control trajectory along that path that specifies the state variables in the configuration space of a mobile system"[3]. The planning tool requires a map of given area and a starting position. In our case, the

---

[1] THRUN, Sebastian; BURGARD, Wolfram; FOX, Dieter. *Probabilistic robotics*. Cambridge, Mass.: MIT Press, 2005, p. 191. ISBN 9780262201629.

[2] Ibidem, p. 194.

[3] GHALLAB, Malik; NAU, Dana; TRAVERSO, Paolo. *Automated planning: theory and practice*. Boston: Elsevier/Morgan Kaufmann, 2004, p. 2. ISBN 1-55860-856-7.

starting position is not sure. Therefore, to paraphrase Automated planning[4], the single prepared plan may result in various paths during execution.

Navigation can be viewed as a composition of localization, map building and path planning. In our case, with already prepared map and defined competences of localization and planning, all that remains is just a process of following the prepared plan in local coordinates.

More comprehensive explanation of these topics can be found in the books Probabilistic Robotics, chapter 7[5], and Automated Planning[6].

1.2.2. Multiple Hypothesis Tracking

The idea of Multiple Hypothesis Tracking (MHT) was first published by Reid in 1979[7]. This approach enumerates and then manages all possible theories instead of directly filtering to a single option (e.g. using Extended Kalman Filter[8]). The original idea aimed at multiple target tracking for both military and civil use (e.g. air traffic control or ocean surveillance). Later it was used in robotics for the global localization problem, e.g. to reduce localization errors[9]. The research in the last years

---

[4] Ibidem, p. 376.

[5] THRUN, Sebastian; BURGARD, Wolfram; FOX, Dieter. *Probabilistic robotics*. Cambridge, Mass.: MIT Press, 2005, pp. 191-236. ISBN 9780262201629.

[6] GHALLAB, Malik; NAU, Dana; TRAVERSO, Paolo. *Automated planning: theory and practice*. Boston: Elsevier/Morgan Kaufmann, 2004. ISBN 1-55860-856-7.

[7] REID, Donald. "An algorithm for tracking multiple targets," in *IEEE Transactions on Automatic Control*, vol. 24, no. 6, pp. 843-854, Dec 1979.

[8] THRUN, Sebastian; BURGARD, Wolfram; FOX, Dieter. *Probabilistic robotics*. Cambridge, Mass.: MIT Press, 2005, pp. 201-214. ISBN 9780262201629.

[9] REUTER, J. “Reducing localization errors by scan-based multiple hypothesis tracking”, *Computational Intelligence in Robotics and Automation*, 1999, Monterey, CA, 1999, pp. 268-273.

turns back to multiple target detection and tracking, for example in multi-target multi-camera scenario[10] or detection and tracking in crowded areas[11].

When we apply this theory to our problem, then hypotheses should be generated for all possible locations and orientations. With each new measurement, probabilities of all stored hypotheses will be recalculated. They can be eliminated when proven false, joined with another theory for converging to the same pose or just updated in the list for use in later cycles. After a sufficient movement of the robot there will remain only a single theory – the true one.

There are two main approaches to implementation of the MHT model. The first one follows the idea stated above – the planning algorithm prepares a path that will allow robot to remove every false theory and only then, when a single theory remains, will navigate robot to its goal. The elimination can be done by finding a unique location on the map and reaching it with robot; then the location is sure. Other possibility is to find paths that only some plans from the remaining theories can follow, thus elimination others. Thirdly, it is also possible simply to navigate the robot to a colliding course with obstacles that the plans from maintained theories try to avoid – when an obstacle is actually found, those plans that did not avoid it can be dropped. This approach guarantees a certainty, that the position the robot reached is the desired goal, which may be necessary in some scenarios. However, this approach is both time and computationally consuming. In some scenarios an approximation could suffice – that the position reached is just an estimated goal position. The second approach to MHT model exploits this option with less time and computation resources consumed. With this approach a most trusted plan is chosen and the elimination of false theories is performed during its execution as a side effect. The plan can be chosen based on the probability of the corresponding theory, which is also the only approach navigation tools without MHT can use. Alternatively, the

---

[10] YOON, Kwangjin; SONG Young-min; JEON, Moongu. "Multiple hypothesis tracking algorithm for multi-target multi-camera tracking with disjoint views", in *IET Image Processing*, vol. 12, no. 7, pp. 1175-1184, June 2018.

[11] CHEN, J.; SHENG, H. et al. "Enhancing Detection Model for Multiple Hypothesis Tracking", *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, Honolulu, HI, 2017, pp. 2143-2152.

shortest path can be chosen to try to reach the goal as soon as possible, or the plans can be fused in some way to produce their "representative" plan, for example by choosing a short path that majority of plans would follow. This approach may fail (the robot can reach a target that is not the demanded goal), but in some scenarios, it is acceptable, for example when the goal position can be detected when it is reached. In that case, reaching a false goal soon, replanning and reaching a new goal can be faster than eliminating all position hypotheses beforehand.

1.2.3. Sensor Fusion

The goal of sensor fusion is to provide the robot with measurements from different sensors to broaden its knowledge about its environment. Observations made by single sensor suffer with various issues – they can contain noise, they may not see everything the robot needs, sometimes the sensor fails entirely. To overcome these shortcomings, fusion of multiple inputs is used. It "combines data from multiple sensors, and related information from associated databases, to achieve improved accuracies and more specific inferences than could be achieved by the use of a single sensor alone"[12].

In our case, a GPS sensor would suffice for localization of the robot. However, it cannot be used indoors and sometimes even outside it can be inaccurate (e.g. a street between tall buildings). Another powerful sensor, the Kinect from Microsoft[13], can provide an RGB image of its surroundings with distances to seen objects, but it fails entirely under direct sunlight. Therefore, different sensor types should be available for the robot with option to prefer or ignore some of them.

---

[12] HALL, D.; LINAS, J. "An Introduction to Multisensor Data Fusion", *Proceedings of the IEEE*, vol. 85, no. 1, pp. 6–23, 1997.

[13] Kinect Development Center, "Kinect – Windows app development" [online], https://developer.microsoft.com/en-us/windows/kinect, July 2018.

1.2.4. Robot Operating System

The Robot Operating System (ROS) is an open-source framework in robotics. "It provides a structured communications layer above the host operating systems of a heterogenous compute cluster"[14]. More specifically, it "provides libraries and tools to help software developers create robot applications. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more"[15]. An independent team can implement custom modules and easily integrate them into the existing system.

We chose this framework, because it allows free sharing of developed code and techniques among teams around the world to enhance abilities of our robotic systems. It provides a fully operational system for controlling robots and manages the complex task of communication and data transfer among all participating components. It also provides tools for reasonable logging, debugging and testing of developed modules. Since ROS distributions are bonded with corresponding Ubuntu distribution, we chose the last one with Long time support[16] (Xenial Xerus[17]), the Kinetic Kame release[18]. Its end of life date is set to April 2021.

1.2.5. Simulators

We compared popular simulators for ROS environment, excluding those that cannot be used for free for educational purposes (and those with extremely limited

---

[14] QUIGLEY, M. et al. "ROS: an open-source Robot Operating System". In *ICRA workshop on open source software*, volume 3, 2009.

[15] Robot Operating System Wiki, "Documentation – ROS Wiki" [online], http://wiki.ros.org, July 2018.

[16] List of Ubuntu distribution releases, "Releases – Ubuntu Wiki" [online], https://wiki.ubuntu.com/Releases, December 2016.

[17] Ubuntu 16.04.4 Xenial Xerus LTS, available online at http://releases.ubuntu.com/releases/16.04/

[18] Kinetic Kame distribution description, "kinetic – ROS wiki" [online], http://wiki.ros.org/kinetic, December 2016.

trials, like Construct Sim, that allows only 10 hours of simulation per month[19]). We are left with the V-Rep simulator[20], the Player-Stage-Gazebo set[21] and a standalone Gazebo simulator[22].

The Player-Stage-Gazebo set used to be the major simulator for ROS, directly integrated in the ROS environment. Unfortunately, its development was terminated, the last update was released on 26th November 2010[23]. Using such outdated version with current release of ROS could cause unforeseen issues and just launching the tool would require a trying change of configuration. Therefore, we abandoned this simulator.

The V-Rep simulator provides an intuitive GUI for managing and launching simulations. There is also an interface for communication with ROS components (actually, there are three versions with different functionality, see[24]). However, any communication outside the editor must be scripted, sensor messages translated to ROS messages, etc. The amount of work required for our simulations would be tremendous, so we have forsaken this tool too for the last alternative.

The Gazebo is a stand-alone tool for robot simulation. It provides a GUI ("client") for monitoring and editing the simulated world that is run by the Gazebo server. With the interface package for ROS[25] it is also possible to control the simulation from ROS environment by standard tools (ROS messages and services). The main benefit versus the V-Rep simulator is an extensive list of user-made

---

[19] Simulator and development tools Construct Sim, available online at http://www.theconstructsim.com

[20] Coppelia Robotics V-Rep, available online at http://www.coppeliarobotics.com/

[21] The Player Project, available online at http://playerstage.sourceforge.net

[22] Gazebo, available online at http://gazebosim.org/

[23] The Player Project repository [online], https://sourceforge.net/projects/playerstage/files/, July 2018.

[24] V-Rep ROS interfaces, "ROS Interfaces" [online], http://www.coppeliarobotics.com/helpFiles/en/rosInterfaces.htm, July 2018.

[25] Gazebo ROS interface, "gazebo_ros_pkgs – ROS Wiki" [online], http://wiki.ros.org/gazebo_ros_pkgs

libraries for countless drivers used in simulations. Because of that we can use all necessary components with just few extra lines for linked libraries.

1.3. Related work

1.3.1. Current sensor fusion in ROS

There are many sensor fusion techniques already implemented in the ROS framework. But they were developed for specific purposes and cannot be used generally as the rest of the standard navigation package[26]. They expect specific sensors or use fixed filters for position estimation[27][28][29], or their configuration requires extensive programming[30][31]. To the best of our knowledge, these is no ROS package that would provide easy direct integration of multiple sensor inputs of different type to the navigation package.

1.3.2. Current approach to localization uncertainty

Solving of the localization problem requires either tracking of multiple hypotheses of possible robot locations, or a quick reduction to a single theory. That

---

[26] ROS navigation package, "navigation – ROS Wiki" [online], http://wiki.ros.org/navigation, July 2018.

[27] Robot localization package, "robot_localization – ROS Wiki" [online], http://wiki.ros.org/robot_localization, July 2018.

[28] Ethzasl sensor fusion package, "ethzasl_sensor_fusion – ROS Wiki" [online], http://wiki.ros.org/ethzasl_sensor_fusion, July 2018.

[29] EKF sensor fusion package, "robot_pose_ekf – ROS Wiki" [online], http://wiki.ros.org/robot_pose_ekf, July 2018.

[30] Bayesian filtering library, "The Orocos project" [online], http://www.orocos.org/bfl, July 2018.

