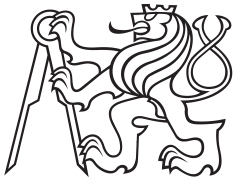
Master Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

An Integrated Approach to Multi-Robot Exploration of an Unknown Space

Bc. Vojtěch Lhotský

Supervisor: RNDr. Miroslav Kulich, Ph.D.
Field of study: Cybernetics and Robotics
Subfield: Robotics
May 2016

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

DIPLOMA THESIS ASSIGNMENT

Student: Bc. Vojtěch L h o t s k ý

Study programme: Cybernetics and Robotics

Specialisation: Robotics

Title of Diploma Thesis: An Integrated Approach to Multi-Robot Exploration of an Unknown Space

Guidelines:

1. Get acquainted with current approaches to multi-robot exploration, especially with [1,2].
2. Get acquainted with methods of simultaneous localization and mapping for RGBD data (RGBD SLAM), e.g. [3-5].
3. Choose an appropriate open source implementation of RGBD SLAM and experimentally verify its behavior with real data. Focus on computational complexity, precision of computed position in various environments.
4. Propose integration of RGBD SLAM into implementation of [1].
5. Verify experimentally the proposed solution and describe and discuss obtained results.

Bibliography/Sources:

[1] Tomáš Juchelka: Exploration algorithms in a polygonal domain, Diploma Thesis, CTU in Prague, FEE, Dept. of Cybernetics, 2013.
[2] Miroslav Kulich, Tomáš Juchelka,Libor Přeučil: Comparison of exploration strategies for multi-robot search, Acta Polytechnica , Vol. 55, No. 3 (2015), pp. 162-168.
[3] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, W. Burgard: An Evaluation of the RGB-D SLAM System, Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA), 2012.
[4] F. Endres, J. Hess, J. Sturm, D. Cremers, W. Burgard: 3D Mapping with an RGB-D Camera, IEEE Transactions on Robotics, 2014.
[5] M. Labbé and F. Michaud: Online Global Loop Closure Detection for Large-Scale Multi-Session Graph-Based SLAM, Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, 2014.

Diploma Thesis Supervisor: RNDr. Miroslav Kulich, Ph.D.

Valid until: the end of the summer semester of academic year 2016/2017

L.S.

prof. Dr. Ing. Jan Kybic
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, December 22, 2015

Acknowledgements

I would like to thank my supervisor RNDr. Miroslav Kulich, Ph.D. for his advice and guidance throughout this project and also to other members of the Intelligent and Mobile Robotics Group for the help mainly with hardware aspects of this work.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, date .....

.....

signature

Abstract

This work focuses mainly on the practical aspects of the multi-robot exploration of an unknown space. Each robot is equipped with an RGB-D camera and builds a 3D model of its neighbourhood. The team of robots is controlled by a centralized exploration approach using a 2D polygonal map of the environment. This work follows and extends the Exploration algorithms in a polygonal domain thesis by T. Juchelka [1], focuses on improving T. Juchelka's framework and connecting that framework with tools and libraries for simultaneous localization and mapping using an RGBD camera. The approach is verified by many experiments with both single and multi-robot exploration in various environment.

Keywords: multi-robot exploration, mapping, SLAM, RGB-D camera, polygonal domain

Supervisor: RNDr. Miroslav Kulich, Ph.D.

Abstrakt

Tato práce se zaměřuje hlavně na praktickou stránku multi-robotické explorace neznámého prostředí. Každý robot je vybaven RGB-D kamerou a staví si svůj 3D model okolí. Celý tým je pak řízen centralizovaným přístupem za použití 2D polygonální mapy. Práce navazuje na Explorační algoritmy v polygonální doméně od T. Juchelky[1], soustředí se na vylepšování původního frameworku a na propojení tohoto frameworku s nástroji a knihovnami pro souběžnou lokalizaci a mapování za pomocí RGB-D kamery. To celé je ověřeno pomocí mnoha experimentů s jedním i více roboty v různých prostředích.

Klíčová slova: multi-robotická explorace, mapování, SLAM, RGB-D kamera, polygonální doména

Překlad názvu: Integrovaný přístup k prohledávání neznámého prostředí týmem robotů

Contents

Figures

Tables

Chapter 1

Introduction

This work focuses mainly on practical aspects of autonomous multi-robot exploration of an unknown environment. Frameworks for multi-robot exploration are currently (2016) available, for example Explorer package [2] for ROS (Robot Operating System [3]), but they mostly work only in 2D environment on a grid map using a laser rangefinder.

A framework working with a 2D polygonal map is available, but it was never tested on a real robot. That framework is a result of the "Exploration algorithms in a polygonal domain" thesis by Tomáš Juchelka [1]. Polygonal maps have several advantages over the grid maps. They are much more memory efficient and precise especially in large environments. On the other hand some operations are much more complicated in a polygonal domain, for example incorporation of new scans into the map. The EAPD (Exploration in a polygonal domain) framework is written in C++ using ROS and provides a way for autonomous exploration by multiple robots equipped with laser rangefinders, but it does not provide localization.

The main idea behind this thesis is to follow and extend the work by T. Juchelka, especially to make it work in real environments, but also to add a creation of 3D map using an RGBD camera. Most libraries necessary for an autonomous multi-robot 3D exploration with an RGBD camera are already implemented in the Robot Operating System (Sec. 3.1):

- Drivers for RGBD cameras
- Simultaneous localization and mapping (SLAM) using data from RGBD cameras
- Libraries providing simple communication, visualization, data processing and transformations

The main goal in this case is to use those libraries and frameworks, add missing features, create a complete framework for a multi-robot exploration, which produces a 3D map and perform experiments in the real world environments in order to determine and solve the issues with practical deployment.

One question arises. Why to use 2D polygonal map when a 3D map is also created? The main reason is that a continuous transfer of 3D data is very demanding on a wireless network. Using a 2D polygonal map for exploration algorithm while each robot builds its own 3D map (which can be merged after the exploration finishes) creates much lower load on the network. In order to implement such solution it is necessary to solve many issues that do not appear in the simulator and provide new features and extensions in several directions

1. Usage of an RGBD camera sensor (specifically Microsoft Kinect 2), processing and filtering its data.
2. Creation of additional 3D map of the environment.
3. Using a 3D SLAM approach for creating the map and for determining the position of the robots.
4. Solving many problems with the real environment (noise, communication, inaccuracies of sensors and drives, computational power requirements, etc.).

As the original work was done only in the simulator, the first goal is to consider requirements and to determine which hardware and software libraries to use. The computational requirements are one of the biggest problems as the current software providing 3D SLAM is very demanding.

The next step is to connect all the parts of the software with the "Exploration in a polygonal domain" package (later referred as "EAPD"), enhance and optimize that package and make it more user friendly by providing many parameters, which can be easily modified. That is followed by doing many experiments in a real environment to detect and solve possible problems and to finalize an exploration framework. The comparison of different approaches for planning and distributing the goals of the robots is not the part of this work as it is already well covered in [1].

Current approaches to multi-robot exploration are introduced in the beginning of this work (Chapter 2) with a focus on the one developed by T. Juchelka. The following chapter focuses on the implementation (Chapter 3). The framework (Sec. 3.1) and the original implementation of EAPD (3.2) are briefly covered followed by the description of the modifications made as a part of this work (3.3). The next section (3.4) explains the architecture of the whole application, especially the connections between the exploration itself and other parts, which are later described in more detail - Laser simulator (Sec. 3.5), 3D SLAM (Sec. 3.6) and odometry (Sec. 3.7).

Chapter 4 describes the selected sensors and computer as well as the robot itself with the modifications made during this work. The following chapters focus on the experiments. Chapter 5 covers the experiments with the computational and connection requirements of the whole framework and

RGBD SLAM. Chapter 6 focuses on the testing of the Microsoft Kinect 2 sensor [4] and the following chapters cover the experiments with a single robot (Chapter 7) and finally experiments with a multi-robot exploration (Chapter 8).

Chapter 2

Current approaches to multi-robot exploration

This chapter aims to provide a basic view on the problem as good sources exist [1][5]. It also focuses on the approaches already implemented in the Robot Operating System framework [3].

2.1 Exploration using a 2D occupancy grid

The majority of nowadays approaches (for example Explorer package for ROS [2]) uses occupancy grid maps. Such maps are created by sampling the environment into grids consisting of small squares. Occupancy grid is a matrix where each cell has a value representing the probability that the corresponding area is occupied (see Fig. 2.1). The disadvantage is that occupancy grids can take a lot of memory and computational time in large environments.

The Explorer package is a frontier based exploration that incorporates both coordinated and uncoordinated exploration strategies selecting goals for robots.

2.2 Exploration in a polygonal domain

The polygonal map (Fig. 2.2) is much more memory efficient environment representation and can retain its precision even for large environments where an occupancy grid must use bigger cell size.

Several approaches for the exploration in a polygonal domain exist, for example "Exploration and Mapping of Unknown Polygonal Environments Based on Uncertain Range Data" [6], but this work focuses mainly on the approach by T. Juchelka [1].

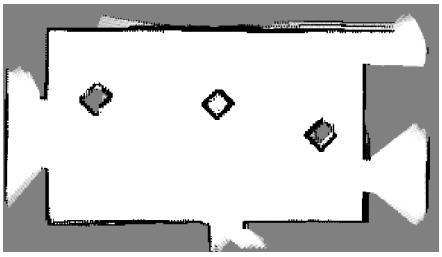


Figure 2.1: An example of an occupancy grid

The original framework deals with the problem of exploring an unknown environment with a group of robots. They are equipped with a ranging sensor (laser rangefinder or similar) and cooperate in order to make the exploration fast and efficient.

Usage of polygonal map also brings several problems. Some of the libraries used for exploration do not work on polygons. Updating the map is more complicated (merging and clipping polygons or obstacle expansion).

The exploration procedure consists of following steps:

1. Update the position of the robots from the odometry.
2. Convert the data from laser rangefinders to polygons.
3. Incorporate the new polygons into the polygonal map using the clipping library.
4. Create an offset map (alternative to the obstacle expansion in occupancy grid).
5. Select the target frontiers.
6. Create a visibility graph.
7. Generate the plans for robots using a specified method (for example Yamauchi [7])
8. Drive the robots using the Smooth Nearness Diagram algorithm

This exploration framework provides good results in a simulated environment, but it works with several assumptions that are not met in the real world:

- The positions of all robots are known and precise.
- The sensors are close to ideal (no large errors).
- The environment is static.
- The floor is ideally flat.

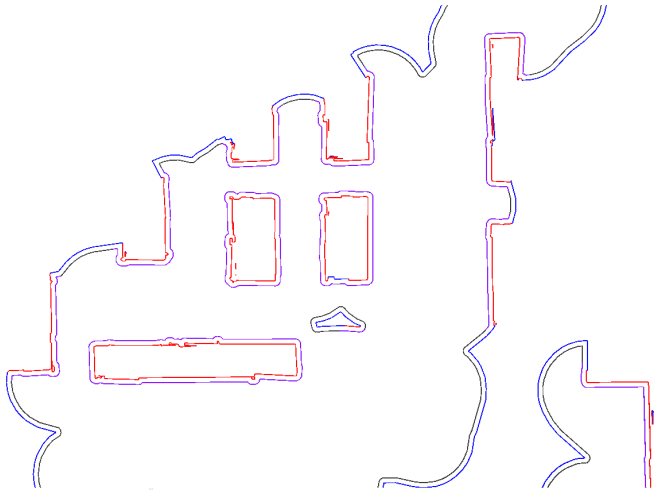


Figure 2.2: An example of a polygonal map generated by EAPD. The red colour represents obstacles, the blue one frontier, the violet one expanded obstacles and the grey one expanded frontier

Removing those assumptions either by extending the framework or by incorporating other libraries is one of the main goals of this work.

Chapter 3

Implementation

This chapter focuses on the implementation of the exploration framework. The first part of the chapter explains the general approach and modifications made in the original software. The second part covers the architecture of the whole framework used with an RGBD camera and other specific parts of the framework like laser simulator, SLAM or odometry.

3.1 Framework (Robot Operating System)

The Robot Operating System (ROS) [8] is a large framework which contains many libraries, tools and drivers useful for creating robotic applications.

The advantage of the ROS is that it provides a simple way for applications to communicate with each other even when they are on different machines. Each process running under the ROS framework is interpreted as a graph node. The nodes can interchange informations by sending messages to a specified topic or by subscribing to a topic in order to receive messages. Nodes can also provide services for other nodes.

Another advantage is the ROS way of handling different coordinate frames. The frames are also represented as nodes in a graph. Transformations between frames (edges in the graph) are published using the topic system and the current transformation between two frames can be simply obtained by using the ROS TF library (there must be a path in the graph between them). An example of a transformation tree is presented in Fig. 3.1. The base frame is the map frame which is connected to the frames of the robots. This example is from EAPD running in a simulator with 2 robots.

The transformations are also very useful for handling positioning of sensors on the robot or even when the robot has movable joints. A separate transformation between the frame of the robot and a sensor (or a joint) can be published so that the message from the sensor can be easily transformed to any other frame. An example of a transformation to the coordinate frame of a sensor can be seen in Fig. 3.1 - from `robot_0/base_link` to

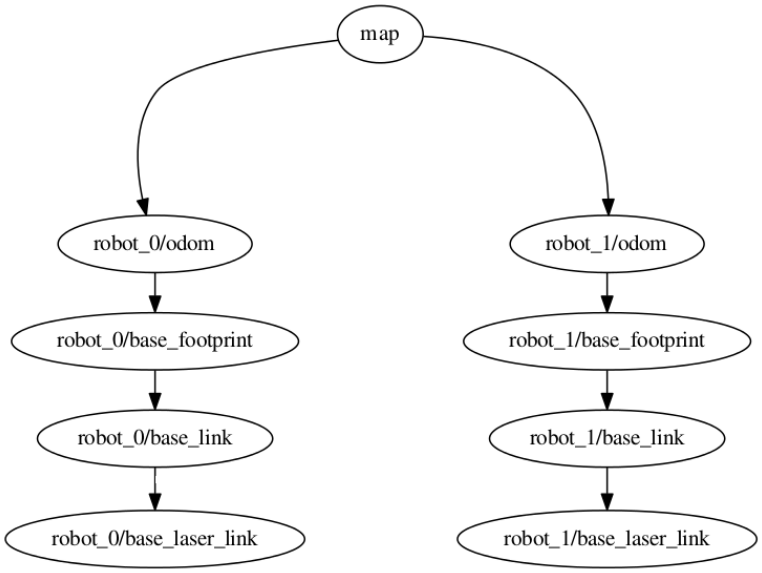


Figure 3.1: An example of a transformation tree in ROS. This is a transformation tree from the exploration in a polygonal domain running in a simulator.