[31] RATASICH, D. et al. "Generic sensor fusion package for ROS", *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Hamburg, 2015, pp. 286-291.

reduction is usually done by some implementation of the Extended Kalman Filter[32] over known features in the environment, or by using the Monte Carlo Localization[33] method and viewing only the dominant theory in the particle filter. The popular standard navigation package[34] uses the later – selects the currently most probable theory and ignores the rest. Drawback of the single theory approach in an ambiguous environment is that there are multiple location options at the beginning with equal probability. Choice among them is actually made randomly – because of sensor noise, a dynamic obstacle, or just because of stochastic nature of Monte Carlo Localization algorithm. Henceforth, all decisions are made according to the chosen theory and it is unlikely that the localization process would roll back to another theory. Problem could be solved by tools for detection and resolving of the kidnap problem, when the chosen theory reaches a dead end and no valid theory remains. However, all progress is lost at that point and localization process starts anew.

Tracking of multiple hypotheses has been described in many papers before, but always for a specific scenario. These works require certain features that robot will detect and compare with stored map for its localization. For example, Jensfelt and Kristensen used walls and doors for active localization[35], Choi, Kim and their team utilize "lines and points" landmarks[36] (walls and circles on the ceiling), Ito and his team exploit WiFi signal strength[37]. Again, to the best of our knowledge, there is

---

[32] THRUN, Sebastian; BURGARD, Wolfram; FOX, Dieter. *Probabilistic robotics*. Cambridge, Mass.: MIT Press, 2005, pp. 201-210. ISBN 9780262201629.

[33] FOX, Dieter; THRUN, Sebastian et al. Monte Carlo Localization: Efficient Position Estimation for Mobile Robots. *Proceedings of the National Conference on Artificial Intelligence*, 1999, pp. 343-349.

[34] ROS navigation package, "navigation – ROS Wiki" [online], http://wiki.ros.org/navigation, July 2018.

[35] JENSFELT, P.; KRISTENSEN, S. "Active global localization for a mobile robot using multiple hypothesis tracking", in *IEEE Transactions on Robotics and Automation*, vol. 17, no. 5, pp. 748-760, Oct 2001.

[36] CHOI, H.; KIM, E. et al. "Multiple hypothesis tracking for mobile robot localization", *Proceedings of SICE Annual Conference*, Akita, 2012, pp. 1574-1578.

[37] ITO, Seigo et al. "W-RGB-D: Floor-plan-based indoor global localization using a depth camera and WiFi", *IEEE International Conference on Robotics and Automation*, Hong Kong, 2014, pp. 417-422.

no paper that would try to solve the Global Localization problem with a general multiple hypothesis tracking model.

There are three options how to localize the robot. The easiest one is to ignore position uncertainty and just use the most probable model. The second approach directs to elimination of false hypotheses and navigate to the goal. The last option tries to navigate to the goal with all theories intact – it lets the theories to be removed naturally during navigation and use the remaining ones to "vote" for the next movement. Since every one of these options can be beneficial in different situations, we decided to provide plugins for each of them.

2. Specification

In this chapter we will describe the problem we decided to solve in our work and some tasks related to it. We will also try to encompass implications of said problem to provide boundaries for prepared solution and requirements for developed tools.

Our goal is to navigate a robot while using the concept of multiple hypothesis tracking. We need to design and implement a solution that can maintain an overview of scattered positions on the map (that may represent actual position of the robot) and that would benefit from the MHT theory. That solution should actively direct path of the robot to eliminate false hypotheses or to confirm and strengthen a valid estimation of location of the robot.

It would be helpful to use some sensor fusion technique for better localization and navigation, so that the robot can use more versatile instruments for sensing its surroundings. For that, we need to implement a tool that would encompass various sensor types used in robotics and use measurements stored in their messages for navigation of our robot.

To prove the validity of our approach we need to program a prototype of such guiding system in the Robot Operating System (ROS) environment. To provide benefit to the widest range of users possible, we will extend some existing navigation package or create our own package and test the solution in the ROS environment.

2.1. Expected restrictions

We aim on small affordable robots that cannot bear all the equipment its owner would like to use. They have small base which cannot reasonably carry too many sensors since they would not fit on their platform. It is also not likely that these robots would have expensive 3D scanners that would give them a precise picture of their surroundings. In other words, the expected robot will be small and with only few common sensors.

The real constraint lays in the working space of the robot. The goal is to be able to navigate robot in an already existing ambiguous indoor environment (e.g. warehouse with multiple shelves). If it was outdoors, the GPS would solve most problems (in most cases) by itself right away. If there was a possibility for it, the best option would be to artificially add landmarks to the working space of the robot. Then it would be easy for robot to navigate itself from any location anytime. To give some examples of such modifications, there could be QR codes on the floor, Bluetooth beacons on crossroads of warehouse alleys or magnetic lines hidden in a concrete floor. Sometimes there are such landmarks already in place, but small robots cannot read them (e.g. cards on the office doors – they are simply too high and too small for the robot to read). However, any reconstruction is expensive and usually not compatible with any ongoing activity that business requires there, so the owners cannot change the area to accommodate needs of their robot.

To summarize the expected restrictions, the robot will work in a stable indoor environment (it can use a map of it) with many similar locations and sparse or no landmarks. It will have several common sensors for monitoring its environment, but it will not get full information about its current location.

2.2. Messages from sensors in ROS environment

Part of our goal is to fuse data from various sensors. The idea is that the more sensors we can integrate into our solution (sensors that make sense to attach to a mobile robot, that can help with navigation), the more information our robot receives about its surroundings, the better estimation it can make about its location. There is also the aspect of versatility – with elaborate sensor fusion the robot does not have to rely only on a single sensor type or a limited preconfigured set of them. Last, but not least important reason for sensor fusion is that in certain weather or area conditions some types of sensors are less precise or downright harmful to the localization process. In such situation, the ability to replace data from such useless sensor with data from another sensor type is truly priceless.

In the ROS framework, there is a package of standardized messages from numerous types of sensors[38]. Some of them might help with localization in very specific scenarios (e.g. FluidPressure message could be used by automated underwater vehicle[39], but it is worthless for earth-bound robots), few are not at all relevant for navigation (e.g. Joy message that holds information about joystick axes). But there are several types that are widely used in robotics, like LaserScans or NavSatFix messages (data from GPS). The created module should at least fuse all mainstream sensor types and preferably offer a way to integrate any other types of sensors that could be used in localization of the robot.

Under certain conditions even very specific sensors can support localization. In case there is a map with separated areas of effects that these sensors can perceive, the robot could use such information for more precise position estimation. The Illuminance messages can be used when positions of light sources are known, the same is true for an equivalent map with magnets and the MagneticField messages. The RelativeHumidity and the Temperature messages would require stable conditions to be useful, but it is possible with similar map as in the previous examples. We have already mentioned the FluidPressure messages for the underwater vehicles. And finally, there are the Image and the ConpressedImage messages from cameras. They can provide the most detailed visual information among all standardized messages from sensors, but their messages are too large for a common processing (only few seconds of streamed Images took dozens of megabytes of space). Therefore, they are converted to more abstract types, like the PointCloud or even the LaserScan messages.

We consider the following ROS message types vital for integration: Imu (relative changes measured by the inertial unit), LaserScan (scans from a laser sensor), PointCloud2 (extracted objects from depth images and camera feeds), Range (findings from infrared and ultrasonic sensors) and NavSatFix (positions from GPS, Galileo etc. sensors). Imu (or just odometry) with one of these sensor types and

---

[38] List of ROS sensor message types, “sensor_msgs – ROS Wiki” [online] http://wiki.ros.org/sensor_msgs, July 2018.

[39] Autonomous underwater vehicle, “Autonomous underwater vehicle – Wikipedia” [online], https://en.wikipedia.org/wiki/Autonomous_underwater_vehicle, July 2018.

bumper/cliff event management are sometimes the only sensors the robot has and it can still survive reasonably well. However, there are environments where each of these types will fail when the robot has to rely just on them. For example, the GPS sensor usually moves in random nearby space when the robot is placed in any indoor "world" and thus the sensor is inaccurate (at best). Another example, an open space office, provides many similar looking surfaces for long range sensors and no landmarks. Combination of said (general) sensors with some of the less common (specific) ones could eliminate ambiguity and greatly help with the navigation of the robot. For other cases, where even combined sensors are insufficient, there is just the MHT approach.

2.3. MHT

The multiple hypothesis tracking model allows, as its name conveys, in contrast to other localization techniques, to monitor numerous possible positions of the robot. The robot does not have to immediately decide where exactly it is. It maintains all those position theories until they are disproved by adequate amount of contradictory sensory reading. Probability of each hypothesis can vary, they can move up and down on the likelihood ranking until they get under a given threshold. Such theory would then get discarded from memory – this option was refuted. Even the recently best hypothesis can be disproved in few cycles by sufficient data.

The robot can navigate itself by the most probable theory, by sum of all theories positions or any other means, as we described in the section 1.2.2 – Multiple Hypothesis Tracking. It can benefit from MHT as long as it handles all of its possible locations. It even does not have to use more than one of them at a time – it could be using just the once best valued hypothesis and stay with it until it's disproved. Because all maintained theories have a "history" – they were repeatedly confirmed by data in the previous rounds and thus should be a better estimation than any new theory created when the previous one failed.

To use this model the robot only needs to appropriately initialize its internal buffer of theories and guarantee their management. The stored positions must be

updated frequently as the robot moves and the hypotheses “quality” should be recalculated regularly to trim the list of wrong estimates. In that way the robot can benefit from knowledge about its all possible locations.

3. Solution design

In this chapter we will introduce techniques that we use to solve problems with localization of the robot in an ambiguous environment. There are many solutions for noisy sensor readings and position uncertainty. Our main goal is to provide robots with a choice, so they may find possible paths from calculated positions and decide which path is the best for them. To do so, we need a planning tool aware of multiple theories and capable of handling them. Moreover, the robot needs to be able to process data from all sensors available. And finally, there should to be a way to extend both the planner and the list of recognized sensors in the future.

First, we will describe fundamental components of Robot Operating System (ROS) framework[40] that our solution uses or extends. It would be useless to define every aspect of the ROS environment and its shared structures since there is already a comprehensive documentation available online. Next, we will focus on necessity of a true sensor fusion. And finally, in the end of the chapter we will explain our approach to the Multiple Hypothesis Tracking (MHT) model.

3.1. Components of ROS framework

Design of our solution is built upon the ROS with intent to extend (not replace) the current solution to allow consideration of multiple position theories during path planning. This way, solutions of other teams that use the navigation package can benefit from our work without necessity of reworking their set of ROS components. As such, the whole navigation framework and basic ROS environment structures are used. In this section we will describe just the most important parts of framework that we used or modified.

[40] Documentation of the ROS framework, "Documentation – ROS Wiki" [online], http://wiki.ros.org/, July 2018.

3.1.1. Move base

The move base node is the keystone of all movement of the robot. This node receives request to reach certain destination and oversees all necessary processing to get robot to desired place. It manages both global and local planner provided at launch time. It orders a path from estimated position of the robot to given goal from the global planner and passes that path to local path planner, from which it requires instruction for each next step. It conveys this instruction to nodes handling motor activity. It also controls the outcome of the processed plans and whether the robot got stuck, in which case it starts predefined countermeasures.

This tool already provides slots for modules for global or local planners. The only remaining task is to handle situation with no position hypothesis (with too many theories actually), as we discussed earlier. The most effective way to distinguish between standard planning and such situation, when robot does not know almost anything about its location, would be to reimplement processing of the set of actions control-plan-execute-recovery. However, this would disrupt ordinary processing of the navigation package and our solution could not be used as a direct replacement by other teams. Therefore, we will work within boundaries of the current system of modules in the move base instead of changing it. A custom global planner can be created and that can distinguish between a state with few theories and another state with hundreds or more theories. It would in turn use modules for specific planners, one for each state. In this way the base's ability to detect stuck robot and respond to it is preserved.

The current Global planner library[41] looks for the best path from starting point to target location over occupancy grid. It can use either A* or Dijkstra algorithm, follow grid boundaries or gradient descent and quadratic approximation can be turned on in its configuration. This tool is used as a default global planner in the navigation package.

---

[41] The Global Planner documentation, "global_planner – ROS Wiki" [online], http://wiki.ros.org/global_planner, July 2018.

The present default local planner[42] uses Dynamic Window Approach[43] to local navigation – it generates a valid search space and then looks for optimal solution in that space. This tool receives path plan from global planner and issues velocity commands to the base. It reads from odometry and obstacle topics to check execution of sent commands and to avoid eventual dynamic obstacle.

3.1.2. AMCL

The AMCL[44] node implements Adaptive Monte Carlo localization (also known as KLD-sampling)[45] algorithm as described by Dieter Fox. It maintains a particle filter with samples distributed among possible locations on the map. It updates those samples with data from odometry readings and sensor messages. Finally, this tool distributes positions of all samples and robot position estimate.

Particle updates with laser scan messages are performed using algorithms Sample Motion Model Odometry, Beam Range Finder Model or Likelihood Field Range Finder Model[46]. One of these models is selected at launch of the robot (in configuration) and each sample is updated by it with every received message.

We suggest two extensions of this module. The first extension would broadcast positions of clusters. These are already internally used for KLD-sampling (so no extra computation is required) and they are essential for MHT planning. The second one concerns the accepted sensor message types – this tool reads data only from laser messages and no other sensor type can be processed. Therefore, handlers

---

[42] DWA local planner documentation, "dwa_local_planner – ROS Wiki" [online], http://wiki.ros.org/dwa_local_planner, July 2018.

[43] Dynamic window approach, "Dynamic window approach – Wikipedia" [online], https://en.wikipedia.org/wiki/Dynamic_window_approach, July 2018.

[44] AMCL node documentation, "amcl – ROS Wiki" [online], http://wiki.ros.org/amcl, July 2018.

[45] FOX, Dieter. "KLD-Sampling: Adaptive Particle Filters", In *Proceedings of the 14th International Conference on Neural Information Processing Systems*, pp. 713-720, December 2001.

[46] THRUN, Sebastian; BURGARD, Wolfram; FOX, Dieter. *Probabilistic robotics*. Cambridge, Mass.: MIT Press, 2005, pp. 136, 158 and 172. ISBN 9780262201629.

for range sensors and absolute position messages should be integrated, as well as general sensor message processing instead of fixed laser dependency.

3.1.3. Sensor fusion

As we described earlier, the AMCL node is built upon laser sensor messages. We already suggested extending this tool with handlers for range and satellite (NavSat) sensors message types as two completely different examples of sensors with specific processing. Other messages can be translated by existing tools (called convertors) to one of the already accepted message types. For example, the PointCloud to LaserScan convertor[47] allows easy integration of RGBD devices (such as Kinect) to existing solutions.

3.2. Sensor fusion

The AMCL is currently build on the laser sensor messages. We want to let the robot use all types of standardized sensor messages – either by a hardcoded handler, or with a help from some translating module. Either way the current solution must be extended to accept other types of sensor messages for localization updates.

Output from some sensor types can be converted to a laser message with minimum precision loss (e. g. a PointCloud message cut at specific horizontal line gives the same result as a laser sensor would "see" in case it was there). Translating other sensor types would be too expensive – it would result in too many iterations of processing or it would mean a significant loss of data. Finally, some sensors require a different handling altogether.

Some sensors provide information through other means than messages, for example events sent by triggered bumper or cliff sensors. Such data must be

[47] Pointcloud to LaserScan convertor documentation, "pointcloud_to_laserscan – ROS Wiki" [online], http://wiki.ros.org/pointcloud_to_laserscan, July 2018.

collected by other means than sensor fusion listeners, ideally by a dedicated node, for example Safety controller for Kobuki[48].

3.2.1. Range sensors

Range sensors cover an arc and, in contrast with laser sensors, they usually report only the closest object seen inside that arc, without any idea concerning the angle from the sensor. The only additional information they provide is size of the scanned arc boundaries. Such detection of a single obstacle (or first few targets) should be reverse-counted by some sort of a flood-fill algorithm. With that you can find the closest expected object (or objects) on the known map from the maintained position of the robot, count its distance from sensor and finally compare it with sensor readings to evaluate correctness of your expectation. This type of message is actually much more demanding on computing power than a broad laser scan. When you consider the amount of information gained (i.e. points seen by the sensor) versus the amount of checked map positions, the range sensor messages come out truly inferior to the laser messages. But these sensors are cheaper and can be mounted on affordable robots, so we need a way to work with them.

It is possible to create a set of range sensors linked to work as one. Such set can provide processed readings with multiple targets and augmented results (known angles towards these targets) and these messages can be converted to the PointClound messages, for which there are already tools for processing. However, this is not a common case, because it is easier (and cheaper) to add a single laser scanner than to configure a set of range sensors. Either way, as we stated above, the range sensor messages need a custom processing algorithm that manages the provided information.

---

[48] Kobuki safety controller documentation, “kobuki_safety_controller – ROS Wiki” [online], http://wiki.ros.org/kobuki_safety_controller?distro=kinetic, July 2018.

3.2.2. Satellite sensors

In the case of satellite messages, the necessity of different processing is even more obvious. These sensors give absolute position information and therefore neither the current solution nor any transformation can be used. There are many types of satellite messages (there are currently 4 defined constants for existing Global Navigation Satellite System signal types within Satellite message[49]). But all of them uphold to the given standard so the real difference would be in sensor itself than in the positioning system. Sadly, the true issue with this type of sensors is caused by properties of satellite navigation – there is very limited use for GPS when robot moves indoors. On the other hand, these messages are the easiest source for localization when the robot travels outside and thus we cannot omit them completely.

Beside longitude and latitude sensor can provide altitude and covariance matrix about readings precision. Unfortunately, there are both real and simulated sensors that pass only default values in these attributes, so we cannot rely on them. However, when the information about confidence in calculated position is provided, we can use it inside position estimation updates. It requires specific message handler, that will iterate over theories of estimated positions of the robot and grade them according to their distances from current message position. This classification can be adjusted by the trust of the sensor in position precision, leading into handler usable even with noisy sensor reading.

There is also another opportunity for satellite messages (and absolute position messages in general) to be incorporated in position estimation of the robot. They can affect the updating phase of AMCL together with odometry readings. Tools that can fuse multiple odometry inputs[50] can add these messages to the data flow they process. They only need to remember previous position from certain sensor to create relative position update. They already should store such information given the

---

[49] NavSatStatus message documentation, "sensor_msgs/NavSatStatus Documentation" [online], http://docs.ros.org/jade/api/sensor_msgs/html/msg/NavSatStatus.html, July 2018.

[50] Robot localization package documentation, "robot_localization – ROS Wiki" [online], http://wiki.ros.org/robot_localization, July 2018.

various update frequencies of sensory readings they need to join in one robot position update feed for localization mode.

3.2.3. Other sensors

In specific scenarios can even some "exotic" types of sensors be useful. For instance, strength of Wi-Fi signal can be used in office with known positions of Wi-Fi routers[51]. Or magnetic lines in factory floor can help robots to "stay in line", so to say[52]. Therefore, we cannot limit of the list of processed message types to the currently most popular sensor types, because even the list of defined ROS messages[53] can be extended. So, as we said earlier, there must be a way to add new sensor handlers or at least convertors of new sensor messages.

3.3. Multiple Hypothesis Tracking

The present implementation of Adaptive Monte Carlo Localization (AMCL[54]) uses standard Particle Filter localization boosted with ability to detect the kidnap problem, at least in theory. The particle filter inherently allows to track multiple theories, since it consists of thousands of "particles", each representing estimation of position of the robot and orientation in the known world (using a map prepared before launch of the robot). However, this opportunity is totally ignored by the current solution. Particles aggregate themselves in clusters during cycles of the localization algorithm. These clusters can be presented to the robot (or rather they

---

[51] ITO, Seigo et al. "W-RGB-D: Floor-plan-based indoor global localization using a depth camera and WiFi", *IEEE International Conference on Robotics and Automation*, Hong Kong, 2014, pp. 417-422.

[52] Magnetic Guide Sensors, available online at https://www.roboteq.com/index.php/roboteq-products-and-services/magnetic-guide-sensors, July 2018.

[53] List of ROS sensor message types, "sensor_msgs – ROS Wiki" [online] http://wiki.ros.org/sensor_msgs, July 2018.

[54] AMCL node documentation, "amcl – ROS Wiki" [online], http://wiki.ros.org/amcl, July 2018.

can be exposed in the ROS environment to other nodes). Planning algorithm can then consider these theories and provide a better plan.

Nowadays the default global planner only uses the weighted center of all particles, disregarding any clustering. Such attitude loses any advantage clusters could provide. A planner, that would consider multiple hypothesis, requires publication of said clusters and their weights from AMCL. It is already possible to create a custom planner and link it to the move_base node as a module. Such module then can provide plans from given position with knowledge about obstacles in the map. There is one more weakness though, when the robot starts or when it realizes it was kidnapped. The particles of estimated positions should be distributed across all possible positions on the map. In such situation there are thousands of theories which are not clustered at all, since the robot has not moved yet. Planning should switch to another planner suited just for these occasions – it would prepare a short plan with few simple moves that would allow robot to inspect its surroundings, after which a true plan with fewer hypothesis could be made. Current move_base node cannot handle it, so either it should be extended or a smarter planner needs to take its place. Such a "smart" planner would distinguish between several clusters and hundreds of them and prepare appropriate plan – it would either call a real planner or a starter planner as described earlier.

A starting planner should prepare a simple plan with few steps, that would move robot nearby the beginning position. Such plan should rotate the robot or traverse a small distance, so that all sensors can send several messages and particle filter can be updated. The filter should converge within several cycles to dozens of clusters or less and then we can have a true plan prepared.