`robot_0/base_laser_link`.

The ROS is currently a popular framework so the drivers for a wide range of different hardware (robots, sensors, ...) are available. The framework contains also a wide range of different tools, for example:

Rviz An application used for 3D data visualization.

RQT A framework for GUI development in ROS. It contains many different tools for data processing, monitoring, debugging, etc.

Catkin make A build system extending the standard CMake for easier building of ROS packages.

The version used in this work is Indigo running on Ubuntu 14.04 (The ROS can run on other operating systems as well, but Ubuntu has currently the best support).

3.2 EAPD implementation

The EAPD (Exploration in a Polygonal Domain) package is a C++ implementation (using ROS libraries) of the multi-robot exploration algorithm by T. Juchelka (Sec. 2.2) [1], which has been improved and extended as a part of this work.

The whole package contains three ROS applications:

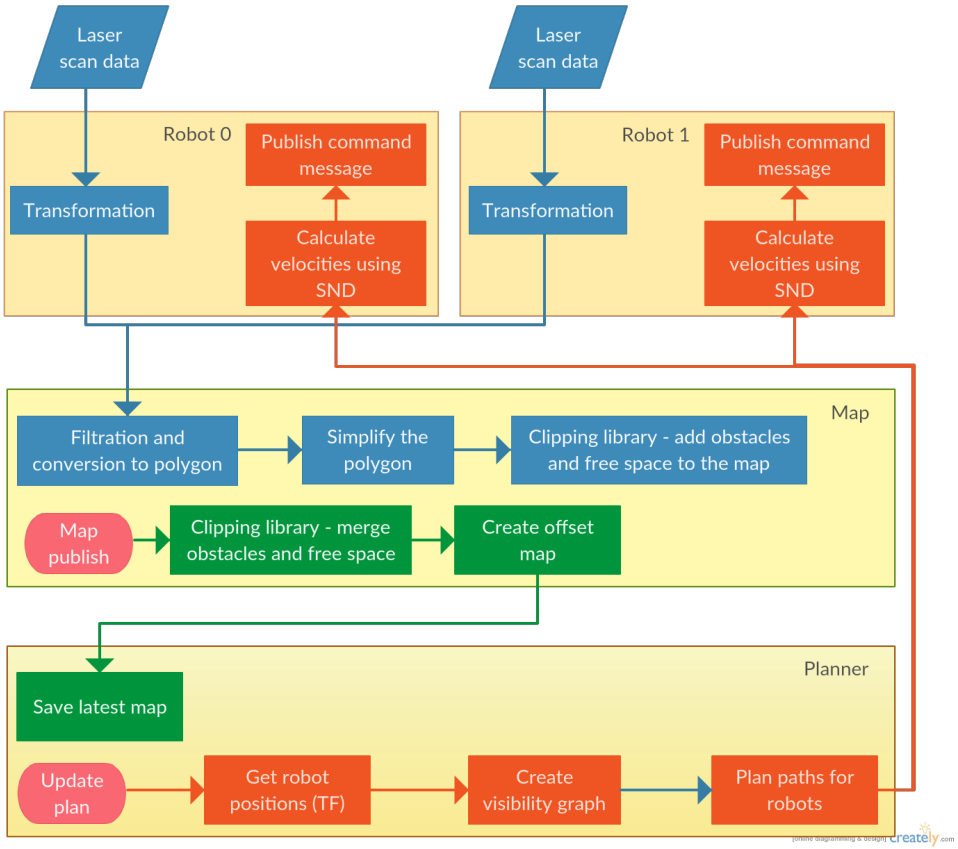


Figure 3.2: A diagram representing the implementation of EAPD.

Robot processes the data from sensors on one robot and publishes them to the map and the planner. It also drives the robot according to the path from the planner using SND (Smooth Nearness Diagram) approach [9].

Map uses the laser scans together with the transformations from the robots and creates a polygonal map of the environment. That map is published and optionally visualized in Rviz (Sec. 3.1).

Planner uses the map and the positions of the robots, selects the goals for the robots and plans their paths using the Dijkstra's algorithm [10].

In a single robot case, all three applications run on one machine - the computer typically placed on the robot itself. In a multi robot case, the map and the planner run on a central machine which collects the data from all robots and provides them with the planned paths (see Fig. 3.2).

The whole package uses a laser scan messages and transformations as an input and publishes the commands for the robots as well as the polygonal map. An interface for visualizing the data (map, plan, visibility graph) in Rviz visualization tool is also implemented.

3.3 Modifications in EAPD

The original exploration was tested only in the simulator and several problems appeared during the tests on a real robot. Some parts of the application are significantly modified and several new features are added.

3.3.1 Transformations

The original EAPD package used odometry topics as a transfer channel for the information about the positions. This was not desired in the real world application for several reasons:

- Odometry and position on the map is not the same.
- Transformations provide extrapolation [1].
- Positions were sent through some parameters in a laser scan message.

Due to these reasons the original approach is replaced by ROS transformations. A part of the code has already been rewritten for the usage of transformations [11] and the rest is completed during the work on this thesis.

The transformations do not have any of the issues listed above and they allow nicer and more readable source code. The structure of the transformation tree of a multi-robot task in the simulator has been presented in the previous chapter (Fig. 3.1). The real environment transformation tree (for one-robot case) is presented in Fig. 3.3. The main difference is the usage of RGBD camera instead of a laser.

Each coordinate frame has a different meaning. The map frame is a fixed coordinate frame that contains both polygonal and 3D map. Each robot in a multi-robot case has its own 3D map so the fixed map frame is used only for the polygonal map and the 3D maps have their own coordinate frames (`map_0`, `map_1`, ...). The transformations between these frames must be static, but they can be computed prior to the start of the exploration.

If the odometry would be ideal the `odom_combined` frame would be identical to the frame of the map. The odometry is not ideal in the real world so the SLAM must make corrections of the position by providing a transformation between the map and odometry frames.

The rest of the frames represents different positions on one robot. For example, if the sensor is not in the centre of the robot, providing a static transformation between the robot and the sensor frames resolves this problem and the point clouds are always automatically transformed into the correct coordinate frame.

[1] Missing extrapolation causes inaccuracies in the position because when some old laser scan message arrives, it is always matched with the latest odometry.

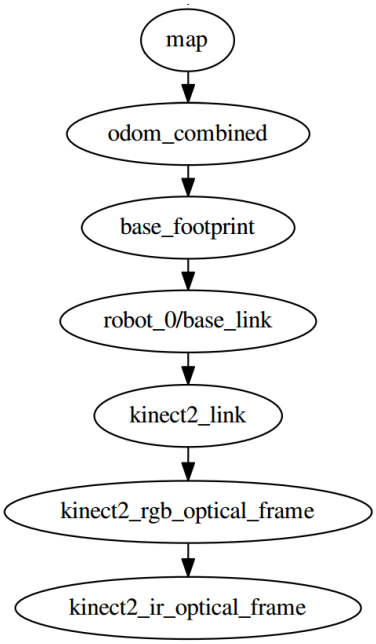


Figure 3.3: The transformation tree of one-robot EAPD running with RGBD camera

The following parts of EAPD are modified to use transformations:

Callback of a message from a laser scan The message from a laser scan was originally directly placed into the map at the position from the latest odometry message. Transformations allow using different coordinate frames for the laser and the robot as well as non-zero transformation between the odometry and map frames.

Robot position The position of the robot is now always obtained through the ROS transformations API. This change simplifies the source code because all callback functions for odometry are now replaced by a simple request for transformation and it allows to use extrapolation for determining the position in a specified point in time.

Laser scan transformation The laser scan is transformed from its own frame to the frame of the map. The advantage is that the scan does not need to be from robot's frame but it can have its own transformation frame.

3.3.2 Exploration boundaries

The environment in which the exploration takes place may be too large and the robot would try to map everything, which could be nearly impossible and may cause omission of some important places. Restricting the exploration can be beneficial in such cases.

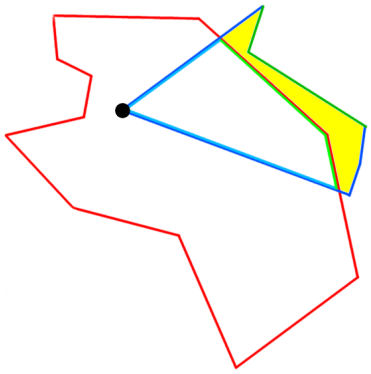


Figure 3.4: Filtering data outside of the bounding polygon (red). The input polygon is visualized in blue (free space) and green (obstacle). The new polygon in light blue and light green. The yellow area is outside of the bounding box therefore it is removed.

The restriction is implemented in such way that it allows to set a polygon which represents the boundaries for the exploration. This polygon behaves like a solid wall, the robots do not go outside of it. It is also represented in the polygonal map as a solid wall.

This behaviour is achieved by using a simple polygon filter which is applied on all polygons generated from arriving laser scans. All points outside the boundaries are removed and new points on the edge of a bounding polygon are added (see Fig. 3.4).

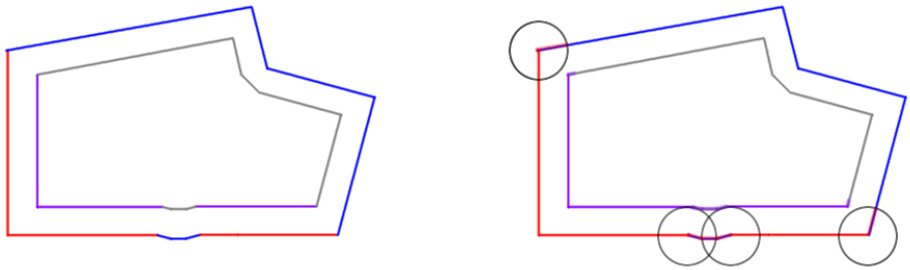
Changing the bounding polygon is possible on the fly by publishing a new one through ROS topic, but it would not be applied on already mapped areas until they are revisited by the robots.

3.3.3 Offset map

The original EAPD package solved the problem of expanding the obstacles by simply generating an offset polygon (using clipper library [12]). The problem is that this method keeps a frontier in front of narrow passages (see the bottom passage in Fig. 3.5a). The planner finds the path to that frontier and the robot goes there even though it can not get further through that narrow passage.

The result is an infinite loop where the robot moves to and from that frontier. It can not get through. When the robot reaches that frontier the planner sends it somewhere else, but it sends it back in the next planning iteration (as it is again the closest frontier).

The solution to this problem is to replace the frontiers close (closer than the distance used as an offset for expanding the map) to the obstacles by a wall (see Fig. 3.5b). This seals the problematic passages and the offset map can be generated in the same way as before.



(a) : The original offset where the impassable corridor is still marked as a frontier.

(b) : The offset after extending the obstacles. The impassable corridor is not marked as a frontier and the robot would not try to get there.

Figure 3.5: Offset map

3.3.4 Updating the map

Every measurement is converted to a polygon, split into obstacles and frontiers and merged with the map in the original work using the clipping library [12]. The obstacles and free space in the map are stored separately. When the map is published the obstacles are merged with the free space recreating the frontier.

This approach is simple, fast and creates a very nice polygonal map, but it works only with a static environment and an ideal (or close to ideal) sensor. Any object detected in already mapped space is simply discarded[2] when obstacles are merged with the free space.

Even if the environment is static, this may still be an issue. If the robot would for example go over a small bump and the sensor would be momentarily turned slightly upwards, it may "see" over a small obstacle and place the space behind it into the map. The obstacle would be removed and never placed back, because it would be in the area marked as a free space.

Another issue may arise when some glossy object is present. The sensor usually detects some false points due to the bad reflections and if those points are behind the actual object, it is never placed into the map.

This issue is solved by forcing all obstacles (from the "obstacles" part of the map) back into the clipped map. A new problem is that now the free space can never replace an obstacle. It is fixed by clearing all free space between the robot and the detected obstacles when a new scan is placed into the map.

This approach ensures that the newest data are always preferred. The published map looks less nice because obstacles are more fragmented, but the

[2]The obstacle remains stored in the "obstacles" part of the map, but it can never appear on the publish map due to the clipping.

exploration using that map is much more resilient.

3.3.5 Small fixes and improvements

The main changes are already covered in the sections above but during the work with the EAPD package many small problems appeared. A short description of these problems is provided bellow:

Number of active robots

Each robot in the original version of EAPD increased the counter in a ROS parameter named `robot_count`. This approach failed when the exploration was launched for the second time on the same ROS master. The parameter increased again so the final number of robots was doubled.

This is solved by determining the number of active robots directly in the planner node with a possibility to set up the `robot_count` parameter manually.

Visualization

The EAPD package contains a small library for visualizing the data in Rviz. That library originally used a fixed scale, which caused problems with viewing too small or too large maps. This is resolved by adding a `scale` parameter.

The part of the visualization concerning the visibility graph from the planner was rewritten (originally it failed to visualize data properly in more complex environments).

Parameters

All parameters are now obtained by using the ROS parameter server. That removes the necessity for recompiling the whole package each time some parameter is changed. It also allows creation of a well-arranged parameter file.

Laser range

Originally, this parameter had to be set to the same value as the range of the real sensor. Using smaller values would cause creation of a wall around the robot in the map and immediate crash of the planner because there was no reachable frontier.

The parameter can now be safely set to smaller values. The situation when there is no frontier can still happen (for example when the robot finishes the exploration), but the planner does not crash any more and it starts slow rotation of all robots. This is helpful in cases when for example someone opens door and the exploration can continue.

Start by rotation

A new option for starting the exploration by rotating the robots first is

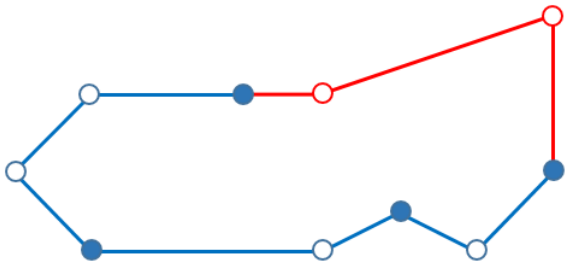


Figure 3.6: Selecting target frontiers. The part of the polygon representing the obstacles is visualized in red, frontier is blue. The selected points are filled.

added to the planner. It is better when the robot first turns to see what is around it and starts the exploration after that.

Path through known walls

It was possible that the robot would have been inside of an expanded obstacle, when the position had been updated after the map had been published. That caused planner to generate a path through the obstacles instead of the free space. This issue is solved by cutting the robot footprint from the map inside of the planner node.

Selecting target frontiers

A function for selecting target frontiers for the planning is in the planning part of EAPD. The input of this function is a set of polygons. Each edge of a polygon can represent either an obstacle or a frontier. The goal is to reduce the number of frontiers used in the planning algorithm. It is pointless to plan a path to several different frontiers that are very close to each other.

The frontiers are selected by a predefined distance. The goal is to have the maximal distance between the two selected frontiers lower than the predefined threshold while selecting as few frontiers as possible.

The original function contained some problematic parts which caused a risk of an infinite loop. This function is completely rewritten in the new version of EAPD. The original function selected targets by the radius but the new version uses the distance along the edges of the polygon. It is also guaranteed in the new version that the beginning and the end of the frontier section is always selected. The behaviour of such function is shown in Fig. 3.6. The pseudo code of the presented method is in Algorithm 1.

Build environment The Robot Operating System originally used Rosbuild [13] environment for the compilation of packages. The environment contained a set of scripts that incorporated ROS libraries into the make (and CMake) environment. This system is still supported, but it is slowly replaced by the new one called Catkin [14].

Catkin is also a build system extending the CMake environment and it is improved in many ways (for example better support of installing packages

Algorithm 1 Generate target frontiers.

```
1: for polygon P ∈ map M do
2:     if P[0] is a frontier and the polygon contains some obstacle then
3:         Set i_s to the beginning of some polygon segment.
4:     end if
5:     d = 0
6:     for point P[i] ∈ P starting with i_s do
7:         if P[i] is a frontier following or followed by the wall then
8:             Select P[i]
9:             d = 0
10:        else if P[i] is a frontier then
11:            d = d + |P[i] − P[i + 1]
12:            if d >= d_m then
13:                Select P[i]
14:                d = 0
15:            end if
16:        end if
17:    end for
18: end for
```

Note: If $i_s \neq 0$, the loop must continue from zero when the end is reached in order to process all points in the polygon.

on different operation system). Catkin also removes the Makefile from the project and uses only CMakeLists.txt.

The EAPD package was originally made in the old Rosbuild environment, but it has been migrated to the Catkin and it can be simply compiled, deployed or installed from the Catkin workspace now.

3.4 EAPD on a robot with RGBD camera

The EAPD package needs a data from a laser rangefinder and a precise position in order to work. The first step is to simulate the laser from RGBD data provided by the RGBD sensor. The next one is to use some SLAM approach for obtaining a precise position of the robot.

If the selected SLAM approach would allow to use the odometry as an initial guess for incorporating new scans into the map, providing a precise and reliable odometry may significantly improve the map quality.

3.4.1 Devices and drivers

The RGBD camera drivers are specific for each device, but they usually have the same structure. One part is the actual driver which works with the

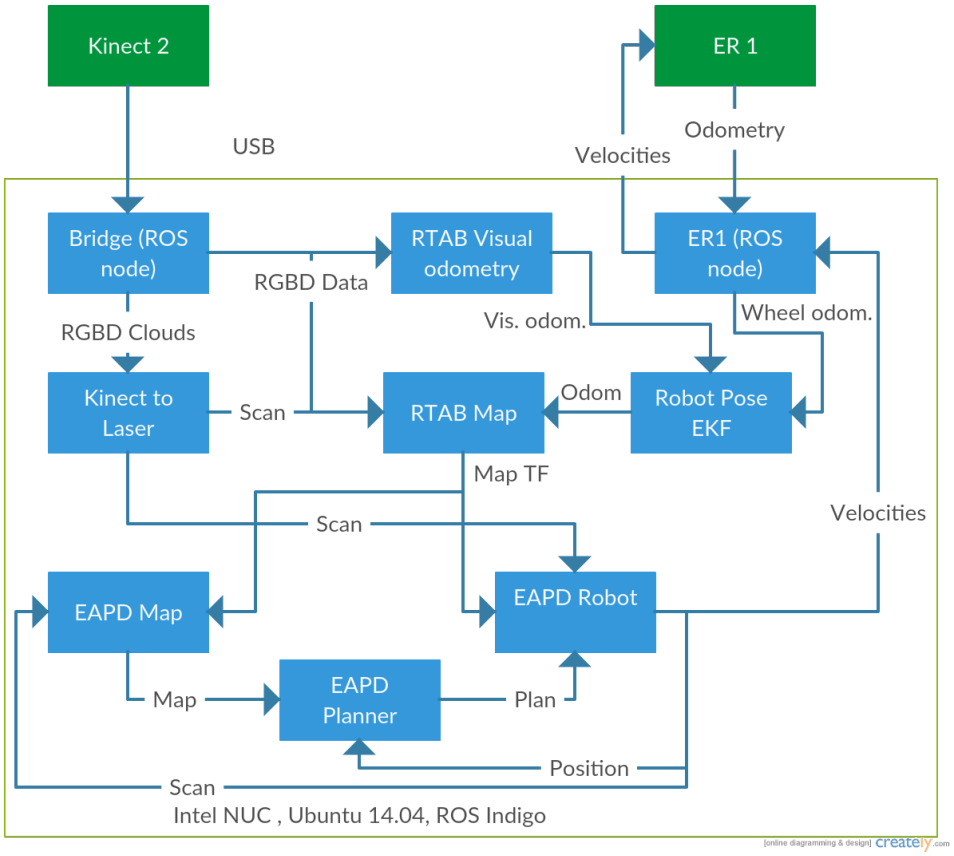


Figure 3.7: The connection between modules in a single robot case.

device and provides an API for other applications. The other part is the ROS node using that API for publishing the RGBD data via ROS topics.

For example when using the Microsoft Kinect 2 [4] the libfreenect2 [15] library provides basic drivers for connecting the sensor and the ROS Kinect 2 Bridge [16] publishes the data.

The implementation works with ground robots able to process the commands in the form of linear and angular velocities and ideally providing the wheel odometry as well. The data are processed in a ROS driver which provides an interface between the connected device and ROS.

An example of the robot driver is the ROS ER1 node (provided by the Intelligent and Mobile Robotics Group - CTU in Prague) for the ER1 [17] robot which is used for experiments in this thesis. The node accepts ROS twist messages containing linear and angular velocity and publishes the odometry measured from the wheels on ER1 robot.

3.4.2 Application architecture

In the single robot case all processes run on a single computer (typically placed directly on the robot). Both RGBD camera and ER1 robot are connected to the computer using USB interface. The ROS drivers for those components are separate nodes - bridge for the RGBD camera and the ER1 node.

The visual odometry is computed from the RGBD data and together with the wheel odometry is sent to the Extended Kalman Filter [18] for processing and generating more reliable source of odometry in a form of a transformation between `odom` and robot `base_link` frames. It is used afterwards by the SLAM (RTAB Map [19]) together with the RGBD data and the simulated laser scan to create a 3D map of the environment and publish the transformation between map and odometry frames.

The EAPD package uses the transformations and the simulated laser scan to create a polygonal map, which is expanded and used for planning the path and driving the robot along that path by publishing `cmd_vel` messages to the ER1 node.

The detailed scheme of the data transfer between all software modules is provided in Fig. 3.7.

The multi robot case is slightly more complicated. Each robot runs its own SLAM node, the laser simulator and also the `Robot` node from EAPD (see Sec. 3.2). The polygonal mapping and planning is processed on some central machine (or on one of the robots) which communicates with other robots using WiFi. This solution has very light load on the wireless communication because only 2D laser scans, transformations, positions and plans are transmitted. Each robot keeps its own 3D map. Transferring those 3D maps in runtime over WiFi would result in much higher connection load although it is still possible. The detailed scheme of the data transfer between software modules in the two robot case is provided in Fig. 3.8.

3.5 Laser simulator

The EAPD package requires the data from a laser rangefinder sensor, but it is not necessary to have such sensor physically placed on the robot. The scan can be simulated from the RGBD data, but due to the character of the package and the robot the following requirements must be met:

- Filtering erroneous data - The data from the RGBD sensor are often noisy, especially in glossy environments. The noise is generated from bad reflections from the glass, mirrors or similar objects. It is not so easy to filter because reflections often behave in such a way that erroneous

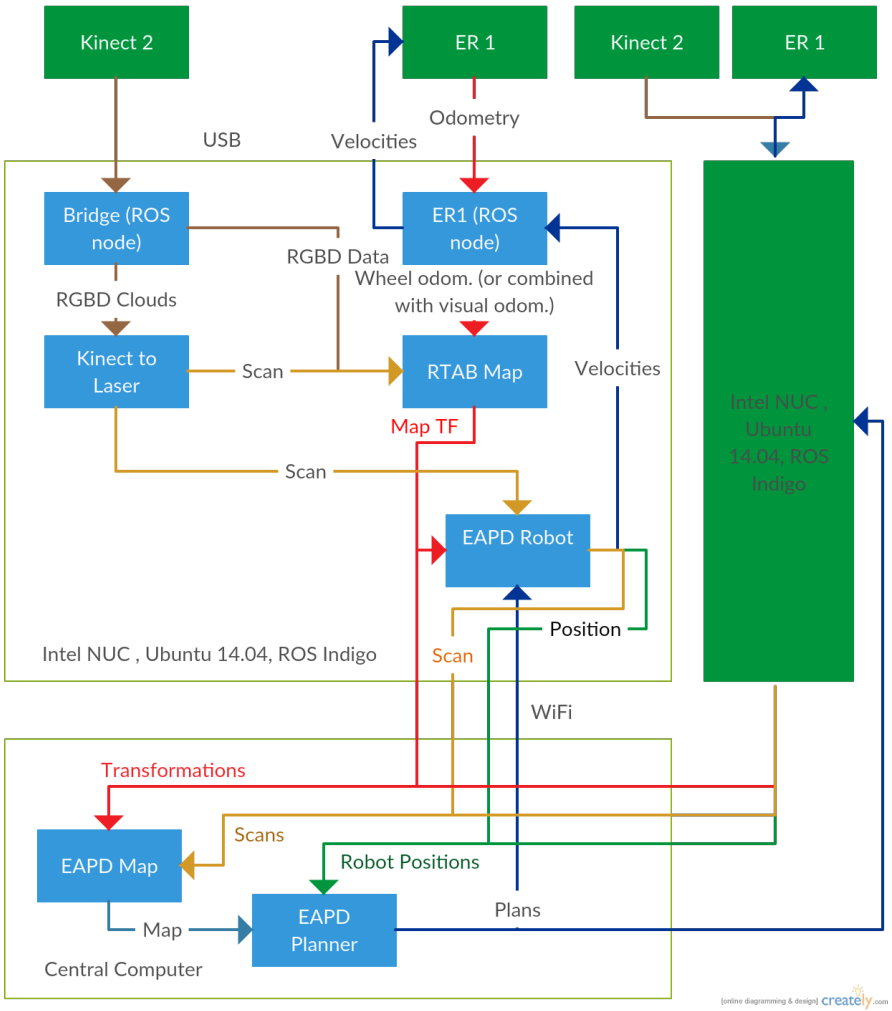


Figure 3.8: The connection between modules in a multi robot case

points create big clusters or even silhouettes of objects behind glass or mirrors.

- Possibility to set up the minimal and maximal height - The simulation must consider all obstacles, which could endanger the robot therefore it is not convenient to use just one row from the depth image.

- Taking advantage of ordered data (or point clouds) into rows and columns - Processing unordered point clouds is much more computationally demanding.

A ROS packages which could provide the simulation of a laser rangefinder already exist. The `depthimage to laserscan` package [20] works well with RGBD images and is very fast. Unfortunately it does not provide a possibility for setting minimal and maximal height. It only provides the `scan height` parameter which sets the number of used pixel rows. That is not sufficient

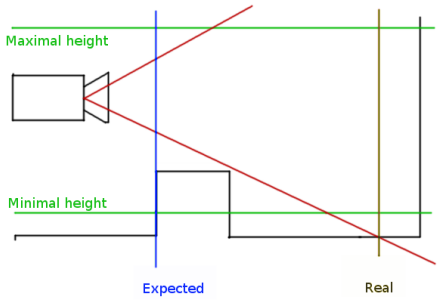


Figure 3.9: Problem with height in pixels. The green lines show required minimal and maximal height. The red lines show how can setting the height in pixels result in incorrect measurements.

because the height in pixels in reality represents constraints in the angle and not in the real height. That causes missing low obstacles close to the camera or detecting floor far from the camera (see Fig. 3.9)

Another possibility is to use the `pointcloud to laserscan` package [21]. It has wider options and provides `min height` and `max height` parameters however neither of the packages provides an option to filter outliers.

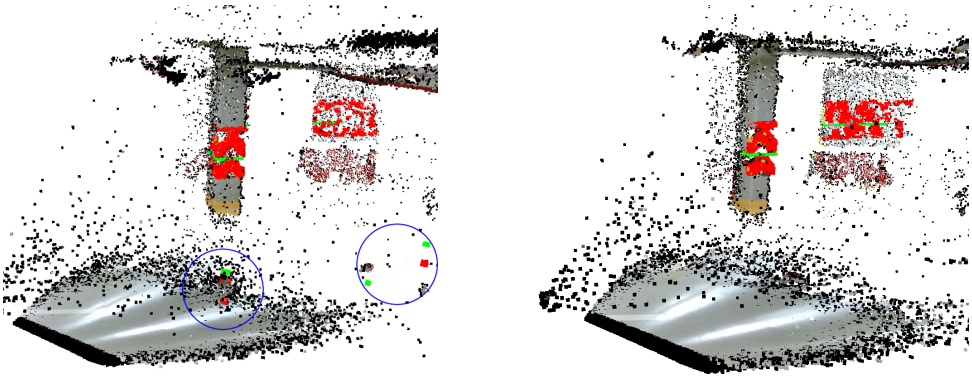
Since both ROS packages do not meet the requirements a new package `kinect to laser` was created. It uses the ordered depth clouds, filters data and has an option for setting the minimal and the maximal height. The data processing consists of several steps:

1. **Point cloud transformation**: The point clouds from the RGBD camera are transformed from the sensor's frame to the robot's frame in order to correct orientation.
2. **Outlier filtration**: The outliers are removed from the data.
3. **Ground and ceiling removal**: The pass through filter is used for elimination of the objects not relevant for the navigation (either passable or above the robot).
4. **Scan simulation**: A laser scan is simulated from the filtered point cloud and published.