A real global planner (for an actual plan to the goal) should be able to consider all clusters and their weights. It would be unnecessarily slow to process more than dozens of theories, therefore there ought to be a separate planner (e.g. the starter planner described earlier) to trim these false theories. There are many ways to handle multiple position theories (as we described in the Chapter 1), so we need either multiple planner modules or the planner should be able to change its path selection strategies according to a configuration.

3.3.1. MHT solving

Current path planners do not try to work with multiple theories, they only prepare a plan from the most probable position. In the case when that estimation was false and the plan fails, the planner just creates a new plan. Robot can actually get lost forever (until restart) if the true position is dropped because of low starting probability. We propose a global planner, that creates plans for each position hypothesis and that chooses final plan according to selected strategy. When preparing strategies, we focused on the second approach covered in Chapter 1, section 1.2.2, since it is more relevant to the expected usage than exhaustive computation required by the first approach (while modules for it can still be implemented by other parties). All strategies depend on elimination of wrong plans by updates of particle filter with common sensor updates. The choice of strategy helps either reaching the goal or false plan elimination. We considered the following strategies: shortest path, first fail, best option and weighted voting. The shortest plan path tactics aims to cross out false theories by executing the fastest plan possible. The policy of first failing plan selects the plan whose path would collide with obstacle the soonest when applied to other starting positions. The approach of best option chooses plan from starting position with the highest weight.

The last method checks relative positions reached by each of the plans in certain distance from their starting points. These positions are assigned values according to weights of plans that would lead to them (or theories they are based on). The desired direction is counted as a weighted arithmetic mean. This strategy then selects the plan that is the closest to that direction. Such approach should combine averaging of all available information and still result in executable plan.

3.3.2. Particle filter

The particle filter also should be modified to accommodate distinct types of sensor messages as stated earlier. Current solution is fixed to handle only laser sensor messages, even though there is space prepared for other handlers. We found out that

range and satellite sensor messages should be processed separately. These sensors require specific handlers that calculate reward for each particle with every message received. The main difference is in information provided by these messages – either absolute position, nearest point in range or multiple points seen by the sensor. Handlers that respect this diversity must be integrated into the updating phase of Monte Carlo localization algorithm.

4. Implementation

Our solution for navigation of the robot in an ambiguous environment is based on the Robot Operating System (ROS) framework where we extended the current navigation package[55] implementation. This package is a popular tool used to navigate a robot in a known world, but it works only with a single position estimation. In the first subchapter we describe how we modified several components of the existing solution to accommodate the Multiple Hypothesis Tracking (MHT) theory. Next, we define how we changed processing of received sensor messages so that more sensors can be used to update position of the robot. And in the last section we cover how we built the whole solution of ROS components.

4.1. MHT

The keystone of our work is using the multiple position hypothesis model to improve localization results of the robot. We extended the current navigation framework, namely the particle filter in AMCL node, to create, manage and publish a set of theories about position of the robot. This set can be distributed to ROS environment with every update from any sensor. There is also an option to publish only a portion of all updates (e.g. send every fifth update) to prevent overloading of processing core with too many messages caused by wide variety of connected sensors.

Position theories are created during KLD sampling in the Monte Carlo localization algorithm[56]. Each round, when update of all particles is made, a k-d tree is built[57]. We identify clusters made during this process and publish them in ROS environment (i.e. the AMCL node publishes a message with information about all

---

[55] Navigation package documentation, "navigation – ROS Wiki" [online], http://wiki.ros.org/navigation, July 2018.

[56] THRUN, Sebastian; BURGARD, Wolfram; FOX, Dieter. *Probabilistic robotics*. Cambridge, Mass.: MIT Press, 2005, p. 258. ISBN 9780262201629.

[57] K-d tree description, "k-d tree – Wikipedia" [online], https://en.wikipedia.org/wiki/K-d_tree, July 2018.

clusters – location and orientation of their representatives, weight of given cluster and total amount of clusters). Any other node connected to ROS now can receive these messages and work with them. However, we noticed that the prepared clusters do not always correspond to the distribution of samples in the particle filter. Reimplementation specific for MHT could increase precision of position estimation and would allow keeping possible candidates (theories) longer in the pool of clusters. Unfortunately, rewriting of this internal process would have a huge effect on related processes and is out of scope of our work.

4.1.1. Global planner

We prepared a global planner plug-in that subscribes to the topic with a list of position theories. This tool stores the most recent cluster message and prepares a plan when asked to by the move base node. More specifically, it delegates the planning itself to another planner – it provides plug-in slots for several planners and passes information to them when required to create a plan. It recognizes three states of uncertainty – when there are more theories than limit set in configuration; fewer theories but more than one; and one theory. The last option is in fact implemented indirectly, by the supporting planners. The situation with no theory should never happen – it means that the robot left the map and we cannot plan anything. Such situation is logged and error is returned.

The current default global planner is a reliable instrument for finding a path from single start to target position. Therefore, we use it as a default option when the robot is certain of its position. However, it cannot work with multiple theories. We prepared several planners for many theories called “starter planners”, since such situation would usually appear after reboot of the robot. They prepare simple plan with few steps that slightly move robot around. That should suffice to perceive surrounding by the sensors and eliminate most of the false theories. Our planners provide rotation on the spot, short ride straight forward and spiral rotation around starting position. It is also possible to modify the length of resulting plan in configuration – in how many circles should robot turn, how far ride straight or how many spirals it should make. Any amount and combination of these planners can be

used as a fallback option when the list of theories is not limited enough. In fact, any global planner can be placed in these “starter” slots. But their main goal is to reduce amount of theories, not cut them straight to single one, so the ordinary global planners should not be used here.

4.1.2. MHT solvers

We prepared four solvers for less uncertain state (under configured limit) each of them applies different approach to solving multiple hypothesis problem. We implemented a First-fail solver, Shortest path solver, Best option solver and Weighted voting solver. The list of solvers can be extended by implementing the solver base interface (i.e. by implementing a method that accepts a list of position theories and a target position and returns some global plan). Exactly one of these solvers must be plugged in our managing global planner to provide a concrete global plan for it.

The first fail solver checks each position from the given list and prepares a plan from that position. Then it applies that new plan to other positions and counts the first expected encounter with an obstacle for the plan. It iterates over all input starting positions, prepares plans from them, finds impacts and remembers the soonest one. This solver returns the plan that would, when applied to all possible positions at once, fail first.

The best option solver iterates over theories in the input list and searches for the one with the highest weight. It takes that theory’s position and creates a plan from it.

The shortest path solver prepares a plan for each position from the input list and returns the path with the least steps. Its goal is to reach the target as soon as possible (even if that target is incorrect).

Our last plug-in is the weighted voting planner. It creates a plan for each theory as most other planners do. But this one follows each plan until it exceeds a predefined (Euclidian) distance from its starting position and marks the spot that the

plan just reached. Out planner then stores the direction (angle) from the starting position to that spot and assigns it a value equal to the weight of the theory that was used to create that plan. The planner iterates over all theories and collects vectors that represent these directions multiplied by their theory's weight. In case that some plan would finish before reaching defined distance its vector would be multiplied by a constant from configuration (the parameter "early_end_coefficient"). Then the planner fuses these values (sums the vectors) and gains a direction that is a result of "voting" by all theories. The last step is finding a plan whose direction to the marked spot is the closest to the elected direction representative.

4.2. Sensor fusion

We modified the standard message processing used within the AMCL package. Only laser sensor messages used to be recognized and the whole package is built upon them. We generalized all message handling, so that any other type of sensor can add to the picture of surroundings that our robot is building. We used the already existing inheritance of sensor handlers – the laser sensor class (AMCLLaser) is built upon the AMCLSensor class. However, this connection is not used and all communication is handled through the AMCLLaser class. We utilized the general AMCLSensor and replaced specific message handling with it. Managing functions for any type of sensor can now be easily added to the localization process and they can help refine working theories of position of the robot with each reading. Their messages will be accepted by the AMCL node listeners and the position of robot stored in the message (either relative or absolute) will be used for updating of the internal particle filter with estimated positions samples.

As a proof of concept, we prepared sensor function handlers for range and satellite messages and attached them to the localization updates framework.

4.2.1. Range sensor messages

We implemented a flood fill algorithm to evaluate weight of samples in the particle filter. Each range sensor message that is sent to the ROS environment is registered and launches an update of the stored particles. Our algorithm searches the surrounding area of every sample in the filter for its nearest obstacle. It considers a square of provided map with robot in the center of the square and sides of the square are double of the maximal range length of the sensor. That length can be limited both in configuration and by received sensor's message. The algorithm splits the surrounding area into four parts separated by diagonals and searches these segments in rows or columns respectively in given quadrant (triangle) for better performance. The message contains auxiliary information about width of the field of view of the sensor – size of the arc in which objects are perceived. This arc also defines boundaries for our algorithm within said segments of square. The search does not stop with the first hit but runs until the nearest obstacle is found among all of the concerned sectors. Distance to this obstacle is compared with value measured by the sensor using Gaussian filter to overcome noise in the reading of the sensor.

Particles with correct (or a bit noisy) distance from some obstacle are rewarded. Smaller bonus is given also to samples with more distant target then reading expects to accept possible dynamic obstacle. Coefficients for both occasions are configurable. We added an option to set trustworthiness of a range sensor – a constant that reflects how much should the precision of distance of the sample from obstacle be reflected in weight of the sample.

4.2.2. Satellite sensor messages

Satellite sensors provide readings with absolute location of robot in its coordinates. Our handler uses similar approach as with other sensors types – information about location of the robot (longitude, latitude and altitude) from the received message is compared to the position of the sample through Gaussian filter, which yields reward to the weight of the sample. The most difficult part here lays in conversion of the coordinates of the sensor to the map coordinates of the robot. There

are libraries that provide tools for such conversions. But we used just a simple estimation for our simple handler to save the computing power – we set constants for longitude and latitude distance at 50°N for the sake of higher performance. Our goal was to provide a prototype capable of assimilating satellite sensor messages to the sensory input, to prove that the integration of absolute position updates to particle filter is possible, not to compete with existing tools.

Another option of handling messages from satellite sensor is to merge them with odometry updates. This is already implemented by the robot_localization package that collects readings from wheels, IMU and other sensors like the GPS sensor. That component joins these inputs into a single stream that can be used for odometry updates of particle filter.

4.2.3. Laser scan message

The default handling in the Beam range finder model ignored unknown (undefined) map locations. It treated every grid square that was not “free” as an obstacle. There are sometimes blind spots within the map itself, when mapping of the region was not thorough. However, these are of small consequence. The real issue lays beyond map edges – every position outside defined map is unknown. These are usually straight and as such they can resemble any long straight obstacle. Localization of the robot can be easily misled to look for such objects when robot reaches map borders.

We changed this behavior – the message handler is now informed about reaching unknown space and it can interpret the information any way needed. We adjusted parameters that rewarded closer objects (dynamic obstacles) and direct hits. Both will be evaluated as a new obstacle with lower reward in case of the foggy squares. These changes subdued border-long object misinterpretations and wrong samples now disappear quickly.