The particular steps are described in the following sections.

3.5.1 Point cloud transformation

The point cloud from the RGBD camera is not in the robot's transformation frame. The transformation between the robot's and camera's frames can be non-zero, because the sensor must not necessarily be positioned on the axis of robot's rotation, so the cloud must be transformed to the robot's frame.



(a) : The filtering using 8 neighbors (threshold $T = 6$) does not remove 3 erroneous clusters

(b) : The filtering using 24 neighbours - 5x5 square (threshold $T = 14$) removes the erroneous clusters.

Figure 3.10: Filtering comparison. The red colour represents the points that are not removed.

3.5.2 Outlier filtration

The Point Cloud Library provides several methods for filtering outliers. The problem is that those methods do not take advantage of a point cloud ordered into a 2D grid (RGBD point clouds created from the colour and depth images are ordered in this way). It means that the neighbours of each point are always at an adjacent row or column and it is useless to search for them through the whole point cloud. The point P is marked as an outlier if the number of neighbours N with euclidean distance $|P - N| \leq d$ is lower than a given threshold T.

The computational complexity of this filter is $O(n \cdot m)$ where n is the number of neighbours and m is the number of points in the cloud.

Using only neighbours from an adjacent row and column (4-neighbourhood or 8-neighbourhood) proved to be insufficient especially in areas with glossy objects. Data from the RGBD sensor are very noisy in such areas and the erroneous points create small clusters, which are not marked as outliers by this method.

Filtering using 8 neighbours is fast and sufficient in environments without glossy or other objects which creates lot of noise in the data, but it is better to use more neighbours anyway just to be safe. Unfiltered clusters can force the robot to stop (see Fig. 3.10) because it considers them as obstacles.

A simple method which would not have too large impact on computational complexity was found. The improvement consists of extending the neighbourhood into a square with the length of the edge equal to $l = 2k + 1$, where k is a given parameter ($k \in \mathbb{N}$). The length l must be an odd number so that the point P can be in the centre.

The pseudo code of the presented filtering method is in Algorithm 2.

Algorithm 2 Organized point cloud filtering

1: **for** $P \in C$ **do**
2: Select neighbours N
3: $X = \sum_N x(N,P)$, where $x(N,P) = \begin{cases} 1, & \text{for } \|N-P\| \leq d \\ 0, & \text{otherwise} \end{cases}$
4: **if** $X < T$ **then**
5: Mark P as an outlier.
6: **end if**
7: **end for**

The parameters for this method depend on the environment. The safest option is to use 5×5 square (24 neighbours) and $T = 20$. This ensures that the filtering works even in environments with glass and mirrors, but reduces the precision of the scan. In the environments without objects with high reflectivity only 3×3 square (8 neighbours) and $T = 5$ can be used which is faster and does not significantly reduce the precision of the scan. The recommended distance threshold in both cases is $d = 0.01m$.

3.5.3 Ground and ceiling removal

The ground and ceiling is removed using a pass through filter (from the PCL library). The filter removes all points with z coordinates outside the specified interval. The result is a point cloud with removed floor and ceiling. This method has one weakness. It requires the sensor to be oriented parallel to the floor. If this condition is not met the filtering may not be successful in removing floor or ceiling especially in the places far from the sensor.

The problem is solved by lowering the range of the sensor to a value under 8 meters and by placing the sensor parallel or facing slightly upwards. Setting the floor height to 0.1 meters provides the sufficient tolerance for small errors in sensor orientation.

The orientation of the sensor can also be measured and published as a static transformation between the RGBD sensor frame and the robot base link.

3.5.4 Scan simulation

The point cloud is processed so the last step focuses on the main goal of the whole algorithm which is generation of a laser scan message from the point cloud data. That is achieved by simulating the scan.The following method works on ordered point clouds as well as non-ordered ones.

the parameters of the scan are found in the first step:

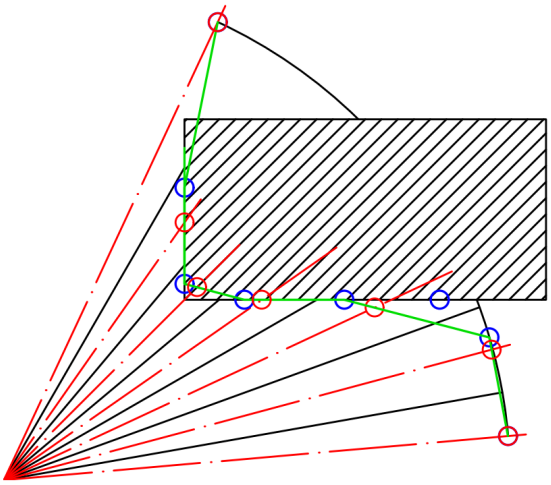


Figure 3.11: Laser scan simulation. The areas between black lines are segments (see Sec. 3.5.4). The simulated beams are the red lines. The points from the cloud are blue, the polyline connecting the points closest to the origin is green and the simulated scan points are red.

Horizontal field of view The minimal and maximal angles about z axis are found by searching for minimal and maximal values in angles of points in the cloud. In open space or outdoor environments without large number of obstacles is not enough points so the difference between minimal and maximal angle can be too small. In such cases the default field of view of the Kinect 2 sensor (70 degrees) is used.

Angle increment The angle incrementation between the virtual beams of the laser scan. It is calculated by dividing the field of view by the required number of points in the scan (usually 128 - 512).

The angular distance between the columns of points from the RGBD camera does not match the angular distance from the simulated laser rangefinder and it is not constant so the whole cloud is sampled in such way that each sample corresponds to one laser beam and the closest point from each sample is selected. The selected points are connected in order to create a polyline. Finally a ray shooting is used in order to find the distances measured by each virtual beam (see Fig. 3.11).

3.6 3D SLAM

The planning algorithm runs in a 2D polynomial map. Since an RGBD camera is used, it would be great to create a 3D map of the environment. There are several available RGBD SLAMs so the first task is to select the most fitting one.

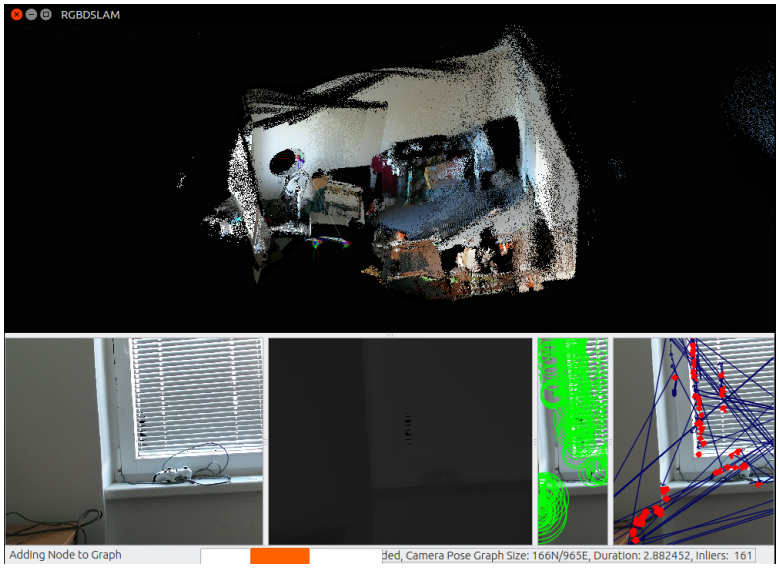


Figure 3.12: RGBD SLAM used with a hand-held Kinect2 sensor in a small room.

3.6.1 RGBD SLAM (Freiburg)

The Rgbdslam[22][23] provided in the ROS is now obsolete (the last version is for ROS Fuerte). The new version (Rgbdslam v2) is available on Github and supports ROS Indigo.

Rgbdslam divides the trajectory estimation into front-end and back-end. The front-end focuses on detecting key points in the visual image, extracting descriptors and matching with the previously extracted ones. The relative transformation is calculated using RANSAC (Random sample consensus).

The g2o framework [24] is used in the back-end. It is an extensible graph optimizer, which can be applied to a wide range of problems. The global optimization of the graph is very useful especially in the cases when a loop-closure happens. The map is represented using the OctoMap framework, which is a Voxel grid managed in a tree structure [23].

Fig. 3.12 shows an example of a test of Rgbdslam in a small room using a hand-held Kinect 2 sensor. In this short experiment a problem with the duration of the map update appeared. Even in this small area the duration increased over 3 seconds on Intel NUC [25] with the default parameters.

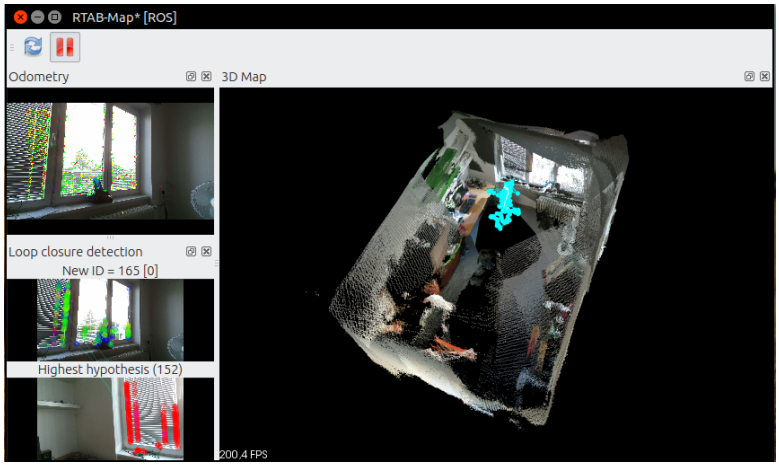


Figure 3.13: RTAB-Map used with a hand-held Kinect2 sensor in a small room.

3.6.2 RTAB-Map

The RTAB-Map is an RGB-D Graph SLAM library with a global Bayesian loop closure detector. That approach uses a Bayesian filter for evaluating loop closure hypotheses over previous images. The likelihood required by this filter is computed by using quantizing the visual words (SURF features). If the loop closure is accepted (enough inliers) the link is added to the graph. The transformation between images is computed using RANSAC. The TORO (Tree-based network optimizer) approach is used for the optimization of the graph (with poses as nodes and transformations as constraints). When a loop closure is found the update can propagate throughout the whole graph and correct the map [19].

RTAB-Map is distributed as a ROS package and it is supported on ROS Hydro, Indigo and Jade. It can be used with a hand-held RGBD or a stereo camera as well as with a camera and a laser rangefinder placed on a robot.

RTAB-Map also allows to use an odometry as an initial guess. If it is not available it is possible to make an estimate of the position using a separate visual odometry node (provided in RTAB-Map package), however that estimation may fail due to the low number of detected features. Using the odometry from wheels (or combination of the two) is therefore more safe and robust.

Transformation from a map to odometry and a complete 3D map are published through the ROS topics. RTAB-Map offers a wide range of different parameters.

The test of RTAB-Map in a small room using a hand-held Kinect2 sensor with a visual odometry is shown in Fig. 3.13. The duration of the map update in this small room was up to 1 second on Intel NUC [25] with default parameters, but it can be tuned to run faster as long as the map is small.

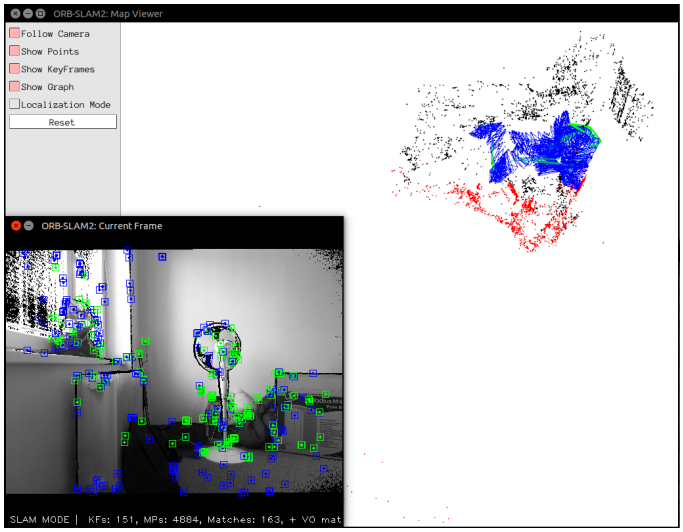


Figure 3.14: ORB SLAM 2 used with a hand-held Kinect2 sensor in a small room.

3.6.3 ORB-SLAM 2

ORB-SLAM 2 is a SLAM library for monocular, stereo and RGB-D cameras. It computes the camera trajectory and a sparse 3D reconstruction. The loop detection and re-localization is done in real-time. [26]

The biggest advantage of this approach (in comparison to those mentioned above) is its duration of the update. It is by far the fastest approach. It is able to process the update in tenths of second on Intel NUC (Sec. 4.4)

A problem appeared during the short test in a small room (Fig. 3.14). In the area with smaller amount of visual features a large error (around 30 degrees in angle) in localization basically made the map unusable. This was tested several times and the map was never created correctly.

3.6.4 Comparison

Rgbdslam v2	RTAB Map	ORB-SLAM 2
- Very slow updates of the map	Faster than Rgbdslam v2, slower than ORB-SLAM 2 but it can provide a fast visual odometry	+ Very fast updates of the map
+ Can re-localize after looking at known position	Problems with re-localization, but it can use the wheel odometry	+ Can successfully re-localize after being lost for a long time, but also becomes lost more often than the others
+ Well readable map, also published through ROS	+ Well readable map, also published through ROS	- The map is not very readable (for people)
+ All important data is published through ROS topics	+ All important data is published through ROS topics & services	- No data is published through ROS topics
	+ Can utilize the odometry provided by the ER1 robot	
+ Wide range of parameters	+ Wide range of parameters	- Low range of parameters

Table 3.2: Comparison between SLAMs

The Rgbdslam is not selected mainly for its long duration of the update and because it is not able to incorporate wheel odometry.

The ORB-SLAM is very fast which would be a big advantage, but it would need to be modified in order to publish map and transformations through ROS and even then there are still issues with the errors in the map (Sec. 3.6.3).

As can be seen from the Table 3.2 the best option is RTAB Map - mainly for the possibility of using the odometry from ER1 robot (possibly combined with the visual odometry) and publishing all required data through ROS topics. It is also highly customizable and faster than Rgbdslam.

3.7 Odometry

The RTAB-Map needs an odometry in order to work properly. There are two options: wheel odometry and visual odometry.

Wheel Odometry

The wheel odometry provided by the ER1 node is generated by measuring the orientation of each wheel and estimating the position change. The advantage of this approach is that it is almost unaffected by the surrounding environment, the odometry does not get lost and it provides reliable data.

The problematic case is when the wheels are slipping. That immediately causes huge error, which is often impossible to recover from.

The main disadvantage of this approach is the drift of this odometry, especially in rotation. Very small errors in the wheel orientation together with some wheel slipping can in time cause large errors in both position and orientation.

Visual Odometry

The visual odometry can be obtained using an application from RTAB-Map package. The biggest advantage of this approach is a very small drift (in comparison to the wheel odometry) and it can not be affected by the wheel slipping.

The biggest disadvantage is the need for a sufficient number of visual features in order to calculate the new position. If a number of detected features is small, the position can not be calculated and the application just sends zeros. Once it get lost, it will usually not start working again[3] so it needs to be reset. The reset would however cause problems in the RTAB-Map itself and the visual odometry thus cannot be used on its own.

Combining odometries

An Extended Kalman Filter (EKF) can be used to combine the readings from both odometries to provide a reliable and precise position. The ROS application `Robot Pose EKF`[18] is designed exactly for this purpose.

That application can handle missing data from one source so the outage of visual odometry would not be a problem. The only problem is that the visual odometry from RTAB-Map package sends zeros in the case of insufficient number of features. The EKF is therefore modified to treat the zero messages in the same way as the missing ones.

The problem with resetting the visual odometry is solved here, because the EKF can use the wheel odometry in the meantime.

[3] The visual odometry can recover from being lost, but the robot would need to look at the features detected in the last correct position so the problem would need to be propagated to the planner.

3.8 Initial transformation

When the EAPD is running on multiple robots in a real environment, the knowledge of the initial transformation between the robots and some fixed point (it can be the start position of one of the robots) is required in order to create a valid map. The simplest way to acquire such transformation is by placing the robots on predefined positions or by measuring their distance and orientation. Such method however complicates and prolongs each experiment and relies on the precision of the placement.

Due to these reasons a different approach is selected - calculating the transformations by comparing RGB and depth images from the sensors on robots assuming that all the robots see almost the same scene. Continuous transfer of such images would make high load on the wi-fi network, but in this case only one RGB and one depth image from each sensor is required.

The transfer of these images is managed by a new ROS application (Fig. 3.15). It acquires images from one measurement on each robot, uses the RTAB-Map visual odometry for calculation of the transformation between the robots and repeatedly publishes that transformation. The RTAB-Map visual odometry node can be stopped once the transformation is computed so that it does not make a load on the CPU.

Although the application supports only the transformation between two robots it can easily work with three or more robots by running several instances where each instance provides a transformation between a pair of robots.

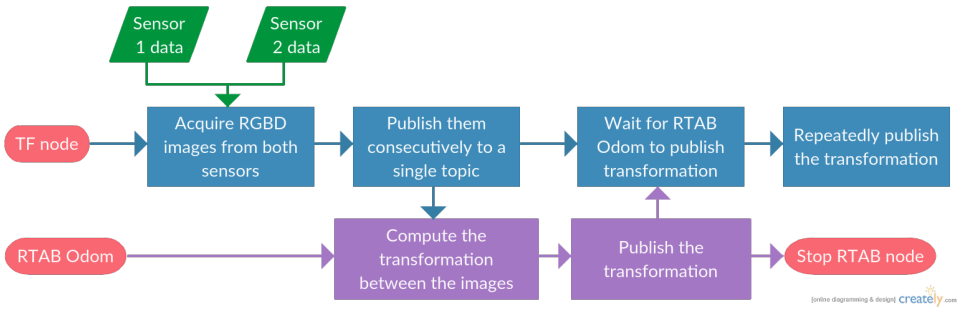


Figure 3.15: A diagram representing the computation of the initial transformation between two robots.

Chapter 4

Robot, sensors and hardware

This chapter focuses on the hardware used for the exploration experiments. It briefly covers the description of the robot and the selection of the RGBD camera, laser rangefinder and on-board computer.

4.1 Robot

The robot is constructed using the Evolution Robotics ER1 Personal Robot System [17]. It is a kit which allows creating a personalized robot using provided parts and aluminium items.

The ER1 robot has already been constructed so the main goal is to modify it in order to fulfil the following requirements:

- Add a larger battery for powering both Mini PC and RGBD Camera.
- Mount holders for PC and sensor.
- Ensure that the sensor is positioned as high as possible without endangering the stability.
- The added construction should be easily removable and modifiable.

A small module made of aluminium items is constructed. It provides holders for RGBD camera, Mini PC, battery, and enough space for placing all cables (see Fig. 4.1).

The whole module can be easily mounted on top of the ER1 robot as can be seen in Fig. 4.2. Everything on the module is powered from its own battery so the only cable connecting it to ER1 is the USB used for the control of the robot. The position on top of the robot is also very convenient as the RGBD camera is placed in a sufficient height (0.5m).

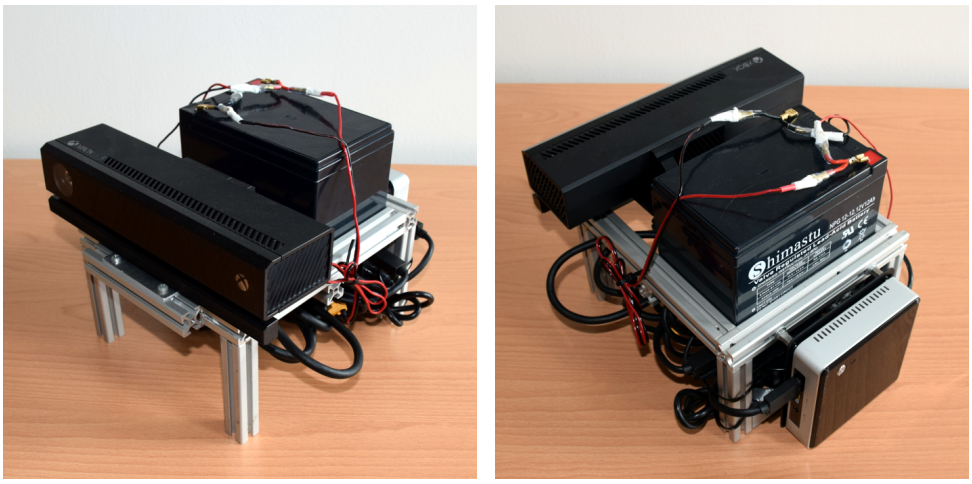


Figure 4.1: The module with sensor and PC for ER1 robot.

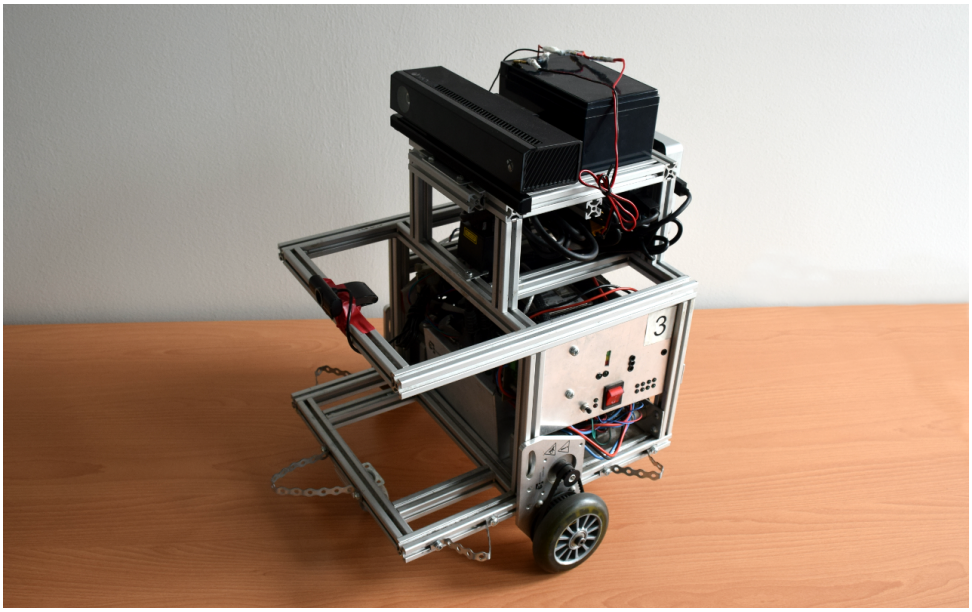


Figure 4.2: The ER1 Robot modified for the purpose of this project.

4.2 RGBD Camera

There are several different RGBD cameras on the market and the following are considered for this work:

ASUS Xtion PRO

The Xtion sensor from ASUS is very small and does not require external power. It is powered from the USB and it's consumption is below 2.5 W. It uses the PrimeSense Light Coding Technology which acquires the depth image by illuminating the scene with patterns in IR light, measuring the deformation of that pattern and triangulating the depth of the scene [27]. It's depth resolution is 640x480.

The Xtion however has several disadvantages. A very big problem is its range, which is only between 0.8m and 3.5m. The second problem is the absence of RGB camera[1]. The field of view is also low - only 58 degrees [28].

The usage of the PrimeSense technology is problematic in the outdoor environments. The emitted IR light is almost invisible in the amount of light coming from the sun so the sensor can not provide reliable depth image.

Microsoft Kinect

The first version of Microsoft Kinect sensor also uses the PrimeSense Light Coding technology so it has the same problems with broad daylight. It's depth resolution is 320x240, but it has better range than Xtion (0.4m - 4.5m). It's field of view is only 57 degrees [29].

It is not as lightweight as Xtion and it can not be powered over USB cable.

Microsoft Kinect for Xbox ONE (Kinect 2)

The second version of the Kinect sensor replaced the PrimeSense Light Coding with a time of flight technology. That significantly increased the accuracy especially in outdoor environments where the original Kinect was not able to work properly in broad daylight.

The new version is still affected by strong sunlight but it can provide reasonable data for a short range, see experiments in Chapter 6.

It's depth resolution is 512x424 and it has a 1080p RGB camera. The range is also better. Although officially it is still 0.5m - 5m [4], the experiments show that the new Kinect provides accurate data approximately from 0.3m to 12m. An important improvement is also the field of view which increased to 70 degrees [29].

This sensor is chosen as the best option mainly for its time of flight camera and increased range. The power requirements are met by installing a 12V battery with sufficient capacity on the robot (Sec. 4.1).

[1] Another version - Xtion PRO Live has RGB camera, but it is still limited in its range.

4.3 Laser Rangefinder

The EAPD package needs the data from a laser rangefinder in order to create a polygonal map. It is however not necessary to use a real laser rangefinder when an RGBD camera is available. The laser data can be simulated by processing RGBD images as described in Sec. 3.5.

4.4 On-board computer

The experiments with Wi-Fi load (see Sec. 5.2) and computational requirements of RTABmap (see Sec. 5.1) proved that the best idea is to use a small computer with sufficient computational power directly on the robot for the complete processing of the RGBD data and 3D map. The biggest advantage is that the robot can run on its own and it does not depend on another central computer. Using a remote computer for processing the data would also reduce the range the robot can go, because the robot would need very strong Wi-Fi connection to the central computer.

The on-board computer runs the RTABmap node and publishes only the 2D data required for EAPD and optionally a 3D map over Wi-Fi. The transfer of the raw (or even compressed) RGB and depth images over Wi-Fi is very problematic.

The Intel NUC mini computer is selected, because it is very small, has a processor with sufficient power - dual core mobile Intel Core i5 processor (1.6GHz - 2.7GHz with Turbo), can be equipped with a sufficient amount of RAM and a solid state drive [25].

That computer is also very economical. Its idle power consumption is only around 7W, which increases to 33W under heavy load. It can be powered together with the Kinect 2 from a single 12V battery [30].

Chapter 5

Power and Connection Requirements

The point clouds from RGBD cameras usually contain around 50 000 - 300 000 points and the rate is around 10-30 Hz. Processing such amount of data in real time is very demanding on the computational power and the connection so it needs to be tested first.

5.1 Computational Requirements

The first tests focused on the computational requirements of the RTAB-Map library which creates the biggest load on a CPU compared to the EAPD and other nodes that processes data. Those tests were done on Kinect 1 sensor (The Kinect 2 was not available at that time).

The Kinect was directly connected to the notebook (Intel core i7 - 4 physical cores 2.4 GHz, 8 virtual cores, 8 GB RAM, Nvidia GT 650M).

RVIZ

The first test focused on computer load when the Kinect data were only visualized in ROS RVIZ tool without further processing. The frame rate was in both cases (compressed / uncompressed) around 30 FPS. The processor load was 19% for compressed and 15-17% for uncompressed data.

RTAB-Map

This test focused on the requirements of the RTAB-Map library. The RTAB-Map update rate was set to 1 Hz and the visual odometry was turned on.

After the start (when Kinect was not yet moved), the RTAB-Map node consumed approximately 20% of processor time (13% calculation, 7% visualization).

When one room was mapped (4x3m, lots of features), the RTAB-Map node consumed 25-30% of processor time and about 700 MB RAM.

The real processor load was always by 4-10% higher due to some other system and ROS processes (37% when the room was mapped).

It is worth mentioning that the RTAB-Map does not really use a multi-core processor so the load was distributed only on one or two cores therefore it cannot really use more than some 30-50% of processor time (on quad core machine).