4.2.4. Sensor message convertors

There are libraries created by ROS community that allow us to use other sensor types without programming specific handlers for them. These converters are usually provided as nodelets[58] that spare computing time and memory by sharing resources of a single process. This is particularly useful for data from sensors like camera that produce large amount of data in exchange for more intricate configuration.

The crucial convertor for our prototype is the depthimage_to_laserscan[59] package that translates Image messages to Laser Scans. It lets us to easily integrate the Kinect sensor feed into the commonly used navigation solution – the navigation stack. Generally speaking, a very important convertor is the pointcloud_to_laserscan package[60], that transforms a 3D message of PointCloud2 (current version of message containing a point cloud in 3D, replaces the former PointCloud message type) to LaserScan in 2D, that we can use during localization as mentioned before. Other message types can be converted to the PointCloud2, from it to LaserScan and so they can influence our extended localization updates as well. Example library for such conversion is the Point Cloud Library for ROS[61], that provides nodelets for reading from and writing to PointCloud2, Image, bags and custom PCD format.

We only have to adjust the configuration of our ROS components to encompass other sensor messages – add lines that launch a new nodelet and its manager and then redirect the streams of sensor data to pass through it.

---

[58] Nodelet documentation, "nodelet – ROS Wiki" [online], http://wiki.ros.org/nodelet, July 2018.

[59] Documentation of convertor from DepthImage to LaserScan messages, "depthimage_to_laserscan – ROS Wiki" [online], http://wiki.ros.org/depthimage_to_laserscan, July 2018.

[60] Documentation of convertor from PointCloud to LaserScan messages, "pointcloud_to_laserscan – ROS Wiki" [online], http://wiki.ros.org/pointcloud_to_laserscan, July 2018.

[61] Point cloud library ROS interface documentation, "pcl – ROS Wiki" [online], http://wiki.ros.org/pcl_ros, July 2018.

4.3. ROS nodes puzzle

The ROS framework allows its users to assemble components from various authors from different corners of the world. It requires them to follow only few simple rules designing communication among used nodes. However, composing all those independent units is a nontrivial task. In the following sections we describe how we brought together our components with standard ROS packages and how we launched our solution in the simulated world.

4.3.1. Gazebo simulator

We used the Gazebo simulator to provide feeds for all inputs of the robot. Every sensor message and odometry feedbacks originate in the Gazebo's engine. The simulator is a standalone application with custom GUI – a client for user to see what happens in the simulated world. There are also packages[62] that wrap functionality of this application and enable its usage in the ROS environment. They are mostly used to send ROS messages from simulated sensors and to execute commands for the virtual wheels of defined robot. Those wrappers also allow us to spawn robots and other objects in the simulated world and thus create a complex and dynamic environment. These tools are the main reason why we chose Gazebo over other considered simulation platforms.

The Gazebo simulator is the incubator inside which we cultivated our solution. Every element of our test robot, the second version of TurtleBot[63], both its size and appearance, is defined in special xacro files. They allowed us to extend the provided robot base with sensors to test our sensor fusion modules. We linked five ultrasonic radars under the second plate to test integration of range messages. And

[62] Documentation for interface between ROS and Gazebo, "gazebo_ros_pkgs – ROS Wiki" [online], http://wiki.ros.org/gazebo_ros_pkgs, July 2018.

[63] TurtleBot overview, "Robots/TurtleBot – ROS Wiki" [online], http://wiki.ros.org/Robots/TurtleBot, July 2018.

we attached a GPS device on the roof of the robot to verify processing data from satellites.

Each of the mentioned sensors requires a plugin that would generate data during the simulation. These tools are provided in libraries and can be linked for Gazebo's usage in the xacro files. We used the GazeboRosSonar[64] for ultrasonic messages. It sends rays in all directions inside defined cone and returns the nearest hit – as a range sensor would do. This result is published as a Range sensor message and it is anticipated by our extension of the AMCL node.

The GazeboRosGps module that we used for generating GPS messages allows us set Gaussian noise and drift among other properties. It takes position of the robot in simulated world (map coordinates), adds a random noise and counts an equirectangular projection[65] from set starting position. This estimation should be accurate enough for short distance usage (like simulations for indoor robot) outside polar regions. The tool publishes its results as NavSatFix messages that we receive in our extension of AMCL node. We use inverse calculation of equirectangular projection to get map coordinates from the measurement.

The only sensor that comes with TurtleBot by default is the Kinect. For simulation, the Openni Kinect[66] plugin from gazebo_plugins[67] set is used. It behaves as Xbox Kinect by publishing the same types of messages. In our case, the depth image messages are the most important type, since these are translated to laser scans by the depthimage_to_laserscan convertor. Depth image provide tremendous amount of information, but such load of data would be too expensive to process. The convertor takes depth values in a predefined height and presents them as results of

---

[64] Documentation of team Hector plugins, "hector_gazebo_plugins – ROS Wiki" [online], http://wiki.ros.org/hector_gazebo_plugins#GazeboRosSonar, July 2018.

[65] Equirectangular projection, "Equirectangular projection – Wikipedia" [online] https://en.wikipedia.org/wiki/Equirectangular_projection, July 2018.

[66] Gazebo plugins description, "Gazebo : Turial : Gazebo plugins" [online], http://gazebosim.org/tutorials?tut=ros_gzplugins#OpenniKinect, July 2018.

[67] Documentation of Gazebo plugins, "gazebo_plugins – ROS Wiki" [online], http://wiki.ros.org/gazebo_plugins, July 2018.

laser scans. LaserScan is expected by the original part of AMCL and it plays a crucial part in the navigation node.

Joined data from wheels of the robot are also sent to the AMCL node as an odometry feed. It is used during the motion update phase of localization algorithm to shift particles in their direction by the measured length provided by odometry message. This data could be augmented with GPS feed by the robot_localization node, as we discussed in the previous chapter.

4.3.2. AMCL

The AMCL node manages the particle filter with all position samples. It listens to all sensory topics and updates the particles according to difference between received and calculated values. Our extended version of AMCL can receive LaserScan, Range and NavSatFix messages. Each of these message types requires a specific treatment which is provided by separate handlers. Names of topics with these message types are set in the configuration. The AMCL node thus functions as some funnel for all inputs – everything that robot could see (or what Gazebo provides) is channeled there.

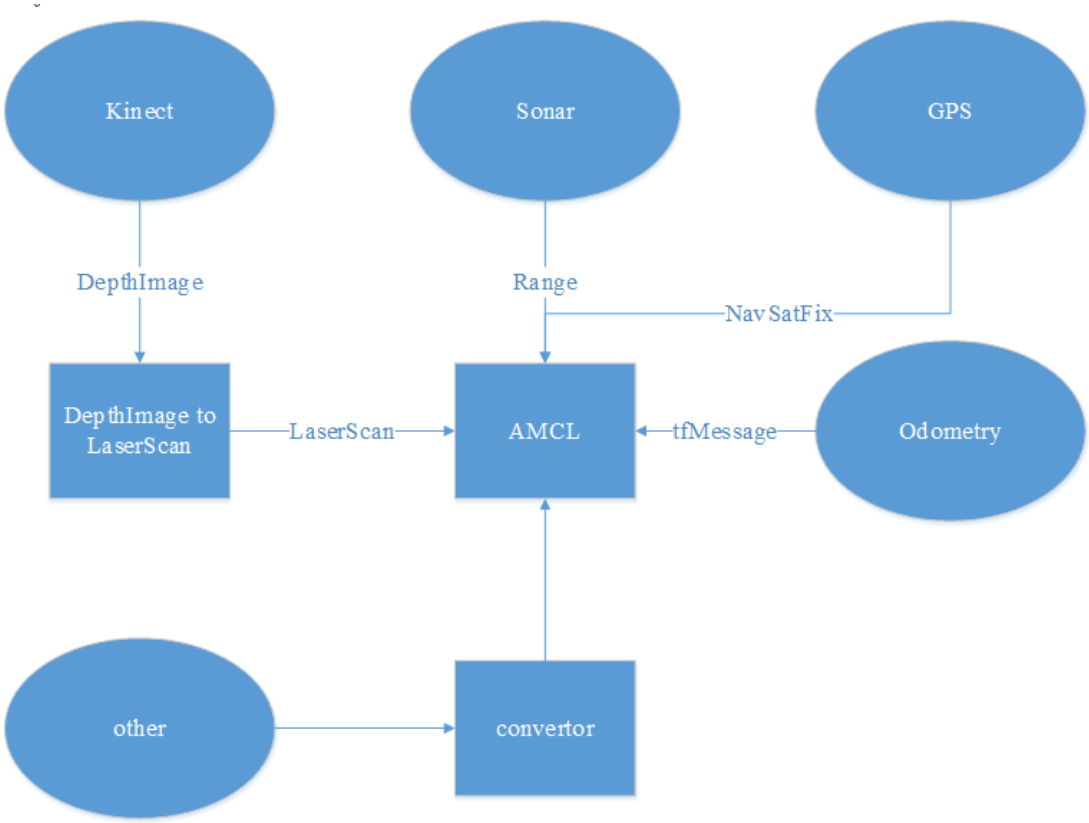

Figure 4.1 – Sensor inputs for particle filter updates inside the AMCL node.

4.3.3. Move base

The move base node is the source of all movement commands. Its complex of modules creates plans, evaluate progress of the robot and sends commands to the underlying structure. Those commands can be sent directly to the nodes of the wheels, or it can be passed to another tool for joining with other possible sources. We used a cmd_vel_mux[68] package that can fuse multiple movement commands sources with different priority and frequency. This tool then passes ordered commands to the wheels, or in our case to the Gazebo simulator, that listens to wheels commands from all sources and simulates movement of the robot in the world.

The move base allows to accommodate a plugin of both global and local planner. We of course configured our custom MHT global planner, that listens to

[68] Documentation of multiplexer for velocity commands, "cmd_vel_mux – ROS Wiki" [online], http://wiki.ros.org/cmd_vel_mux, July 2018.

position theories from our extended AMCL node. The AMCL feed contains positions of currently tracked hypothesis (clusters) inside its particle filter. The last message is stored in our planner and provided to MHT solvers. The planner then uses this stored information instead of the estimated position sent with the target position in the request for plan from the move base node.

The MHT planner in turn provides slots for other tools as we described in the previous subchapter. We used the default Global Planner as a supporting planner for easy situations with single theories and as a fallback plan. Therefore, even in the case all planning went wrong we still have the same chance create a good global plan as without MHT extensions. As a solver we recommend the Weighted Voting Planner that is best suited for deciding dilemmas with multiple theories. For occasions with too many theories we set the Rotation and Spiral Starter Planners. The starter planners can be configured repeatedly after each other to give time to the AMCL node to create theories from chaos of independent particles (abundance of samples occurs with the wake-up and kidnap problems).