Conclusion

The i7 notebook should be able to manage mapping with 2 robots. The mapping with 3 or 4 robot may also be possible. Some processor load can be decreased by shutting down visualization and the mapping should be able to run even though some messages are dropped. The problem is that in this case the mapping was running only on 1 Hz frequency. In reality it would be better to update map much faster (at least more than 2 Hz) because otherwise the robots can get so far or turn so much that the RGBD image can not be properly incorporated into the map due to the low overlay of images.

5.2 Data Transfer Over Wireless Network

The RGBD cameras generate a large amount of data and the tests mainly focused on whether it would be possible or convenient to transfer RGB and depth images over wi-fi and process them on a central computer (so called "Remote mapping"). That would allow using a mini computer with low computational power on-board (FOXCONN NanoPC is used for the tests).

The tests were done using Kinect 1 sensor.

RVIZ tests

There seems to be a problem with data trasfer. When only the raw depth image is processed the transferred data rate fluctuates between 300 KiB/s and 1 MiB/s. The image is however very laggy and it has only 0.5 - 2 FPS (when the FPS is higher - 2, the transfer rate is also higher - 1MiB/s).

When only the depth images are published and they are compressed the transferred data rate is again between 300 KiB/s and 1 MiB/s, but the image has 5-8 FPS and is delayed by 1s.

When the RGB data is added (compressed), the transferred data rate stays around 500 KiB/s and 1 MiB/s, but the FPS drops to 1-4. The connection becomes quite unstable. After 3 - 10s the image stops refreshing even though there is still about 1 MiB/s activity on the network.

When the refreshing breaks, the console on the NanoPC shows repeatedly:

"Device timed out. Flushing device. Starting a 3s RGB and Depth stream flush. Stopping device RGB and Depth stream flush."

The computer load on the NanoPC fluctuates between 40 and 55%.

RTAB-Map tests

When the RTAB-Map node is started, data transfer rate is around 800 KiB/s, but the mapping is not able to work.

Connection speed

The internet speed test showed approximately 15 Mb/s (1.88 MiB/s) for both download and upload, which means that the network capacity is higher.

When the connection speed between the notebook and the NanoPC was measured directly, it was only about 1.1 MiB/s, which was strange, because this connection should not be affected by the bottleneck from an internet provider (the notebook also has higher internet speed 2.2 MiB/s on its own).

Note: All these tests were done in the environment, where only 1 - 2 active wi-fi networks were present so no overlapping problems should have appeared.

Conclusion

Under such circumstances the mapping is not possible via wi-fi by transmitting the data from Kinect sensor. Even with compressed data it would require about 10 MiB/s (80 Mbps) connection for smooth data transfer with 30 FPS.

Lowering the FPS to 3-8 in order to achieve a stable connection is possible, but it is safer to use more powerful PC directly on the robot, process the Kinect data there and publish only 2D data (optionally 3D map). Low FPS also affects the visual odometry node (Sec.3.7) which normally updates about 10 times per second. The advantage is also that the number of robots is not limited by the computational power of the central machine (each robot processes its own data) and the robots can explore larger areas, because the WiFi does not need so strong signal for transmission of 2D data.

5.3 Testing Intel NUC computer

The Intel NUC mini computer was selected as on-board PC (Sec. 4.4). It is much more powerful than the NanoPC and it has low power consumption. The NUC is able to process RTAB-Map, but with the CPU load at 95-100% even when running at 1Hz frequency.

Increasing the frequency is possible (the update phase is shortened while loosing focus on precision) and the RTAB-Map node can run at 2Hz, but as the map grows, the update duration increases.

Chapter 6

Kinect sensor in outdoor environment

The Microsoft Kinect for Xbox one sensor (Kinect 2) uses a time of flight method for building the depth image which is the most significant change from its first version. The first version did not work in sunlight but the second version is generally more precise and it has longer range so it is possible that it may be able to work outdoor even in a bright day.

An experiment in the outdoor environment is done to verify that assumption. The environment is presented in Fig. 6.1. It contains many objects at different distances from the sensor:

- A barrel 1m from the sensor.
- A chair 2m from the sensor.
- Flowers 3m from the sensor.
- Several trees and other objects 4 - 15m from the sensor.

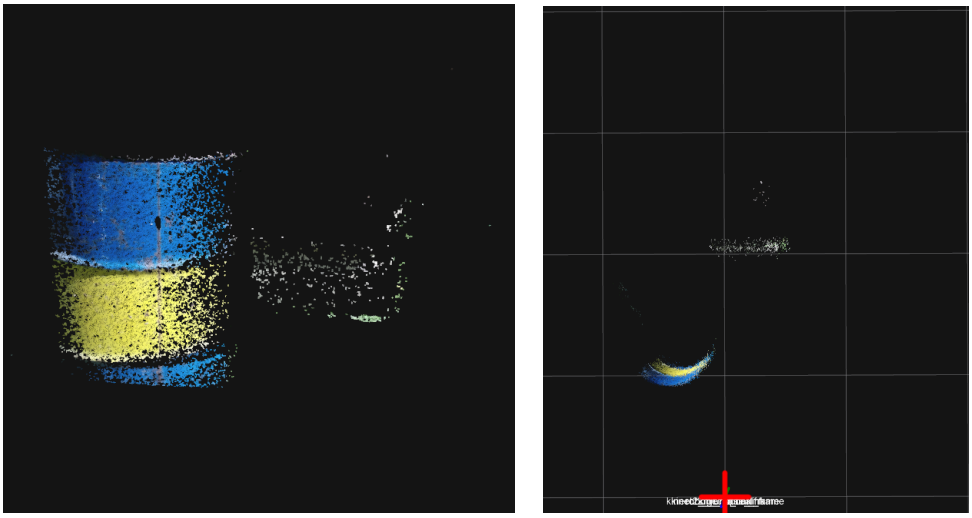
The first experiment is done in a very bright sunlight at 1 PM (May 7, 2016). The results (Fig. 6.2) show that the data are very noisy and only the barrel at a 1m distance is represented in an acceptable quality. The front part of the chair at 2m distance is still visible, but with a large amount of missing points.

The range of the Kinect 2 sensor in a very bright day is approximately 1.2 - 2.4 meters depending on the material of the detected obstacle and many other factors (whether a part of that obstacle is in the shadow, whether the sun shines directly on the sensor and on the reflectivity of the object's surface).

Running RTAB-Map SLAM in a direct sunlight is nearly impossible as it can not effectively localize due to the limited range and large amount of noise.

The same area is also scanned when the most of the scene is covered in a shadow from a building, but the sunlight is still bright so even the areas covered in shadow are illuminated (see Fig. 6.3).

Figure 6.1: Reference outdoor environment with objects at different distances.



(a) : The depth cloud

(b) : Visible objects in a 1m grid from above

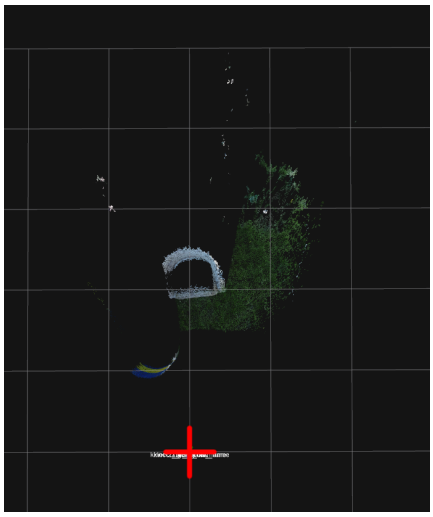
Figure 6.2: Kinect 2 data in sunlight

The range increased to 3.5 - 6m under these circumstances and the majority of objects within 3m range is well represented with almost no noise or outliers. RTAB-Map SLAM is able to run, but the localization is worse than in the indoor environments.

The last experiment with the sensor is done again in the same environment, but this time in the evening when the scene illumination is much lower (see Fig. 6.4).

The sensor correctly detected objects in 12m range and the RTAB-Map works with similar precision as indoors.

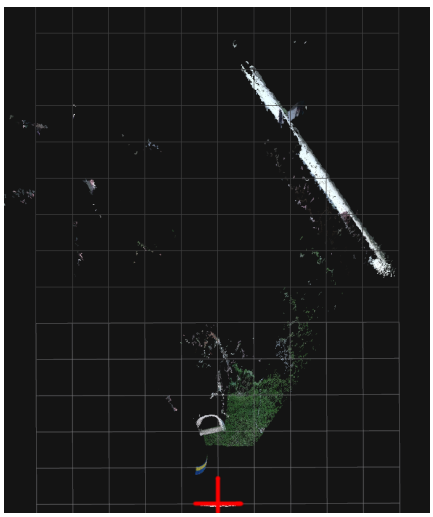
(a) : The depth cloud



(b) : Visible objects in a 1m grid from above

Figure 6.3: Kinect 2 data in a sunny day with most of the scene covered in a shadow from a building

(a) : The depth cloud



(b) : Visible objects in a 1m grid from above

Figure 6.4: Kinect 2 data in the evening

Chapter 7

Experiments with a single robot

Most of the features can be tested on a single robot because there is no need for multi robot exploration in order to test localization or map quality. It is also useful for determining the effect of issues that can appear in a real environment, for example noise in the data, low precision of localization or the presence of reflective objects.

7.1 Indoor offices

Several indoor experiments are done in the BLOX building located in Prague - Dejvice. It is a typical office environment. The robot starts the exploration in the resting area and the kitchen. That area contains a sufficient number of visual features and it is open space where the robot can easily avoid the obstacles.

On the other hand it presents a challenge, because it is a very dynamic place with a lot of moving people and the kitchen is separated from the resting area by a glass wall. The glass on its own is very problematic in combination with a Kinect sensor as it creates a lot of erroneous data due to the reflections (this is the case when the modification presented in Sec. 3.3.4 is necessary).

As can be seen in Fig. 7.2 the exploration of the kitchen and a part of the resting area was successful. The robot even managed to successfully go through a door in the glass wall. That wall can not be seen in the 3D map, but its location is in the place where the colour of the floor changes.

The problem is with corridors leading from the resting area. They are painted with white colour and there are not enough features (only doors to the offices) so the SLAM starts to drift there and the map becomes less and less reliable. A short experiment in the corridor area can be seen in Fig. 7.3. The robot managed to successfully explore the first two corridors but the map started to drift and the loop closure was not successful. The drift becomes more and more apparent as the map expands. The time of the update continuously increases and the incorporation of new measurements is

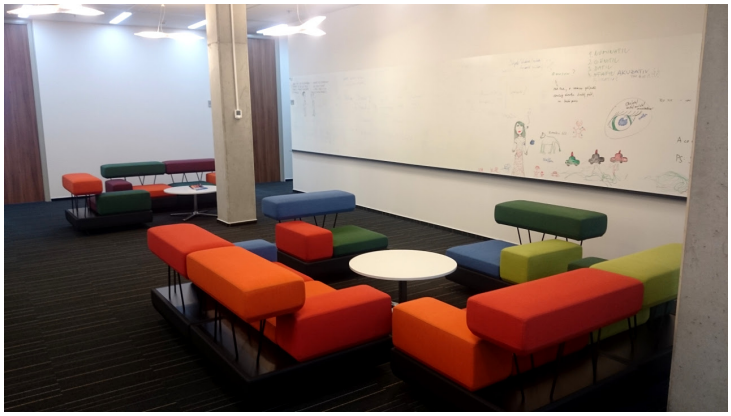
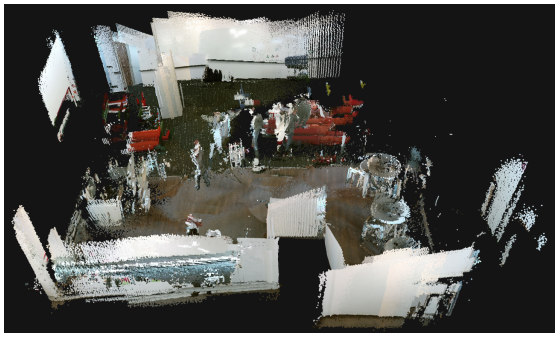


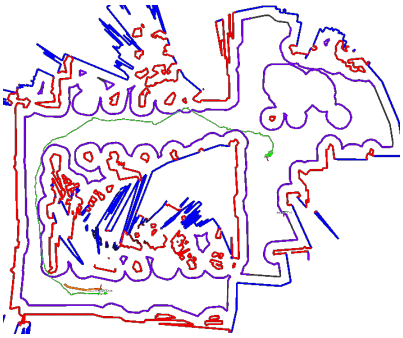
Figure 7.1: The office environment in the BLOX building in Prague Dejvice.

(a) : The polygonal map (obstacles - red, frontier - blue, expanded - violett / grey) with a path of the robot (green).

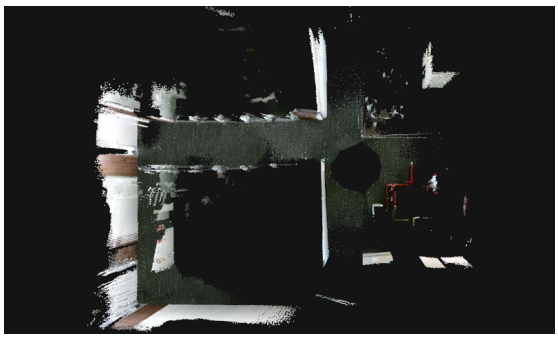


(b) : The 3D map of the kitchen and resting area.

Figure 7.2: Exploration in offices



(a) : The polygonal map (obstacles - red, frontier - blue, expanded - violett / grey) with a path of the robot (green).



(b) : The 3D map of the corridors.

Figure 7.3: The second experiment in the office area.

Figure 7.4: The ground floor of the National Technical Library in Prague used for experiments with exploration.

more and more complicated.

The movement of people is handled reasonably well. Even though the people are sometimes placed into the map (Fig. 7.2b), if there are not too many of them the localization does not get lost and the exploration can continue. The robot even successfully avoids the people in front of it. The quality of the map slightly decreases because the moving person can be placed into it many times so the area looks crowded.