Finally, the created global plan needs to be executed – the move base uses a local planner for implementing global plans inside the known close region and seen obstacles. We used the Dynamic Window Approach planner that takes a global plan and a local costmap and publishes a local plan. This plan is received by the move base and step by step passed to the underlying structure as described earlier.

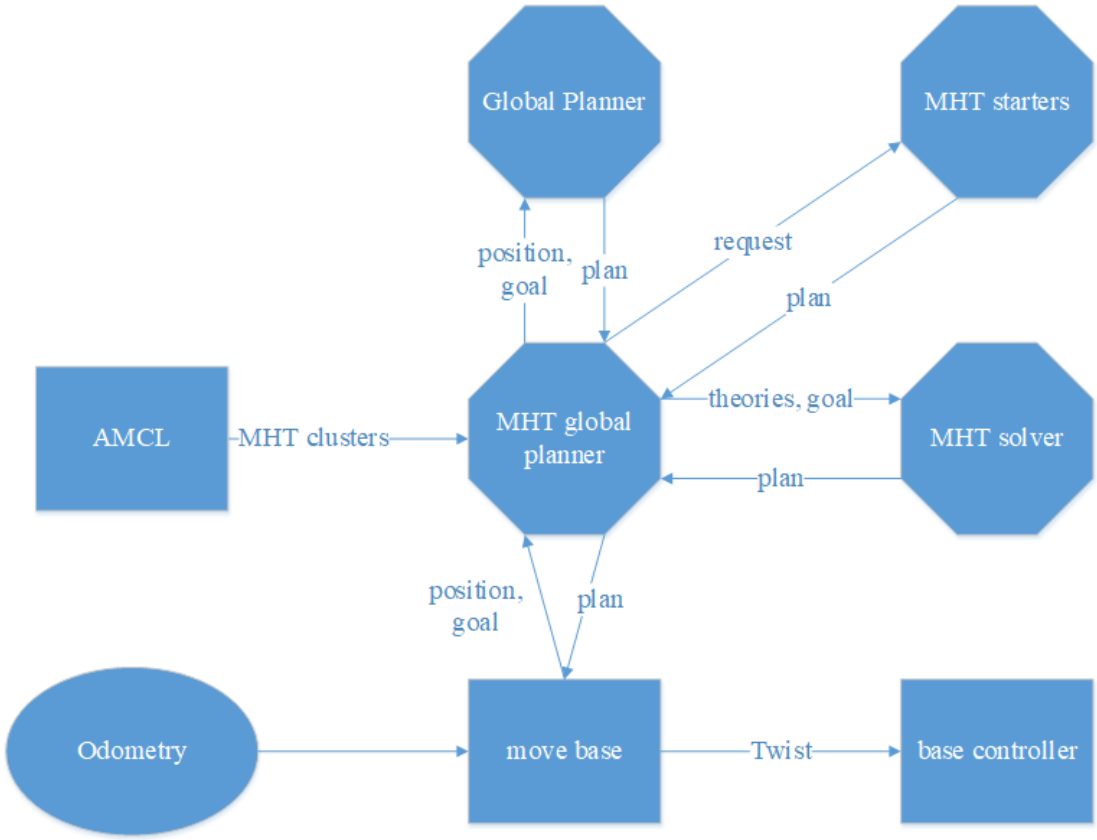

Figure 4.2 – Global planning modules for the move base.

4.3.4. Map

There also needs to be a map server running – both for user's visualization and for localization process of the robot. The published map is either from a custom-made image or from mapping (or rather SLAM) done by robot itself (during some previous run). The mapping tool itself is not needed after the map is created.

4.3.5. Visualization

Most of the messages can be visualized in the RVIZ tool. It shows various types of messages from the ROS environment and allows different visual interpretations of received data. We prepared a configuration file for RVIZ that can be launched from global launch file with all other components required for simulation of behavior of the robot. Some views are disabled by default to prevent

clogging of the shown environment with unnecessary layers of supplementary information. However, those feeds are usually important for the planning process. Thus, we let them stay included in the list of viewable topics.

We can also monitor what is happening in the “real” world through a Gazebo client. This tool shows a view of an observer (or a camera) in the simulated world regardless of what our robot can perceive. This view is useful for comparison with what the robot thinks and for its debugging.

5. Experiment

We tested our solution on the TurtleBot robot[69] in the Gazebo simulator, to avoid possible hardware based issues. We used two simulated worlds – a simple one, that is distributed with Gazebo's package (the "playground" world), and our own world, that resembles an open space office (the "office" world). We monitored progress of the robot in the RVIZ tool[70] and checked the result in Gazebo client. The office consists only of walls and tables, but provides a fitting example of ambiguous environment.

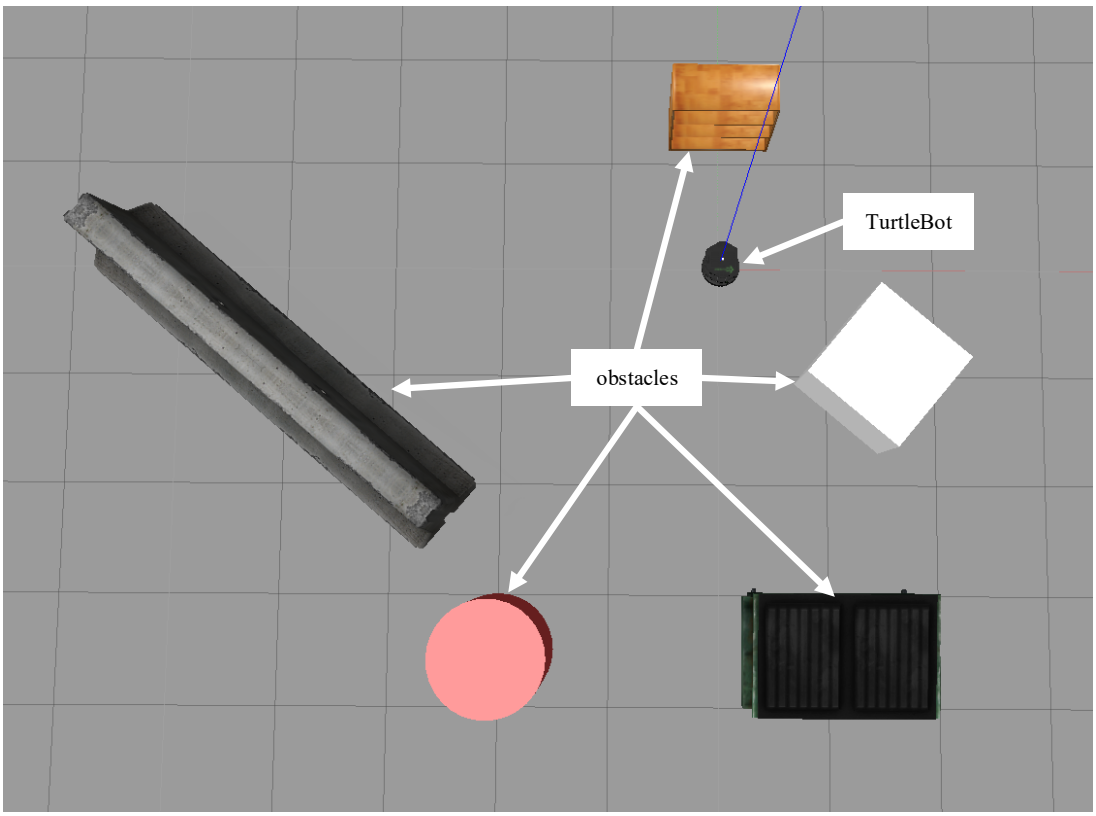

Figure 5.1 – View of the "playground" world in the Gazebo client, robot starts in the world [0,0,0] position heading east.

[69] TurtleBot v2, more online on https://www.turtlebot.com/turtlebot2/, July 2018.

[70] The 3D visualization tool for ROS, "rviz – ROS Wiki" [online], http://wiki.ros.org/rviz, July 2018.

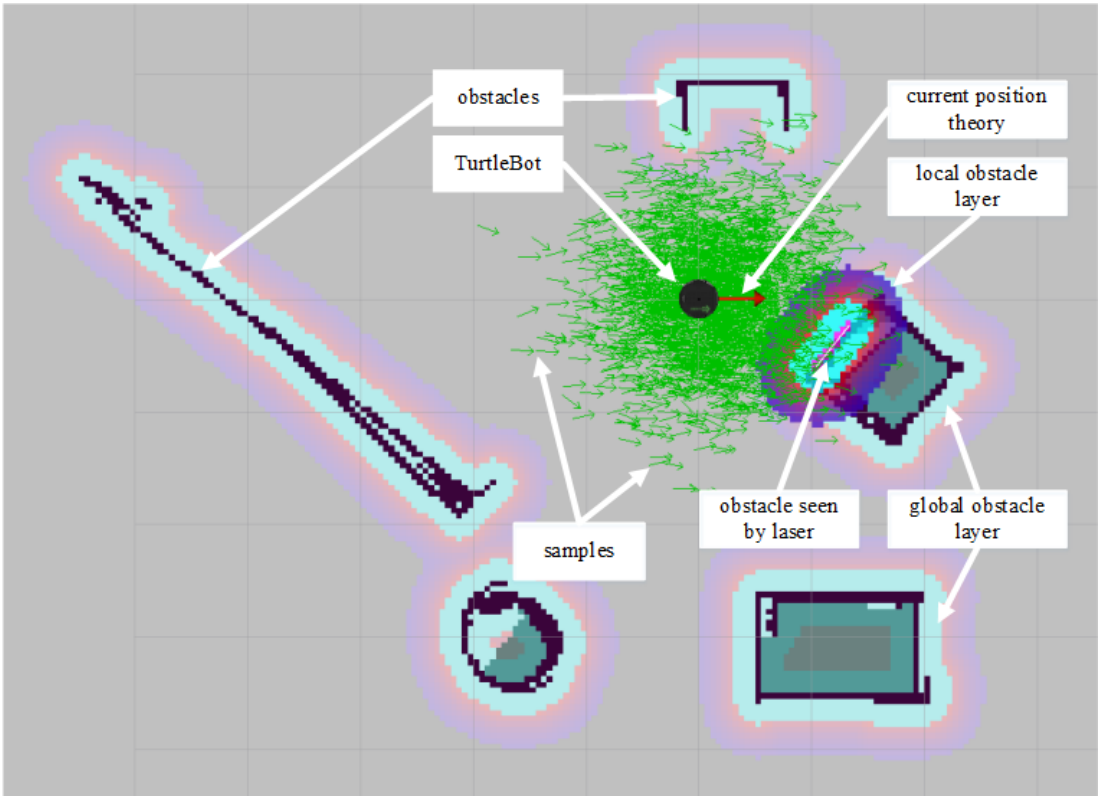

Figure 5.2 – View of the "playground" world in the RVIZ tool. The small green arrows represent the current state of the particle filter, while the large red arrow is (now) the only position theory.

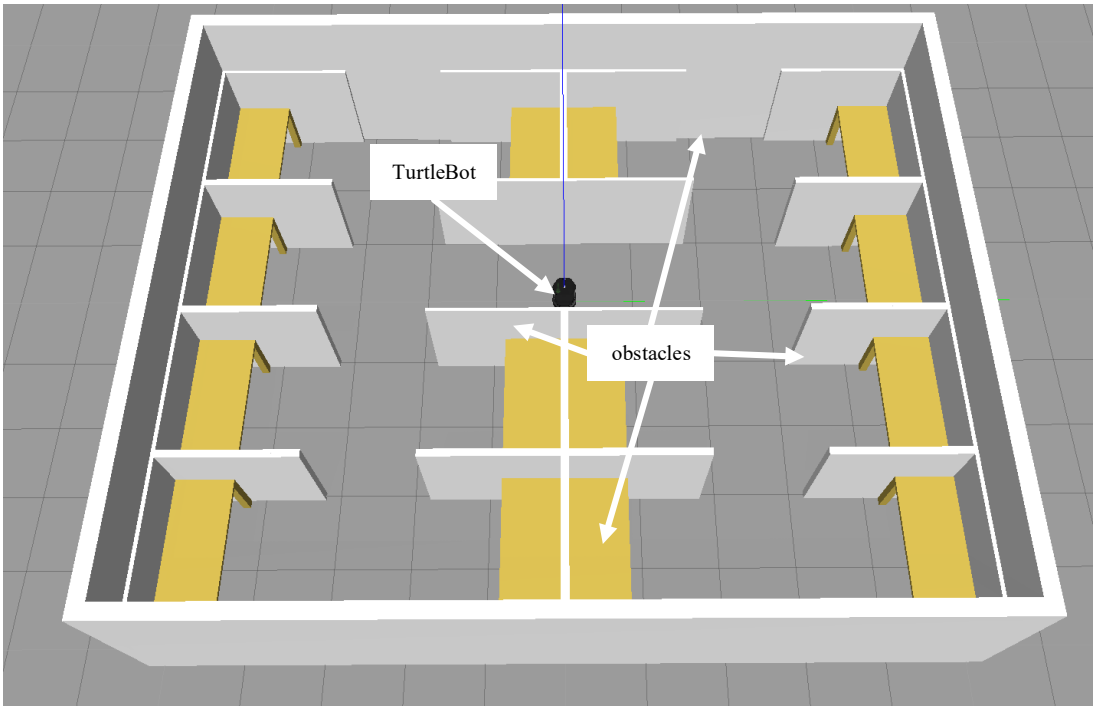

Figure 5.3 – View of the "office" world in the Gazebo client, robot starts in the world [0,0,0] position heading south.

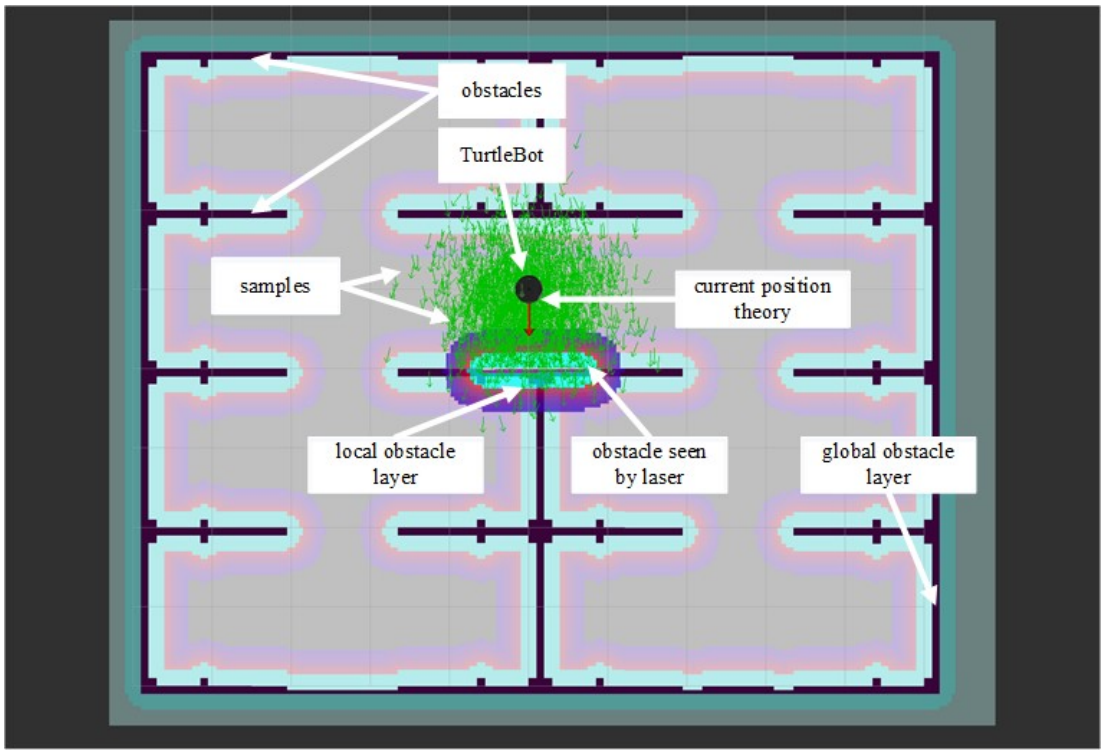

Figure 5.4 – View of the "office" world in the RVIZ tool.

The RVIZ tool provides insight into the ROS environment, we can see everything the robot knows (that can be displayed). With that, we can monitor development of the localization process – mainly the state of particle filter, believed state of the robot, known obstacles and a list of position theories (this applies only for our extended AMCL module that publishes them).

We prepared our tests by spawning robot in the world in the initial position as shown in the figures above, moving it to a chosen location using RVIZ or the teleop tool[71] by keyboard. Then we made the robot forget his whereabouts by calling the service for initiating global localization (by default set to "/global_localization"[72]). The particle filter is then reset and all samples are distributed randomly on vacant locations on the map. We also cleared all remaining (fake) obstacles from the

[71] Teleoperation tool for TurtleBot, "turtlebot_teleop – ROS Wiki" [online], http://wiki.ros.org/turtlebot_teleop, July 2018.

[72] AMCL services, "amcl – ROS Wiki" [online], http://wiki.ros.org/amcl?distro=kinetic#Services, July 2018.

costmaps when needed (service "/move_base/clear_costmaps"[73]), they sometimes remained from movement during the preparation phase. After that we sent a navigation request via RVIZ, which compels robot to localize itself and prepare a plan.

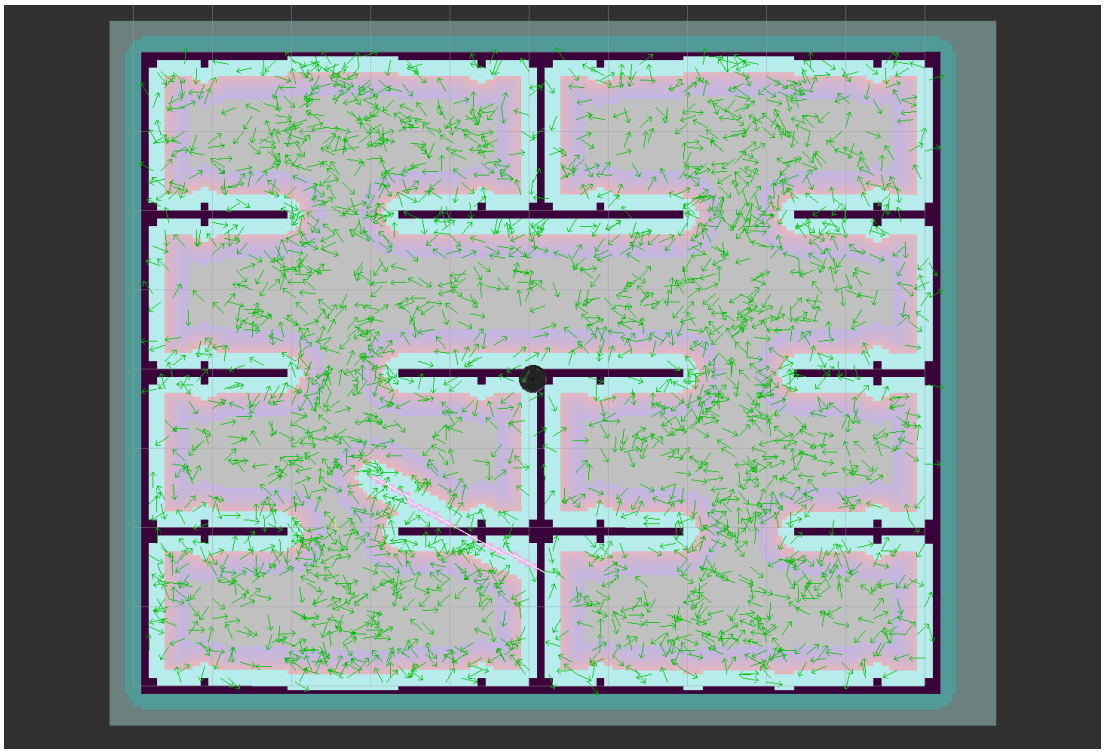

Figure 5.5 – State of the particle filter after localization reset. Position samples (small green arrows) are distributed randomly across the map. The extra light shadow outside black map lines is caused by obstacle currently seen by robot (thin white line). It is considered a dynamic obstacle and as such added to the costmap, relative to current estimated position of the robot. Unfortunately, this can pollute map with fake sighting, but this is how the navigation set works.

5.1. The playground world

First of all, we need to say that navigation in the playground world is truly simple – every type of object is present just once and almost every view on them is unique. Both our and the original navigation tools were able to localize the robot. We used this world to test out sensor fusion and navigation without laser.

[73] Move base services, "move_base – ROS Wiki" [online], http://wiki.ros.org/move_base?distro=kinetic#Services, July 2018.

With only a set of range sensors the localization took extremely long (hundreds to thousands of cycles compared to max. dozens of steps when only a laser is used), but the robot still managed to localize itself. This delay can be expected, given the difference between laser (scans for all obstacles in range) and sonar (finds the nearest obstacle in range).

The GPS sensor determined location of the robot immediately, orientation was amended during following steps as robot moved. This was also anticipated, since the only noise in readings is the noise we set in the simulator.

Combination of laser and range sensors proved to be demanding on computation power (shown by lower frame rate of simulation). Results, when measured in simulation steps, were equal to the setting with only a laser scanner.

5.2. The office world

We prepared the office world to test navigation in an ambiguous environment. Even though it is a small room with only 16 cubicles, the current navigation package managed to localize robot on only 7 occasions of 50. We launched the tests from every location three times, with no visible pattern in results. We understand that given the stochastic nature of used localization algorithm (based on particle filter) the result may differ with every subsequent test. However, we regard these results an opportunity for massive improvement.