The polygonal map (Fig. 7.2a) looks very noisy especially in the right bottom section. That section represents the tables and chairs in the kitchen and as the robot considers only legs as the obstacles (it can pass under the table) the map contains many small circles (sometimes connected). The presence of people also adds small obstacles into the map. The second polygonal map (Fig. 7.3a) looks better, because it consists mostly of direct corridors with few doors and windows.

Even though the map does not look very nice (partly also because of the modification presented in Sec. 3.3.4), its quality is sufficient for the exploration as it clearly defines the passable areas and the planner can find a non collision path.

7.2 Technical library

The ground floor of the National Technical Library in Prague (Fig. 7.4) is a large area suitable for exploration experiments. In comparison with the indoor offices this place is much bigger and does not contain long featureless corridors.

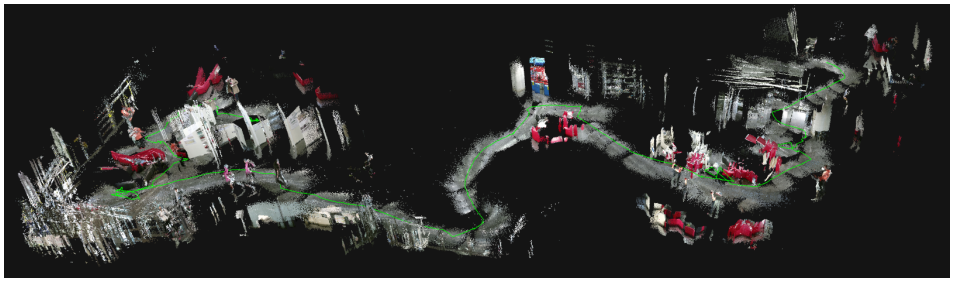


Figure 7.5: The path of the exploration in the National Technical Library in the 3D map.

Figure 7.6: The detail of the library 3D map.

On the other hand with a glossy floor and transparent walls there are much more reflections than in the office environment. The movement of people is also much more frequent.

The mapped area is large (Fig. 7.5) but the exploration was stopped before it could finish. That happened due to a risk of collision with a glass panel that robot was not able to detect. Other experiments were made but usually they had to be stopped due to the long duration of the RTAB-Map update on large maps which caused problems with localization.

The map is reasonably accurate but it still contains several erroneous areas. The first problem is the glass wall (Fig. 7.5 on the left). It is not properly matched and placed to the map probably due to the large number of reflections. Such error could potentially have large impact on the map but in this case the SLAM managed to re-localize using the exposition placed opposite the wall (Fig. 7.6 on the left).

The second problematic part is the area in the middle in Fig. 7.5 which is basically empty. The SLAM must rely on the wheel odometry which can drift from the correct position. As the path through this empty area is not

Figure 7.7: The underground garage in BLOX building (Prague Dejvice)

too long, the damage on the map is very small.

The glossy floor is very challenging because the Kinect sensor sees the reflections in a similar way as a standard camera - the reflected object is mirrored under the glossy floor. This mirror image is then placed into the 3D map under the floor (see the reflections of the boards in the left of Fig. 7.6) Fortunately the objects under the floor do not affect the exploration and the polygonal map but some glossy objects perpendicular to the floor can greatly reduce the map precision.

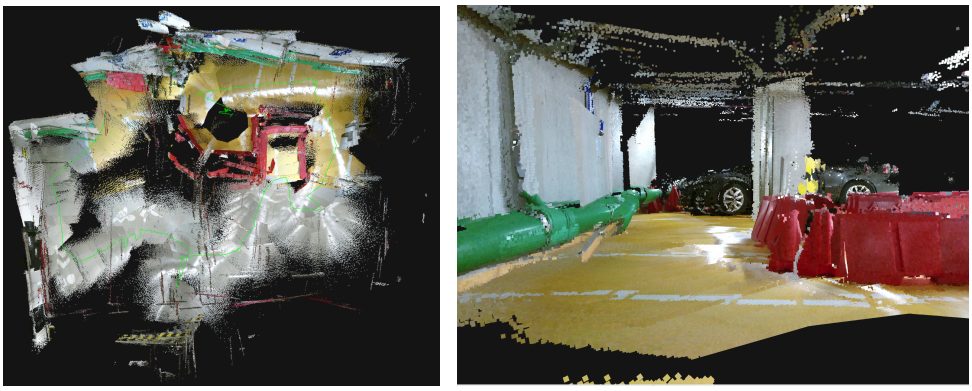
The robot successfully managed to avoid people moving throughout the area mostly without a significant decrease in the precision of localization. Those people can be seen in Fig. 7.5 and 7.6.

7.3 Underground garage

An underground garage is selected as another place for experiments. One smaller floor of the garage in the BLOX building is selected. The advantage of using a smaller garage floor is that the car traffic is lower so the experiment does not need to be repeatedly stopped due to ongoing cars.

Such environment has an advantage in a small number of people passing through so it is almost static. There are also barriers (see Fig. 7.7) that can be carried and used as obstacles (provided that the route for a car is not blocked).

The problematic part is as usually the glossy floor and mainly the aluminium covered parts on the roof. Due to these the sensor data are so noisy that the noise often creates large clusters that are hard to filter (for example



(a) : 3D map of the garage **(b) :** Detail in the 3D map

Figure 7.8: Exploration in underground garage

Fig. 3.10 in Sec. 3.5.2).

The beginning of the experiment was successful. The robot managed to get out of the space enclosed by obstacles and to provide a good map of the place (Fig. 7.8b). The problem appeared when the robot went into the open space (Fig. 7.8a in the bottom). The number of features significantly decreased and an error in orientation caused that the rest of the map was not properly aligned to the first part. The loop closure was not successful so the error remained in the map. Other experiments in the same area usually had similar problems. The localization slowly drifted and sometimes one unsuccessful update destroyed the map.

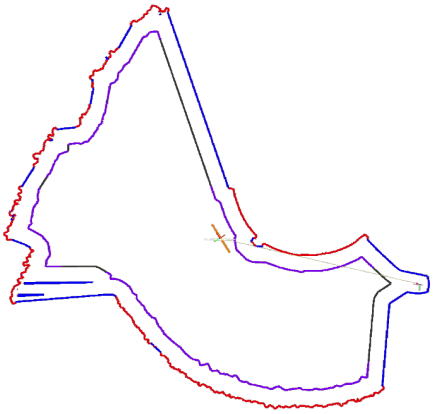
7.4 Outdoor environment

The experiments in outdoor environments are significantly affected by the weather. The Kinect 2 sensor is able to work well as long as there is not too much sunlight (i.e. overcast). The situation in the direct sunlight is worse, but even than the sensor can provide data that can be reliable in a reduced range.

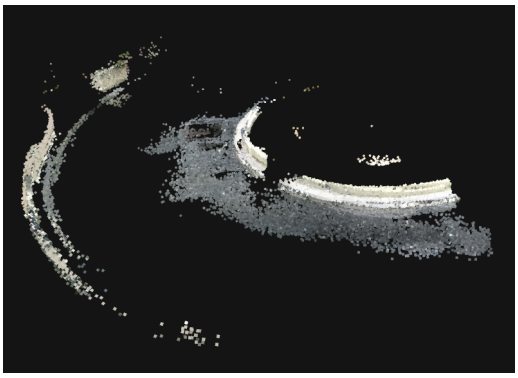
The outdoor environments present another challenge. It is very problematic to use the SLAM on RGB-D data due to its more open space and reduced range of the sensor. The number of features is usually very low so the only reliable position can be acquired from the wheel odometry which drifts in time.

An example of an outdoor environment is presented in Fig. 7.9. The robot started going around the fountain, but the asphalt road was not entirely flat. The environments with uneven terrain are problematic for the robot. The SLAM can handle small changes in the height even if it is set up to use only 2D position but the problem is with the small wheels. The robot is unable to

Figure 7.9: Outdoor environment near BLOX building (Prague Dejvice)



(a) : Polygonal map



(b) : 3D map

Figure 7.10: Exploration in an outdoor environment

go through such environment as the wheels usually start to slip or get stuck. The robot managed to map only a very small area during the experiment (Fig. 7.10) and then it got stuck because of small bump on the road.

7.5 Real environment issues

Many different issues appeared during the exploration experiments in the real environment. This section provides a short description of the most significant problems that can not be solved by simple modifications of the exploration framework.

Field of view

One of the biggest constraints is the field of view of the Kinect sensor which is only 70 degrees [29]. This causes a big problem for the SLAM especially when the robot is rotating. Even though the angular velocity is limited it can still be around $0.3 rad/s$. Lowering the velocity even more

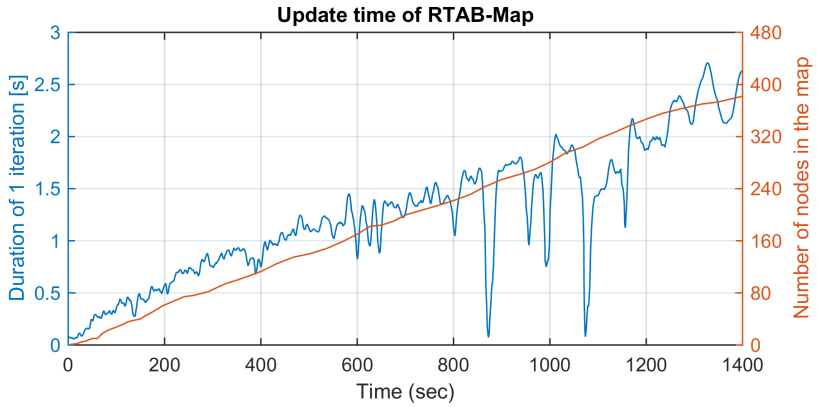


Figure 7.11: The time of the RTAB-Map's update and the number of nodes in the map depending on the length of time the mapping runs (test done on Intel NUC computer (Sec. 4.4).

would make the exploration too slow.

When the exploration is running for a long period of time, its refresh duration can be longer than 2 seconds. In that time the robot can rotate by 35 degrees. Due to the low field of view the SLAM can only match the features from roughly 50 percent of the image.

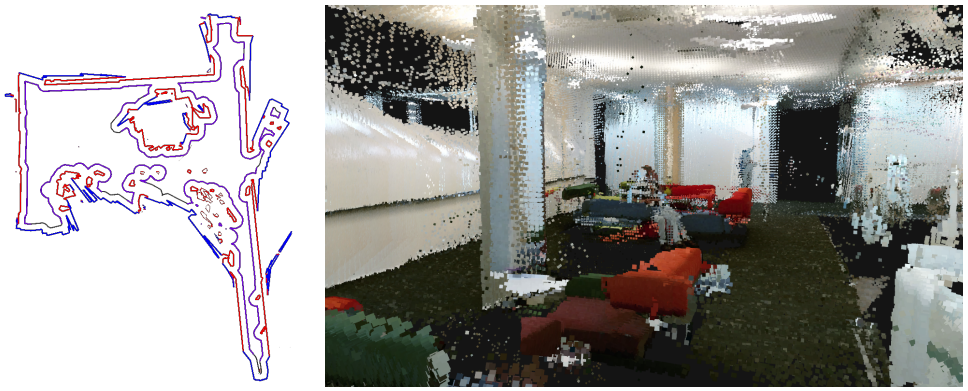
There is an option for limiting the computation time of the RTAB-Map but at the cost of lower quality and even then the duration of the update gradually increases as the map grows. The SLAM must therefore rely on the data from the odometry.

The RTAB-Map visual odometry is very fast and in most cases it can compensate for the long duration of the RTAB-Map update, but it often fails due to the low number of features so the only working source of the position data is the wheel odometry which slowly drifts.

Duration of update

Duration of the RTAB-Map's update is directly linked to the problem with field of view. The time of the update gradually increases. In the experiment presented in Fig. 7.11 the update time was set to $0.5s$. As can be seen in the graph, this threshold is crossed in approximately $200s$ after the start and the time of the update continues to grow as the map expands.

An experiment with a more powerful computer was done in order to verify that the results would be better with lower time of the update. Kinect was connected to the notebook described in Sec. 5.1 which is more powerful than Intel NUC and it was used for manual mapping of the office area. The results were much better as can be seen in Fig. 7.12. Both 3D and polygonal map have higher quality even though the whole mapping was done without the wheel odometry (only with the visual one).



(a) : Polygonal map **(b) :** Detail in 3D map

Figure 7.12: Mapping with more powerful computer

Reflections

Reflective surfaces in the environment cause large amount of noise and also silhouettes of the reflected objects. Most of that noise can be filtered (Sec. 3.5) but the silhouettes remain in the map.

The worst example of such reflective surface is a glass wall (for example 7.1). It generates a large amount of noise and also points from both the glass wall and the objects behind. The robot can not always detect such wall and it may try to get through.

On the other hand, the wall is detected in most cases in a form of few points which are sufficient, because the offset map connects them and creates a single obstacle so the robot does not try to go through.

Not enough features

The quality and success of the whole exploration highly depends on the number of detected visual features. As the number of features decreases, the SLAM has bigger problems with matching scenes and may result in error such as inserting a new scan under an incorrect angle. Such error can be sometimes repaired by a loop closure, but often remains in the map uncorrected.

Chapter 8

Multi-robot experiments

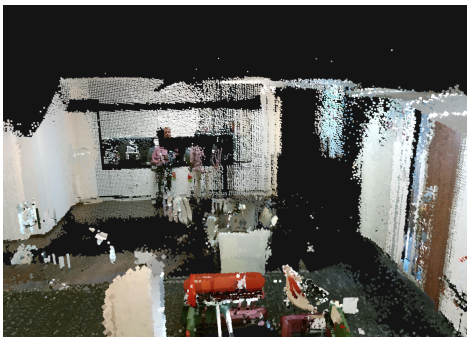
The framework has been also tested by doing an experiment with a simultaneous mapping using more than one robot. The previous experiments uncovered many issues that can appear in real environments, but without an experiment with multiple robots there is no way how to test the matching of the separate 3D maps or adding measurements from different robots into the polygonal map.

A full multi-robot experiment has not been possible due to the problems with a control unit for ER1 robot. One unit has broken during the single robot experiments and it had to be replaced by the unit from the second robot, therefore the second robot has not been operational any more and no other ER1 robot has been available. Without the second robot the experiment was slightly modified so that only one robot is used and the other robot is replaced by a hand-held RGBD camera.