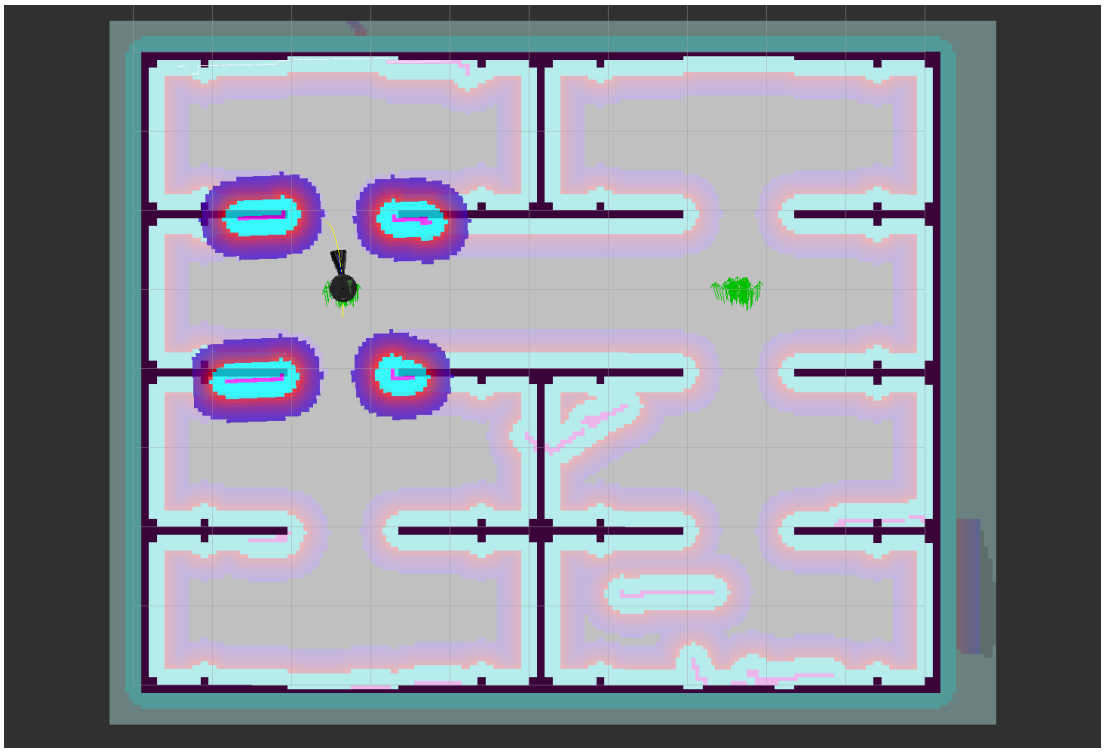

Figure 5.6 – Robot navigation failed, when it ended on location similar to the goal.

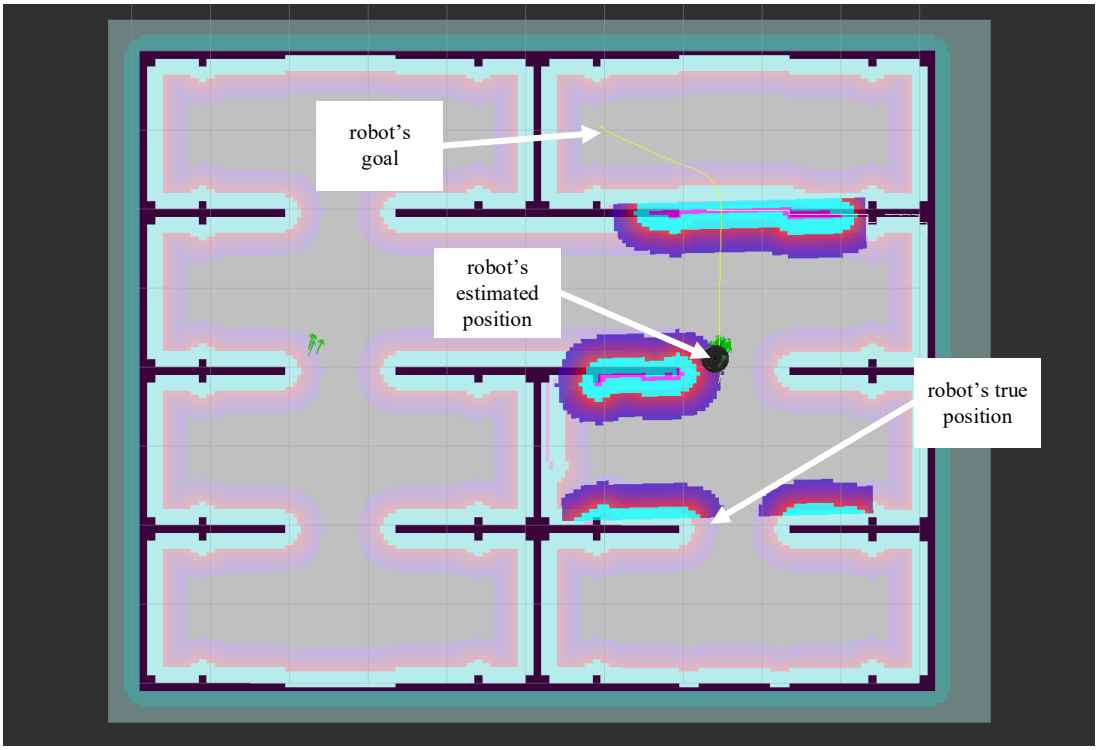

Figure 5.7 – Robot navigation failed, when it found its only path blocked. In reality, the robot was located one cubicle lower, facing the south wall.

When we launched our modifications on the same environment, we got 15 successful global localizations from 50 attempts. While this is a significant improvement compared to the current solution, there is still a potential for even better results. We detected, on several occasions, that a correct location was found, but it consisted of too few samples and did not survive first few elimination cycles. The main drawback was in the randomness of resampling. When we increased the maximal number of samples to 50 000 and minimal to 1 000, our results improved to 75% of positive navigation tests (15 out of 20 attempts). However, such amounts are computationally extremely demanding – our simulation rate dropped to 0.02 (simulated time per real time). Real robots would struggle to run such calculations in real time.

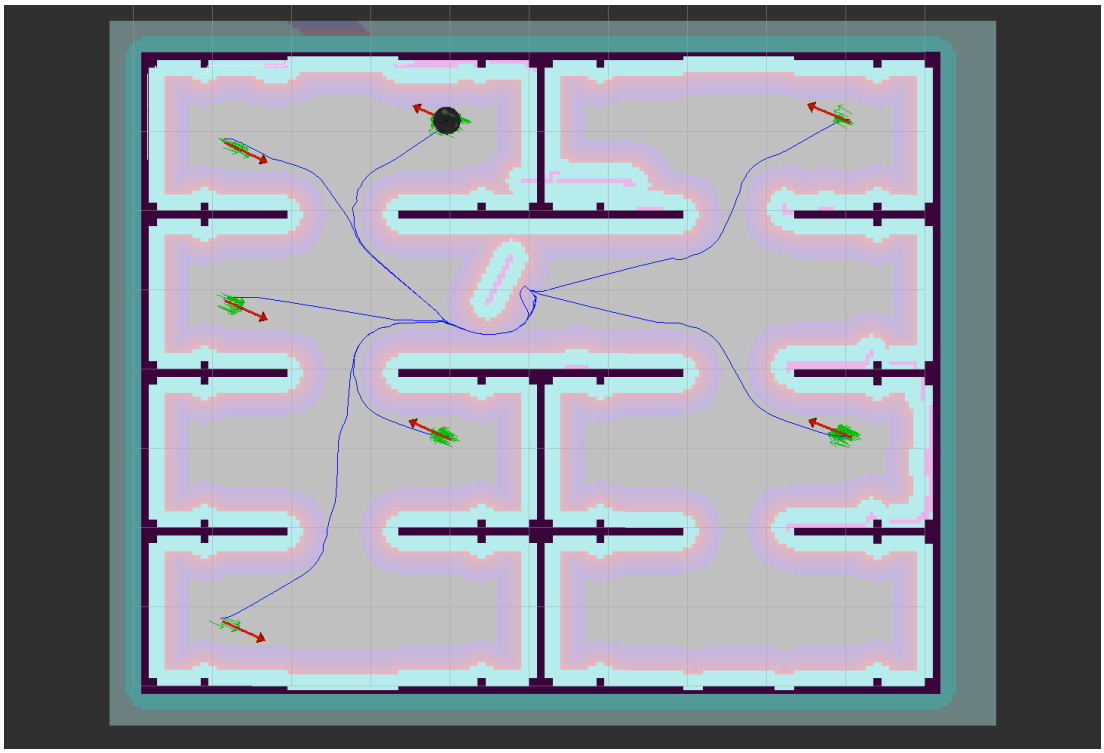

Figure 5.8 – View of all theories and plans the robot maintains. There are 7 theories about possible locations (big red arrows) with 401 position samples (small green arrows). Our tool prepares a path for each theory (blue paths) and performs "voting" among them based on their probability. Currently favored theory is false, robot is actually in the top right corner. This mistake was resolved soon after robot left the cubicle.

Conclusion

We extended the standard navigation package with a sensor fusion and an ability to track and evaluate multiple position hypotheses. We implemented components that integrate the Range and the NavSat messages to the AMCL mechanism and tested them in a simulated world. Robot equipped with only sonar managed to localize itself successfully, although it took much longer compared to robot with a laser scanner. Robot with satellite sensor localized itself almost immediately. Both results were expected given the nature of sensors used. We described integration of existing tools to fuse other sensor types into our solution.

Our planning modules that use multiple hypothesis tracking were equal to the current Global Planner when tested in a simple world. Moreover, our solution outperformed the current set when tried in an ambiguous environment and thus proved usefulness of the MHT concept.

Future work

Our extension of the standard navigation package is capable to navigate the robot even in ambiguous environment. However, throughout the work we encountered several areas where improvement is possible. We would start with reimplementation of the k-d tree preparation process, so that it would better represent the underlying particle filter. Next, we would follow with the move base control states, where we would add position for the starter planners. Then they would not interfere with the recovery mechanism and would not cause so many fake obstacles to appear in the costmap when the position of robot is being estimated.

List of Figures

List of Abbreviations

AMCL – adaptive Monte Carlo localization

EKF – extended Kalman filter

GPS – the Global Positioning System

GUI – graphical user interface

IMU – inertial measurement unit

KLD sampling – Kullback-Leibner divergence sampling

MHT – the Multiple Hypothesis Tracking

PCD format – the Point Cloud Data file format

RGB – red, green, blue

RGBD device – a device with RGB and depth output

ROS – the Robot Operating System

RVIZ – ROS visualization

SLAM – simultaneous localization and mapping

QR code – quick response code

WiFi – wireless fidelity

Attachments

Attachment 1 – Source codes and configuration files

Our source codes are stored in the folder “Workspace”, which also works as a Catkin workspace. There, in the folder “src”, every project or other component of our solution is in its own subfolder. Configuration and launch files are located there as well.

In the folder “gazebo” there are configuration files for the Gazebo simulator. We used some items from the Gazebo database and prepared some new objects for the experiment.