This modified experiment is still sufficient for the multi-robot testing of the framework because both 3D map matching and incorporation of scans to polygonal map is done in a same way for both hand-held camera and an autonomous robot.

The experiment is done in the office environment presented in previous chapter (Fig. 7.1). The initial position for both robots was in front of the kitchen (in the centre of Fig. 8.1) where both robots faced the kitchen. The orientation in a similar direction was required because the algorithm for determining the initial transformation between robots (3.8) relies on matching images from both sensors. The transformation was successfully determined as presented Fig. 8.1. The kitchen area was mapped by both robots (Figs. 8.1a and 8.1b) and was smoothly connected (Fig. 8.1c).

The autonomous robot (first) started the exploration by turning around and then by going left from the kitchen and covered most of the resting area (Fig. 8.2a). The hand-held camera (simulation of the second robot) was moved in the opposite direction so that it covers a different part of the area. The hand-held camera was connected to a notebook and power supply so it was not possible to move it far from the initial position. The kitchen and



(a) : A map from the first robot

(b) : a map from the second robot

(c) : The connected map

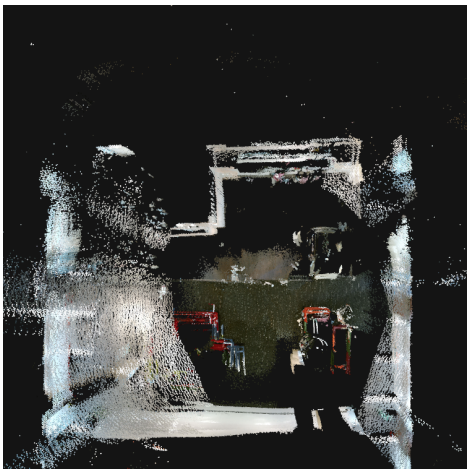
Figure 8.1: Connection of maps from two robots

a part of the corridor behind was covered (Fig. 8.2b).

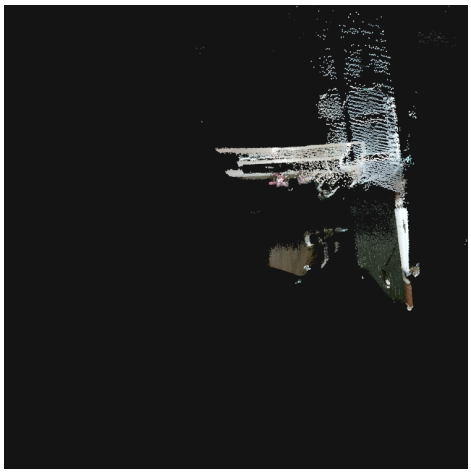
The top view of the connected map is presented in Fig. 8.2c. It is clearly visible from that view that although the kitchen is connected well, the corridor behind is not ideally matched with the rest of the map, but the error is not too large. This happens because of small inaccuracies in both maps mainly due to the reasons presented in Sec. 7.5.

The polygonal map (Fig. 8.2d) looks noisy and erroneous and does not cover the whole area. The 3D map covers slightly more space, because the 2D map assumed lower range of the sensor (5m instead of 10m). This is beneficial, because the robot tends to explore more thoroughly and it covers some areas in more detail. The presence of noise in the polygonal map is mainly because of two reasons. One is the constant movement of people through the area (few of them can be seen even in the 3D map - Fig. 8.1) and the other one is a glass wall between the kitchen and the resting area. The glass wall creates a lot of reflections causing a large amount of erroneous data.

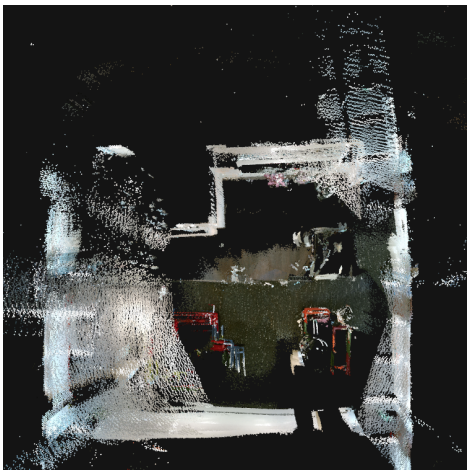
Some parts of the map also seem wrong but they are correct because most



(a) : Map from the first robot



(b) : Map from the second robot



(c) : Connected map

(d) : Polygonal map

Figure 8.2: Map from two robots top view

of the tables are actually placed into the polygonal map as four small round obstacles - legs.

This experiment (similarly as most of the single-robot experiments) had to be stopped sooner than the whole area was covered. The main reason for this was the gradual increase of the computational complexity (Sec. 7.5) that eventually makes the duration of the update so long that the new scenes are not properly incorporated into the map, because the robot already sees completely different objects than in a previous iteration.

Chapter 9

Conclusion

The Exploration in a Polygonal Domain framework by T. Juchelka presented in Sec. 2.2 has been modified and updated in many ways to be more stable, reliable and better for use in real environments (Sec. 3.3). The most significant changes were rewriting the framework in order to use transformations, adding the possibility for setting the boundaries, improving the algorithms for updating the map and for generating the offset map (which is a polygonal alternative to obstacle expansion in occupancy grids). Many other changes, bugfixes and improvements were made as explained in Sec. 3.3.

In order to transfer the exploration framework to the real environment it had to be connected with other libraries (Sec. 3.4) especially with SLAM because the framework does not provide localization. A 3D SLAM approach working with RGB-D camera was selected (Sec. 3.6). The available implementations were compared and the RTAB-Map [19] was chosen as the best suiting for this work. The advantage of using a RGB-D SLAM was creation of a 3D map of the environment. The exploration framework and the SLAM were connected using the Robot Operating System [8] while other libraries and applications were added or created, for example an application for a simulation of a laser scan from RGB-D image (Sec. 3.5) or for computing the initial transformation between the robots (Sec. 3.8).

The hardware for this task was selected according to the requirements arising from the software implementation. It consisted of extending the ER1 robot by adding a new module with the Microsoft Kinect 2 RGBD camera [4], Intel NUC mini computer [25] and a battery for providing sufficient power (Chap. 4).

As this work focuses on the practical aspect of the multi-robot exploration, the power and connection requirements of the implementation were tested (Chap. 5). Continuous transfer of RGB-D images would create high load on the wireless network so it was decided that each robot computes its own 3D map and the maps are matched at the end of exploration. The 2D polygonal map is is managed centrally and it is used for planning the robots because 2D data are much smaller for transmission.

One part of this work was also testing the Kinect 2 RGB-D camera in outdoor environments. The previous version of Kinect sensor did not work in broad sunlight but the new version started using a time of flight method for creating a depth image. The goal was to test if the sensor is now able to handle the sunlight. The tests show that it is still strongly affected by the illumination of the scene. A direct sunlight reduces the range of the sensor to $1.2 - 2.4m$ but it can work reasonably well if the sky is clouded or the detected objects are in a shadow. The range in such cases increases to $3.5 - 6m$. The sensor works still much better in indoor environments where the range is up to $12m$ (Chap. 6).

The whole exploration was tested first on one robot (Chap. 7). The experiments show that the selected method works well for a short time exploration, but it is not ideal for a long time usage because the duration of the RTAB-Map update increases as the map gets bigger (Fig. 7.11). This problem together with a small field of view of the Kinect, low amount of features in some areas and reflections from some objects makes the exploration unstable for long time running. The localization loses its precision because the duration of the update on a big map is so long that the new sensor reading has very few common features with the existing map - the robot moved to far in that time.

The whole framework was also tested with two robots (Chap. 8) where the second robot was simulated by using a hand-held RGB-D camera due to the damaged control unit. The matching of the maps works well (see Fig. 8.2) but due to the problems explained in the previous paragraph the localization is less and less precise as the size of the map increases so only smaller areas can be successfully mapped.

This work managed to show that the polygonal representation can be used in the real world and that it can work well in many different environments. It is fast, memory efficient and reliable, but the results of the tests with the whole framework are not so good especially because of the problems with the low field of view of the Kinect sensor and the computational requirements of the RTAB-Map SLAM which are too high for a mini computer on-board the robot, especially when the map covers large area. Replacement of the Kinect by a stereo camera with a wider field of view together with the usage of more powerful on-board computer could significantly improve the results of the exploration and increase the size of the area that can be explored before the time of the update increases too much. Parallel processing of the SLAM on a graphic card may have also large influence on the time of the update and thus improve the quality of the whole map.

Appendix A

Bibliography

[1] T. Juchelka, "Exploration algorithms in a polygonal domain," diploma thesis, CTU in Prague, FEE, Dept. of Cybernetics, 2013.

[2] T. Andre, D. Neuhold, and C. Bettstetter, "Coordinated multi-robot exploration: Out of the box packages for ROS," in *Globecom Workshops (GC Wkshps), 2014*, pp. 1457–1462, Dec 2014.

[3] ROS.org, "About ROS," 2013. [Online; accessed 1-April-2016] Available from `http://www.ros.org/about-ros/`.

[4] Wikipedia.org, "Kinect for Xbox One," 2016. [Online; accessed 1-April-2016] Available from `https://en.wikipedia.org/wiki/Kinect_for_Xbox_One`.

[5] M. Kulich, T. Juchelka, and L. Přeučil, "Comparison of exploration strategies for multi-robot search," *Acta Polytechnica*, vol. 55, no. 3, pp. pp. 162–168, 2015.

[6] M. Dakulovic, S. Iles, and I. Petrovic, "Exploration and Mapping of Unknown Polygonal Environments Based on Uncertain Range Data," *Automatika – Journal for Control, Measurement, Electronics, Computing and Communications*, vol. 52, no. 2, 2011.

[7] B. Yamauchi, "Frontier-based exploration using multiple robots.," *Proc. of the Second International Conference on Autonomous Agents*, pp. 47–53, 1998.

[8] ROS.org, "Robot Operating System Documentation," 2016. [Online; accessed 2-April-2016] Available from `http://wiki.ros.org/`.

[9] J. W. Durham and F. Bullo, "Smooth nearness-diagram navigation," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 690–695, Sept 2008.

[10] Wikipedia.org, "Dijkstra's algorithm," 2016. [Online; accessed 1-May-2016] Available from `https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm`.

[11] ROS.org, "TF Package Summary," 2015. [Online; accessed 2-April-2016] Available from `http://wiki.ros.org/tf`.

[12] A. Johnson, "Clipper - an open source freeware library for clipping and offsetting lines and polygons," 2014. [Online; accessed 2-April-2016] Available from `http://www.angusj.com/delphi/clipper.php`.

[13] ROS.org, "Rosbuild," 2012. [Online; accessed 10-May-2016] Available from `http://wiki.ros.org/rosbuild`.

[14] ROS.org, "Catkin," 2015. [Online; accessed 10-May-2016] Available from `http://wiki.ros.org/catkin`.

[15] H. Martin, J. Blake, and K. Machulis, "Libfreenect," 2016. [Online; accessed 1-April-2016] Available from `https://github.com/code-iai/iai_kinect2`.

[16] T. Wiedemeyer, "IAI Kinect 2," 2016. [Online; accessed 1-April-2016] Available from `https://github.com/code-iai/iai_kinect2`.

[17] M. Humphries, "Evolution Robotics ER1 Personal Robot System," 2003. [Online; accessed 17-April-2016] Available from `http://www.geek.com/hwswrev/hardware/er1/`.

[18] ROS.org, "Robot Pose EKF," 2012. [Online; accessed 11-April-2016] Available from `http://wiki.ros.org/robot_pose_ekf`.

[19] M. Labbé and F. Michaud, "Online global loop closure detection for large-scale multi-session graph-based slam," in *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pp. 2661–2666, Sept 2014.

[20] ROS.org, "Depthimage to laserscan," 2013. [Online; accessed 30-April-2016] Available from `http://wiki.ros.org/depthimage_to_laserscan`.

[21] ROS.org, "Pointcloud to laserscan," 2015. [Online; accessed 30-April-2016] Available from `http://wiki.ros.org/pointcloud_to_laserscan`.

[22] F. Endres, J. Hess, J. Sturm, D. Cremers, and W. Burgard, "3-d mapping with an rgb-d camera," *IEEE Transactions on Robotics*, vol. 30, pp. 177–187, Feb 2014.

[23] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burgard, "An evaluation of the rgb-d slam system," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pp. 1691–1696, May 2012.

[24] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "G2o: A general framework for graph optimization," in *Robotics and*

Automation (ICRA), 2011 IEEE International Conference on, pp. 3607–3613, May 2011.

[25] Intel.com, "Mini PC—Intel® NUC Kit NUC5i5RYK," 2016. [Online; accessed 19-April-2016] Available from `http://www.intel.com/content/www/us/en/nuc/nuc-kit-nuc5i5ryk.html`.

[26] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós, "Orb-slam: A versatile and accurate monocular slam system," *IEEE Transactions on Robotics*, vol. 31, pp. 1147–1163, Oct 2015.

[27] Wikipedia.org, "PrimeSense," 2016. [Online; accessed 1-April-2016] Available from `https://en.wikipedia.org/wiki/PrimeSense`.

[28] ASUS, "Xtion PRO Specifications," 2016. [Online; accessed 1-April-2016] Available from `https://www.asus.com/3D-Sensor/Xtion_PRO/specifications/`.

[29] M. Szymczyk, "How Does The Kinect 2 Compare To The Kinect 1," 2014. [Online; accessed 1-April-2016] Available from `http://zugara.com/how-does-the-kinect-2-compare-to-the-kinect-1`.

[30] T. S. Ganesh, "Intel NUC5i5RYK Review: A Broadwell-U UCFF PC for Enthusiasts," 2015. [Online; accessed 19-April-2016] Available from `http://www.anandtech.com/show/8986/intel-nuc5i5ryk-review-a-broadwellu-ucffpc-for-enthusiasts/7`.

Appendix B

CD content

Folder	Content
`/cloud_tf`	ROS package for evaluation of the initial transformation between two point clouds
`/eapd2`	Updated implementation of the Exploration in a Polygonal Domain framework
`/kinect_to_laser`	ROS package for calculation of a virtual laser scan from Kinect data
`/robot_pose_ekf`	Original ROS package for processing odometries with several modifications
`/thesis`	This thesis in PDF format
`/thesissrc`	Latex source code of this document